Cavium Networks OCTEON Plus CN50XX Hardware Reference Manual

Contents of this document are subject to change without notice.

The exact features and specifications may change prior to the V1.0 manual revision.

CN50XX-HM-0.99E PRELIMINARY

July 2008

PUBLISHED BY
Cavium Networks
805 East Middlefield Road
Mountain View, CA 94043
Phone: 650-623-7000
Fax: 650-625-9751
Email: sales@caviumnetworks.com
Web: http://www.caviumnetworks.com

Table of Contents

List of Figures

List of Tables

Preface

In this Preface

List of Chapters

This book contains the following chapters:

25. Signal Descriptions
26. Ball Assignments
A. Cavium Networks-Specific Core Instructions

Revision History

The revision history is shown in Table 1.

Table 1 Revision History

Version	Date	Changes
0.99C	3/2008	• In the Features section of the Introduction chapter, changed “8 million packets per second” to “2 million packets per second”. • **Electrical Specifications, Chapter 22: in Table 22–10, added current information for the miscellaneous I/O interfaces; in Table 22–11, added MII to the mix and removed the separate MII table (Table 22-12); in Table 22-13 (now Table 22–12), adjusted the current values for DDR_PLL_VDD33 and PLL_VDD33.** • In Table 25–3, added text to the description of DDR_PLL_VDD33; in Table 25–15, added text to the description of DDR_PLL_VDD33. • In Appendix B, replaced CN56XX ordering information with CN50XX information.

0.99D	6/2008	• CMB/L2C/LMC, chapter 2:
		  • in Section 2.3.8.1, moved the cross-reference to Section 2.3.9 into Step 2
		  • in Section 2.5, changed the typical value of LMC_CTL1[SIL_MODE].
		• POW, chapter 5:
		  • in Section 5.6, corrected cross-reference to the temporary IQ executable interrupt threshold disable bit
		  • in Section 5.13, updated the descriptions of POW_WQ_INT_THR(0..15)[TC_EN,TC_THR,IQ_THR], POW_WQ_INT_CNT(0..15)[TC_CNT], POW_WQ_INT[IQ_DIS,WQ_INT].
		• PIP, chapter 7:
		  • in Section 7.5, corrected the second figure cross-reference in the definition of WORD1[Len] (refers to Figure 7-1)
		  • in Section 7.9, updated the description of PIP_FRM_LEN_CHK0/1
		  • in Section 7.10, updated description of IPD_CTL_STATUS[PBP_EN].
		• PKO chapter 8:
		  • removed CRC block fromFigure 8–1
		  • in Section 8.4, removed mention of CRC in WORD0[S1] and WORD0[S0]; in the WORD2[Ptr] description , corrected reference to PKO_REG_FLAGS[STORE_BE] (incorrectly referred to PKO_REG_FLAGS[STORE-LE].
		  • in Section 8.9, in PKO_REG_BIST_RESULT, removed the [CRC] field.
		• PCI, chapter 9:
		  • in Section 9.14.2, for PCI_WIN_RD_ADDR, changed [WR_ADDR] to [RD_ADDR] and changed the subdivision of this field from <12:3> and <2:0> to <12:2> and <1:0>
		  • in Section 9.15, expanded the Field Description of NPI_CTL_STATUS[CHIP_REV].
		• Packet Interface, chapter 13:
		  • in Table 13–6, added configuration settings for GMII and MII.
		  • in Section 13.8, updated GMX0_INF_MODE[EN]; updated note related to GMX_RX/TX0..2_STAT* CSRs (removed first or second bullet regarding STATS resources)
		• TWSI, chapter 17:
		  • in Section 17.3, added text about TWSI_CTL[ENAB]
		  • in Table 17–10, updated the Description of [ENAB].
		• SMI, chapter 18:
		  • added text to the first indented bullet following Figure 18–1
		  • modified Figure 18–2 to reflect the same text addition
		  • in Section 18.2, corrected the description of SMI_CLK[SAMPLE] and corrected the typical value for SMI_CLK[SAMPLE, SAMPLE_HI]
		• Cavium Instructions, Appendix A: for the PREF instruction, the hint field values of 2, 3, 8–24, 26, 27 were changed to Reserved.

0.99E	7/2008	• **Electrical Specifications, Chapter 22: added Note following Table 22-3; in Table 22–4, added supply voltage for 300/350/400/500 MHz.**

Related Documentation

For additional information, refer to these documents.

Cavium Networks, Inc.'s OCTEON Network Chip architecture is based on officially licensed MIPS Inc. Technology. For reference to MIPS Technology, you should procure the following books:

Table 2 MIPS Publications

Publication	Document Number
MIPS64® Architecture For Programmers Volume I: Introduction to the MIPS64® Architecture Revision 2.00, June 8, 2003	MD00083
MIPS64® Architecture For Programmers Volume II: The MIPS64® Instruction Set Revision 2.00, June 9, 2003	MD00087
MIPS64® Architecture For Programmers Volume III: The MIPS64® Privileged Resource Architecture Revision 2.00, June 9, 2003	MD00091
EJTAG Specification Revision 3.10, July 5, 2005	MD00047

For additional information regarding MIPS® technology, refer to "MIPS Technologies" on page 37.

Symbols Used

The **Notes** in this manual follow standard ANSI recommendations.

NOTE: This format is used to emphasize important factors.

User Comments

We welcome your comments about this document. You can reach us by e-mail at info@caviumnetworks.com.

Using Cavium Networks PDF Files

Open and view PDF files using the Adobe® Acrobat® Reader application, version 5.0 or later. If necessary, download the Acrobat Reader from the Adobe Systems, Inc. web site at: http://www.adobe.com/prodindex/acrobat/readstep.html

PDF files offer several ways for moving among the document's pages:

To move quickly from section to section within the document, use the Acrobat bookmarks that appear on the left Acrobat Reader window pane. The bookmarks provide an expandable outline view of the document's contents. To display the document's Acrobat bookmarks, press the "Display both bookmarks and page" button on the Acrobat Reader tool bar.

To move to the referenced page of an entry in the document's Contents, List of Figures, List of Tables, or Index, click on the entry itself, each of which is hyperlinked.

To follow a hyperlink to a heading, figure, or table, click the blue text.

To move to the beginning or end of the document, to move page by page within the document, or to navigate among the pages you displayed by clicking on hyperlinks, use the Acrobat Reader navigation buttons shown in this figure:

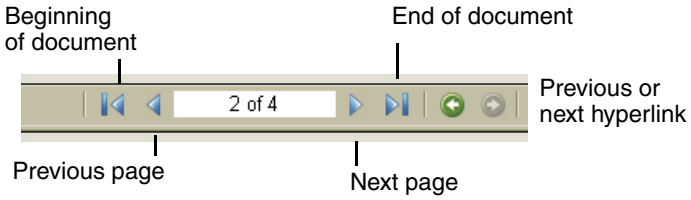


Table 3 summarizes how to navigate within a Cavium Networks PDF.

Table 3 Navigating Within a PDF Document

To Navigate This Way	Click This
Move from section to section within the document.	A bookmark on the left side of the Acrobat Reader window
Move to an entry in the Table of Contents.	The entry itself
Move to an entry in the Index.	The page number
Move to an entry in the List of Figures or List of Tables.	The entry itself
Follow a hyperlink (highlighted in blue text).	The hyperlink text
Move page by page.	The appropriate Acrobat Reader navigation buttons
Move to the beginning or end of the document.	The appropriate Acrobat Reader navigation buttons
Move backward or forward among a series of hyperlinks you have selected.	The appropriate Acrobat Reader navigation buttons

About This Book

The data and illustrations found in this book are not binding. We reserve the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be construed as a commitment by Cavium Networks, Inc.

Cavium Networks, Inc. assumes no responsibility for any errors that may appear in this document. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us at info@cavium.com.

MIPS Technologies

MIPS architecture is the exclusive property of MIPS Technologies, Inc. and is protected under the following trademarks: MIPS64.

MIPS64 is among the registered trademarks of MIPS Technologies, Inc. in the United States and other countries.

Various MIPS RISC processor manuals and additional information about MIPS products can be found at the MIPS URL:

http://www.mips.com

Comments or questions on the MIPS64® Architecture or MIPS documents should be directed to:

Director of MIPS Architecture
MIPS Technologies, Inc.
1225 Charleston Road
Mountain View, CA 94043
or via E-mail to architecture@mips.com.

Contact Us

Cavium Networks
805 East Middlefield Road,
Mountain View, CA 94043
Telephone: 1-650-623-7000
Fax: 1-650-625-9751
Email: info@caviumnetworks.com

Chapter 1

Introduction

OCTEON Plus CN50XX

Features

- Highly-integrated networking security/application processor
- MIPS64® integer instruction set (version 2), highly programmable
- High-performance architecture with up to two cnMIPS™ processor cores
  - CN5010 has one processor core
  - CN5020 has two processor cores
- Dual control- and data-plane support
- Standalone (i.e. self-bootable) or device support
- Core frequency 300–700 MHz, producing up to 2.8 GOPS
- Up to 2 million packets per second
- Excellent performance/watt
- Hardware cryptographic (MD5, SHA-1, SHA-256, SHA-512, DES/3DES, AES, KASUMI, modular exponentiation, and Galois field multiplication) and CRC acceleration
- Hardware packet processing acceleration
- Hardware work queueing, scheduling, ordering, and synchronization
- Hardware TCP acceleration, including checksum and timer
- Integrated packet interface with MAC: RGMII/GMII/MII I/O interfaces
- Integrated USB 2.0 MAC/PHY
- Integrated TDM/PCM interface
- High-bandwidth on-chip memory system including a 128KB eight-way set-associative L2 cache
- Fully coherent memory system
- 32/16-bit DDR2 DRAM interface up to 667 MHz data rate
- 32-bit PCI (66 MHz) interface
- Secure on-chip key memory
- Cryptographic random-number generator

Overview

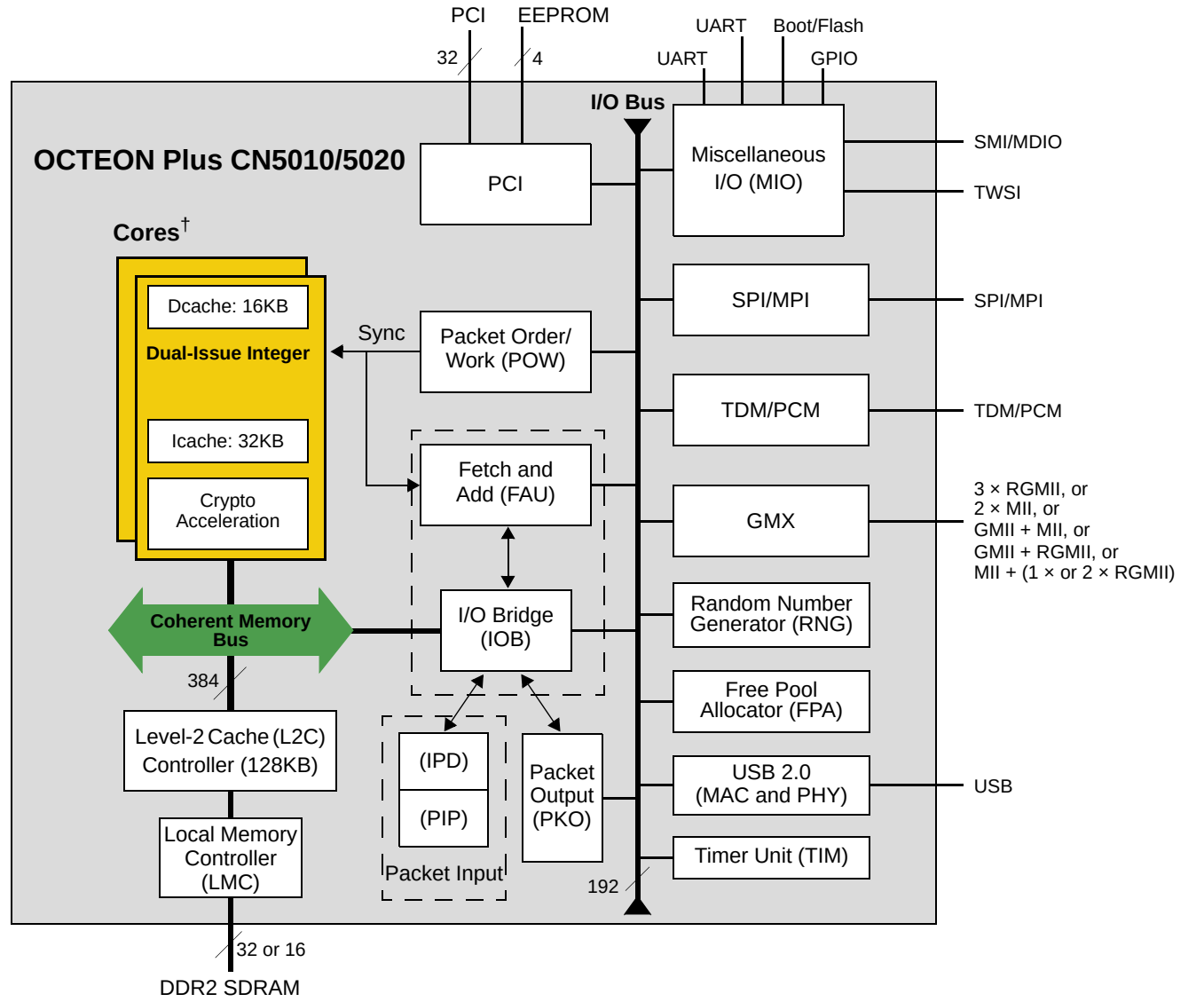
The Cavium Networks OCTEON Plus CN50XX Network Services Processors are targeted at intelligent networking, control-plane, storage, and wireless applications. The OCTEON Plus CN50XX is targeted for many applications, but is particularly well-suited for the following applications and standards:

- Broadband Gateways
- Triple Play Gateways
- Media Vault / NAS appliances
- 802.11 Access Points
- Unified Threat Management Appliances
- WiMAX Stations
- In-line packet processing
- Firewall (FW)
- Virtual Private Network (VPN)
- Intrusion Detection / Prevention (IDS)
- iSCSI
- IPSEC
- SSL
- Network Attached Storage (NAS) Appliances
- Intelligent Storage Routers
- 802.11 Wireless Security Protocols
- Security Appliances
- Secure Network Interface Card (NIC)
- Secure Services Card

The OCTEON Plus family has a base architecture that is the same for all members. Software written for the OCTEON Plus family can scale from top to bottom.

All OCTEON Plus family members use the same cnMIPS™ CPU core and the same on-chip architecture. The cnMIPS core, hereafter referred to as the core, is a simple and high-performance dual-issue implementation of the standard MIPS64® integer (version 2) instruction set. OCTEON Plus hardware preserves the same software interface to all on-chip hardware units between all family members.

Figure 1–1 shows a block diagram of the CN50XX chip. The blocks on the left half of the chip (i.e. the CPU cores, the coherent memory bus (CMB), the level-2 cache (L2C), and the DRAM controller) implement an on-chip multiprocessor and coherent shared-memory system. The right half of the chip contains the I/O bus and I/O interface blocks together with configurable I/O and coprocessing hardware accelerator blocks. These two halves of the chip connect through the I/O bridge (IOB), creating a system-on-a-chip suited for packet-processing, networking, and security applications.



†CN5010 has one cnMIPS™ processor core; CN5020 has two cnMIPS™ processor cores.

Figure 1–1 CN50XX Block Diagram

The family is a scalable architecture that covers a wide range of performance and I/O configurations. OCTEON Plus family members support the following industry-standard I/O interfaces: DDR2 DRAM, PCI 2.3 32/66, RGMII, GMII, MII, TDM/PCM, SPI/MPI, USB, and UART. CN50XX also implements a proprietary boot-bus interface.

Figure 1–2 shows the full CN50XX pinout.

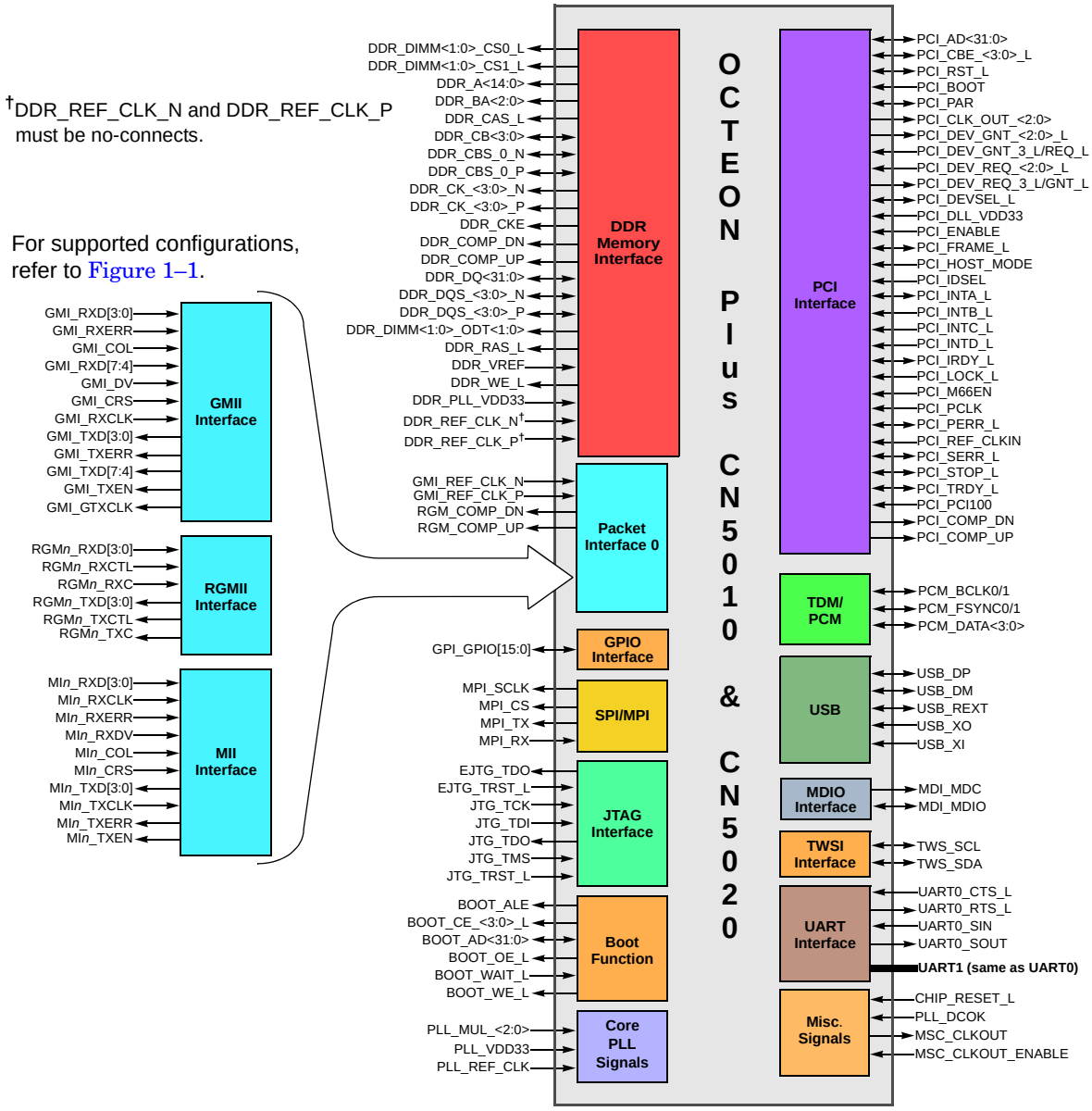


Figure 1–2 CN50XX Signals

1.1 Principles of Operation

This section lists some of the principles of the CN50XX architecture.

1.1.1 CPU Cores

The CN50XX's cnMIPS cores are full-functionality and high-performance MIPS64® integer (version 2) implementations. These cores directly support industry-standard C/C++ (and other) programming environments.

These cores and the CN50XX system-on-a-chip also have all the necessary requirements, including TLBs, to boot and run full- functionality operating systems.

These core features allow for feature-rich usages of the CN50XX, and also are essential to simplify programming tasks. The CN50XX has the high performance necessary for "data-plane" applications, yet avoids the code/data size restrictions present in some competing "data-plane" processors.

CN50XX includes many core instructions beyond the standard MIPS64® integer (version 2) instructions. These include instructions to accelerate packet processing, security processing, and memory/cache processing.

There is extensive hardware acceleration on CN50XX, but the CPU cores direct the complicated higher-layer application-specific processing.

1.1.2 Coherent Multicore and I/O L2/DRAM Sharing

Shared main memory (implemented via the level-2 cache and the DRAM) is the primary communication media for bulk transfers in CN50XX. I/O and coprocessor devices DMA packet and other data in/out of this memory.

CN50XX's level-2 cache is shared by all of the CPU cores instructions/data and hardware device DMA accesses. The L2 cache can be partitioned, and can be bypassed on a reference-by-reference basis.

The CN50XX hardware always maintains L2 cache and CPU core data cache coherence with respect to all DMA and other core accesses. The CN50XX maps all addresses into the L2 cache. Caching can be disabled on a reference-by-reference basis in the CN50XX, but not by address.

1.1.3 Core Partitioning

Software and chip configuration can partition the cores to perform different functions. For example, some of the cores may run an operating system while others perform data-plane functions, or different cores may execute different portions of the data-plane services. All cores can also run the same operating system.

1.1.4 Flexible Packet/Control Interfacing

Packets and control information can flow to/from CN50XX via any of the RGMII, GMII, MII, or PCI interfaces.

The RGMII interface supports up to three ports, and the PCI interface supports one port. This means that internally, CN50XX can support up to a total of four ports.

CN50XX hardware efficiently transfers packets via the PCI interface. There is one input and one output port. CN50XX supports all PCI transfer modes and also includes multiple DMA engines, whose features can be used to support per-flow queueing in the PCI host memory.

1.1.5 In-line Packet-Processing Hardware Acceleration

The CN50XX has in-line hardware to offload from the cores all data movement, many common packet-parsing functions, and other important calculations. The in-line hardware completely offloads the work from the cores.

The CN50XX in-line packet-processing hardware units complete the following tasks before a core receives a packet:

- Allocate DRAM buffers to hold the packet bytes.
- Send packet data into these DRAM buffers via DMA operations in a format convenient to upper-level software. This can enable copy-free core software.
- Parse the layer-two/IP packet, checking for common exception conditions. This is done for every ingress packet. The layer-four TCP/UDP checksum checks are included, among many others.
- Perform optional mutual exclusion via a programmable 7-tuple classification.

When software has finished processing the packet, CN50XX in-line hardware performs the following tasks:

- DMA the packet-send command from the selected output queue (and free the available queue space).
- DMA packet data from L2/DRAM. This includes multiple modes to gather non-contiguous packet data.
- Generate the TCP/UDP checksum. (CN50XX can perform this very efficiently; the packet data is read out of L2/DRAM memory only once to both calculate the checksum and send the packet off-chip.)
- Free the DRAM buffers.

1.1.6 Hardware-Assisted Dynamic Memory Allocation/Deallocation

CN50XX contains free pools that contain pointers to available packet buffers. The hardware can automatically allocate and free packet buffers. The queues that the cores use to submit instructions to the various on-chip coprocessors are also dynamically allocated in chunks. The coprocessor hardware automatically frees the memory chunks containing the instructions after it processes the instructions contained in the chunk. The cores can also use the free pools at any time. Refer to “Free Pool Unit (FPA)” on page 41.

1.1.7 Hardware Work Queuing, Scheduling, Ordering, and Synchronization

CN50XX hardware maintains a work queue in an on-chip hardware unit. This structure provides a primary on-chip communication mechanism between the cores and the hardware units on CN50XX. Both hardware and core software can contribute work to the work queue. The core software can request work at any time, which it becomes aware of either through polling or by interrupts. The hardware schedules the work for the cores.

For example, the in-line input packet processing hardware presents an input packet to the CPU cores by creating and submitting a work queue entry to the queue. The core software receives the packet by obtaining the associated work.

CN50XX has many ordering/synchronizations mechanisms available to core software, but one important one is closely integrated with the work queueing/scheduling hardware. This hardware orders / synchronizes based on tag values associated with the packet/work. The in-line packet processing hardware can create the initial tag for a piece of work from up to a 7-tuple hash. Core software can fully direct the subsequent tag changes needed for the packet/work, and is also in control of the initial tag that it creates.

The work-queueing hardware not only synchronizes, but also schedules and deschedules (in conjunction with core software) work based on tag values.

1.1.8 Essential Quality of Service (QoS) Functions Implemented in Hardware

CN50XX has some essential QoS capabilities implemented directly in hardware:

- The hardware has eight separate input work queues.
- The in-line input packet processing hardware can classify packets into one of the eight input work queues using default values, VLAN priorities and IP Diffserv values, configurable on a per-port basis. The system can also directly select the work queue on a per-packet basis.
- The in-line input packet processing hardware may discard an input packet before buffering and presenting it to a core. The hardware implements both a random early discard (RED) algorithm and a threshold algorithm to decide when or if to discard an input packet. The RED-like algorithm can be configured differently for each of the eight QoS levels, and the threshold algorithm can be configured differently for different ports.
- Each output port can be configured to have multiple queues. The queues can be configured with different priorities. The hardware implements both static priority and weighted round-robin.

Remaining QoS functions are implemented by core software on CN50XX. For example, hardware never decides to drop an outgoing packet – only software does.

1.1.9 Security Features

The cores internally include acceleration for the common security algorithms: MD5/SHA-1/SHA-256/SHA-512, DES/3DES, AES (all modes), AES-GCM, KASUMI, CRC (up to 32 bits), and vector multiplication for fast modular exponentiation needed for RSA/Diffie-Hellman operations.

CN50XX contains a number of features for high FIPS-level certification, including: on-chip key memory, key zeroization pin, and restricted PCI host access.

1.1.10 Coprocessor Accelerators

There are also other available coprocessors that have not already been mentioned:

- Timer (TIM): timer support is particularly useful for TCP implementations.
- True Random Number Generator (RNG)

1.1.11 Debug Support

CN50XX has numerous debug features, including:

- Two UARTs for connection to an external console.
- Full core support for the MIPS EJTAG standard, including single-step and an external JTAG interface with one internal TAP controller per core.
- Multicore debug support.
- Visibility into critical hardware scheduling structures.
- Visibility into hardware/software interface structures

1.2 CN50XX System Applications

TBD

1.3 Remaining Chapters

The remainder of this chapter briefly describes the contents of each remaining chapter in this book.

1.3.1 Coherent Memory Bus (CMB), Level-Two Cache Controller (L2C), and DRAM Controller

This chapter describes the components that implement the CN50XX main memory system. This includes a description of the CMB, the internal architecture and configuration registers of the L2 cache and Controller (L2C), and the internal architecture and configuration registers of the DRAM Controller, and the DDR2 pin interface. Refer to “Coherent Memory Bus, Level-2 Cache Controller, DRAM Controller” on page 53.

1.3.2 I/O Bus and I/O Bridge

This chapter describes the components that implement the I/O bus that connects CN50XX I/O devices and coprocessors to the on-chip main memory system. This includes a description of the I/O bus and the internal architecture and configuration registers of the I/O bridge (IOB). Refer to “I/O Busing, I/O Bridge (IOB) and Fetch and Add Unit (FAU)” on page 125.

1.3.3 CPU Cores

This chapter describes the CPU cores that implement the MIPS64® (version 2) integer instruction set, including internal architecture, the CN50XX-specific enhancements and configuration registers, and the interactions between the cores and the rest of CN50XX. Refer to “cnMIPS™ Cores” on page 143.

1.3.4 Packet Order / Work Unit (POW)

This chapter describes the work queuing, scheduling, ordering, and synchronization hardware on CN50XX. This includes the software interface, the internal architecture, and the configuration registers of the Packet Order / Work Unit (POW). Refer to “Packet Order / Work Unit (POW)” on page 205.

1.3.5 Free Pool Unit (FPA)

This chapter describes the CN50XX hardware free pool implementations. This includes the software interface, the internal architecture, and the configuration registers of the Free Pool unit (FPA). Refer to “Free Pool Unit (FPA)” on page 253.

1.3.6 Packet Input Processing/Input Packet Data Unit (PIP/IPD)

This chapter describes the hardware-centralized input packet parsing and DMA capabilities. This includes the software interface, the internal architecture, and the configuration registers of the Packet Input Parsing unit (PIP) and the Input Packet Data unit (IPD). These units write input packets from any/all of the SGMII, XAUI, and PCIe interfaces into CN50XX's main memory, and also create/submit work queue entries for the input packet. Refer to “Packet Input Processing/Input Packet Data Unit (PIP/IPD)” on page 281.

1.3.7 Packet Output Unit (PKO)

This chapter describes the hardware-centralized output packet processing and DMA capabilities. This includes the software interface, the internal architecture, and the configuration registers of the Packet Output unit (PKO). This unit DMAs packets out from main memory through any of the RGMII/GMII/MII and/or PCI interfaces. Refer to “Packet Output Processing Unit (PKO)” on page 335.

1.3.8 PCI Unit

This chapter describes the PCI interface and the hardware that creates and services PCI transactions. This includes a description of all the PCI transactions that CN50XX can create and service, the PCI BARs, the (internal) software interface, the internal architecture, and the configuration registers. The internal architecture includes ports that interface with the CN50XX centralized packet-input and packet-output hardware, DMA engines for transferring data between CN50XX and a PCI device, direct-access mechanisms from externally-mastered PCI transactions, and core direct-access to locally-mastered PCI transactions. Refer to “PCI Bus” on page 365.

1.3.9 Timer Unit (TIM)

This chapter describes the timer unit, which is a hardware mechanism to submit work-queue entries at future times. This includes the software interface, internal architecture, and configuration registers. Refer to “Timer” on page 449.

1.3.10 Central Interrupt Unit (CIU)

This chapter describes the central-interrupt unit (CIU). This includes the software interface and the configuration registers. This unit is a central distributor for core interrupts, external PCIe interrupts, and other controls. Refer to “Central Interrupt Unit (CIU)” on page 459.

1.3.11 Boot Bus Unit

This chapter describes the external boot interface, its software interface, and the configuration registers. When CN50XX self-boots, the boot interface has the (non-volatile, presumably) storage, like flash memories or ROMs, that contain the initial CN50XX boot code. The boot bus can also support other more sophisticated devices such as those that follow the compact flash standard. Refer to “Boot Bus” on page 479.

1.3.12 RGMII/GMII/MII Unit (GMX)

This chapter describes the RGMII/GMII/MII interfaces, and the internal architecture and configuration registers of the hardware RGMII/GMII/MII unit (GMX). This unit primarily interfaces with the centralized packet input and output logic internally. Refer to “CN50XX Packet Interface” on page 509.

1.3.13 TDM/PCM Unit

Refer to “PCM/TDM Interface” on page 569.

1.3.14 GPIO Unit

This chapter describes the external GPIO interface, its software interface, and the configuration registers. Refer to “GPIO Unit” on page 593.

1.3.15 UART Unit

This chapter describes the external UART interface, its software interface, and the configuration registers. Refer to “UART Interface” on page 601.

1.3.16 TWSI Unit

This chapter describes the external TWSI interface, its software interface, and the configuration registers. Refer to “TWSI Interface” on page 623.

1.3.17 System Management Interface (SMI)

The CN50XX system management interface is a standard ethernet MDIO interface. The chapter describes the interface, its software interface, and the configuration registers. Refer to “System Management Interface (SMI)” on page 651.

1.3.18 Random Number Generator (RNG/RNM)

This chapter describes the random number generator (RNG), which includes the internal architecture, the software interface, and the configuration registers. The random number generator creates truly random numbers. Refer to “Random-Number Generator (RNG), Random-Number Memory (RNM)” on page 657.

1.3.19 SPI/MPI Unit

Refer to “MPI/SPI Unit” on page 663.

1.3.20 USB Unit

This chapter describes the USB interface, which is a dual-role device (DRD) controller that supports both host and device functions and is fully compliant with the USB 2.0 specification. Refer to “USB Unit (USB)” on page 675.

1.3.21 Electrical Specifications

This chapter describes the supply voltages, power sequencing, power consumption, and DC electrical characteristics of CN50XX. Refer to “Electrical Specifications” on page 763.

1.3.22 AC Characteristics

This chapter describes the AC characteristics of the CN50XX I/O interfaces. Refer to “AC Characteristics” on page 773.

1.3.23 Mechanical Specifications

This chapter has an overview of the CN50XX physical package (BGA), and thermal requirements. Refer to “Mechanical Specifications” on page 793.

1.3.24 Signal Descriptions

This chapter describes the CN50XX signals in detail. Refer to “Signal Descriptions” on page 799.

1.3.25 Ball Assignments

This chapter describes the position of the I/O pins in the CN50XX physical package. Refer to “Ball Assignments” on page 811.

1.4 Configuration and Status Registers (CSRs)

CN50XX has many CSRs to control and configure the on-chip hardware. Each configuration register can be accessed by one or more of the following mechanisms:

- cores using MFC0/MTC0 instructions
- cores using MFC2/MTC2 instructions
- cores using ordinary load/store instructions to I/O bus physical addresses
- remote EJTAG/JTAG device using the EJTAG/JTAG TAP interface
- remote PCI host using direct PCI configuration accesses
- remote PCI host using direct PCI memory space BAR0 accesses
- remote PCI host using indirect windowed accesses to OCTEON I/O bus physical addresses via the PCI_WIN_WR_ADDR, PCI_WIN_RD_ADDR, PCI_WIN_WR_DATA, PCI_WIN_WR_MASK, and PCI_WIN_RD_DATA BAR0 CSRs.

These configuration registers fall into the following classes:

- **Core coprocessor 0 (COP0) registers**. These CSRs are duplicated on each core, and can only be accessed by software running on the local core. These are covered in Section 4.11.
- **Core coprocessor 2 (COP2) registers**. These CSRs are duplicated on each core, and can only be accessed by software running on the local core. These are covered in Section 4.2 and Appendix A.
- **Core EJTAG registers**. These CSRs are duplicated on each core, and can only be accessed by software running on the local core. These are covered in Section 4.12.
  - All CN50XX core EJTAG registers are 64-bits, and must be accessed with 64-bit load/store operations.
- **Core EJTAG TAP registers**. These CSRs are duplicated on each core, and can only be accessed by an external EJTAG device via the CN50XX’s EJTAG interface. These are covered in Section 4.13.
- **NCB registers**. These registers are accessed directly off the I/O bus. These registers can be accessed by any core and by remote PCIe devices/hosts through a windowing mechanism via BAR0 read/write operations.
  - Each NCB register has a CN50XX-internal address.
  - All CN50XX NCB registers are 64 or 32 bits, and must be accessed with 64-bit or 32-bit load/store operations.

- **RSL registers**. These registers are accessed indirectly off the I/O bus. They are similar to NCB registers, except that access to them is much slower as it is indirect access. The hardware can only service an RSL register access about once every 30–40 cycles. These registers can be accessed by any core and by remote PCIe devices/hosts.
  - Each RSL register has a CN50XX-internal address.
  - All CN50XX RSL registers are 64-bits, and must be written with 64-bit stores, but can be read with any size load.
- **PCICONFIG registers**. These registers can be accessed directly from the PCI bus and from the I/O bus.
  - Each PCICONFIG register has both a PCI bus address and an OCTEON-internal address.
    - Cores can access the PCICONFIG registers via the CN50XX-internal address.
    - Remote PCI devices/hosts can access PCICONFIG registers with a PCI configuration space read or write operation that targets CN50XX.
  - All PCI config registers are 32-bits. 32-bit load/store operations must be used by cores to access the PCICONFIG registers.
- **PCI registers**. These registers can only be accessed directly from the PCI bus, and cannot be accessed by the local cores. Remote PCI devices/hosts can access the PCI registers with a PCI memory space read or write operation to CN50XX's BAR0.
- **PCI_NCB registers**. These registers can be accessed directly from the PCI bus and from the I/O bus.
  - Each PCI_NCB register has both a PCI bus address and a CN50XX-internal address.
    - Cores can access the PCI_NCB registers via the internal address.
    - Remote PCI devices/hosts can access PCI_NCB registers with a PCI memory space read or write operation to CN50XX's BAR0.
  - Cores can access PCI_NCB registers via 32-bit load/store operations.
- **JTAG TAP registers**. These registers can only be accessed by an external device via CN50XX's JTAG interface.
- **TWSI core registers**. These registers control the TWSI core and can only be accessed indirectly via MIO_TWS_SW_TWSI (MIO_TWS_SW_TWSI is an ordinary RSL CSR). TWSI core registers are covered in Chapter 19.

The detailed descriptions of these CSRs are distributed throughout the remaining chapters of this document. Table 1–1 lists the CSRs types that are present in the different chapters.

Table 1–1 CN50XX CSR Types

Chapter	Units	CSR Type
2	Level-Two Cache Controller (L2C), DRAM Controller	RSL
3	I/O Bridge	RSL
5	Packet Order / Work Unit	NCB
6	Free Pool Unit	RSL
7	Packet Input Processing, Input Packet Data	RSL, NCB
8	Packet Output Processing	RSL
9	PCI Bus, NPI	PCI_NCB, PCICONFIG, NCB, PCI, NCB
9	NPI	NCB
10	Timer	RSL
11	Central Interrupt Unit	NCB
12	Boot Bus	RSL
13	Reduced Gigabit Media Independent Interface, ASX	RSL
14	TDM/PCM	NCB
15	GPIO	NCB
16	UART	NCB
17	TWSI Interface	NCB
18	System Management Interface	RSL
19	Random Number Generator, Random Number Memory	RSL
20	SPI/MPI	NCB
21	USB	NCB

NOTE: The listed CSR addresses are all little-endian format. Because most CN50XX CSRs are 64-bit registers, and all 64-bit addresses are endian-neutral, most addresses in this book are endian-neutral.

1.4.1 CSR Field Types

CN50XX has a variety of CSR Field types as described below.

- **RO** – Fields that can be read from only.
- **R/W** – Fields that can be read from or written to.
- **RAZ** – Reserved fields.
- **R/W1C** – Fields that can be read from or written to with a 1 to clear.
- **RC** – Fields that are cleared when read.
- **RC/W** – Fields that are cleared when read or written to.

Chapter 2

Coherent Memory Bus, Level-2 Cache Controller, DRAM Controller

This chapter describes the following topics:

- Coherent Memory Bus (CMB)
- Level-2 Cache Controller (L2C)
- DRAM Controller (LMC)
- L2C Registers
- LMC Registers

Section 2.1 discusses the coherent memory bus (CMB) and contains the following subjects:

- CMB Overview
- CMB Buses
- CMB Description
- CMB Memory Coherence Support
- CMB Transactions

Section 2.2 discusses the Level-2 cache controller (L2C) and contains the following subjects:

- L2 Cache and Data Store
- L2C Memory Coherence
- L2 Cache Indexing (Set Selection)
- L2 Cache Replacement and Way-Partitioning
- L2 Cache-Block Locking
- Cache-Block Flush and Unlocking
- Memory Input Queue Arbitration
- COMMIT and FILL Bus Arbitration
- L2C ECC Codes

Section 2.3 discusses the DRAM controller and contains the following subjects:

- Main Memory DRAM Addressing
- DRAM Part Addressing
- DRAM Transaction Examples
- DRAM Programming
- DRAM Refreshes
- DRAM Scheduler Performance
- DRAM Chip Selects and ODT
- DRAM Controller Initialization
- DRAM ECC Codes

2.1 Coherent Memory Bus (CMB)

2.1.1 CMB Overview

The coherent memory bus (CMB), shown in Figure 2–1, is the communication channel for all memory and I/O transactions between the two cores, the I/O bridge (IOB), and the level-2 cache controller (L2C). It runs at the core-clock frequency.

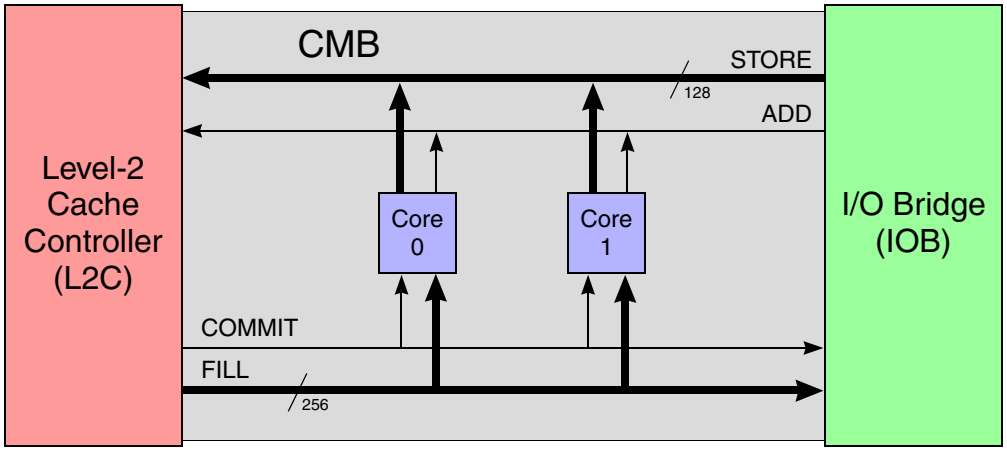


Figure 2–1 Coherent Memory Bus Diagram

2.1.2 CMB Buses

The CMB is comprised of four buses:

- The Address/Command bus (ADD), which typically transfers an address, and always transfers control information to initiate CMB transactions.
- The Store Data bus (STORE), which always transfers the store data associated with the transaction.
- The Commit/Response Control bus (COMMIT), which always transfers control information that initiates transaction responses from the L2C.
- The Fill Data bus (FILL), which transfers fill data (cache blocks) from the L2C and reflection data for core-to-IOB transfers.

2.1.3 CMB Description

The CMB is a split-transaction highly pipelined bus. All the ADD, STORE, COMMIT, and FILL buses are decoupled by large queues. This decoupling allows for variable timing between the different ADD/STORE/COMMIT/FILL operations required to complete different CMB transactions. Consequently, CN50XX schedules the individual buses at maximum efficiency for highest bandwidth CMB operation. (Some COMMIT bus operations are coupled, at fixed timing, to later FILL bus cycles.) At 550MHz operation, the CMB peak performance is 211 Gbps.

NOTE: CMB transactions transfer no more than 128 bytes of data at a time. This is the cache-block size.

2.1.4 CMB Memory Coherence Support

The CMB contains write-invalidate coherence support. The core data caches are write-through. L2C maintains copies of the core data-cache tags and initiates CMB invalidations to core data caches when other CMB sources update cached blocks. The cores contain large write buffers to minimize the number of CMB write operations, and support a weakly consistent memory model to maximize performance. The CMB commit information allows weakly consistent bus sources (like the cores) to order operations, as necessary.

Memory requests to coherent memory space are directed to the L2C. Memory requests can be initiated by either a core or the IOB. The IOB initiates memory requests on behalf of I/O agents located on the other side of the IOB. Memory responses are returned by the L2C to the source of the original request, either the core or IOB.

Other requests, such as core-to-IOB transfers, physically go through the L2C, and are reflected onto the FILL bus before being received by the destination core/IOB.

Coherent Memory Bus Transaction Example

FILL Transaction

Figure 2–2 shows the steps of a complete fill transaction, a cache-block load, on the CMB. A fill is any CMB transaction that puts a cache line into either the L1 instruction or data caches. (In Table 2–1, these are LDD, PSL1, LDI, and LDT transactions.) As with all CMB transactions, the transaction is initiated by either a core or the IOB. The first step (not shown in the figure) for this transaction, as with all CMB transactions, is that the core/IOB arbitrates for a cycle on the ADD bus. This is necessary to guarantee that only one among the cores and the IOB drives the ADD bus on a given cycle. The ADD bus arbitration is fair (round-robin) between all the drivers of the ADD bus. Once the core or IOB wins arbitration, it puts control information indicating that the transaction is a fill, together with the address of the cache block, onto the ADD bus in a single cycle.

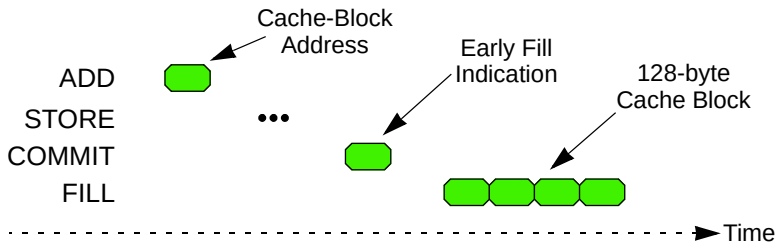


Figure 2–2 Fill

The L2C receives the ADD bus information, possibly queues it, and eventually services the transaction. In this case, servicing consists of consulting the L2 cache and/or DRAM to obtain a copy of the cache block. As Figure 2–2 indicates, once the L2C has a copy of the block, it sends an early fill indication on the COMMIT bus. Starting three cycles later, the L2C then transfers the 128 byte cache block on the FILL bus in four consecutive cycles to complete the CMB transaction. The number of cycles required to complete the fill transaction varies depending on a number of factors, including whether the L2C found the block in the L2 cache or in memory.

Coherent Memory Bus Transaction Example

Store Without Invalidate

Figure 2–3 shows a store, without invalidate, on the CMB. The ADD bus cycle indicates a store transaction, and contains the address of the store as well as the number of 128-bit octaword transfers required on the STORE bus. The STORE

bus cycles are scheduled later, once the STORE bus is available and buffer space is available to receive the data, wherein the core or IOB drive the store data of the transaction onto the STORE bus. This example is a case where five STORE bus cycles are required for the complete store. The number of STORE bus cycles may range from as small as one, up to a possible eight to transfer an entire 128-byte cache block.

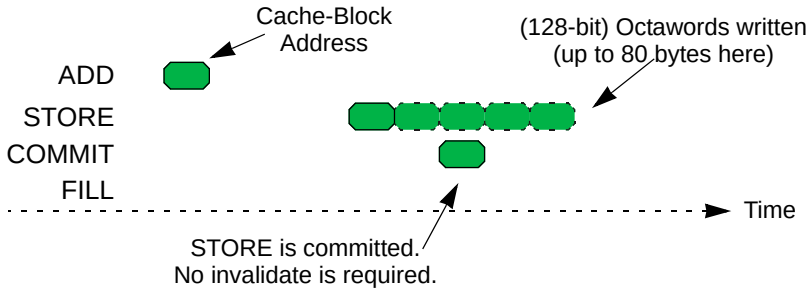


Figure 2–3 Store Without Invalidate

The L2C buffers the ADD and STORE bus information and services the write operation. Figure 2–3 shows a more common case where a copy of the cache block is not held in another core; no core Dcache invalidation is required. L2C places a commit operation on the COMMIT bus. The commit indicates that the store is visible to all bus users at this time, and may be sent long before the actual store operation is completely retired. For example, L2C can send the commit operation for a store even though the store is not yet deposited in the cache/DRAM, provided that any subsequent CMB transaction will see the updated store value.

Coherent Memory Bus Transaction Example

Store With Invalidate

Figure 2–4 shows a store with invalidate on the CMB. This is similar to Figure 2–3. One difference is that the transaction requires only three cycles on the STORE bus rather than five. The other difference is that a commit/invalidation operation appears on the COMMIT bus, followed three cycles later by an invalidation cycle on the FILL bus. The invalidation cycle causes a data-cache block invalidate in one or both cores, and is assumed to be required in this example, because an out-of-date copy of the cache block resides in at least one data cache.

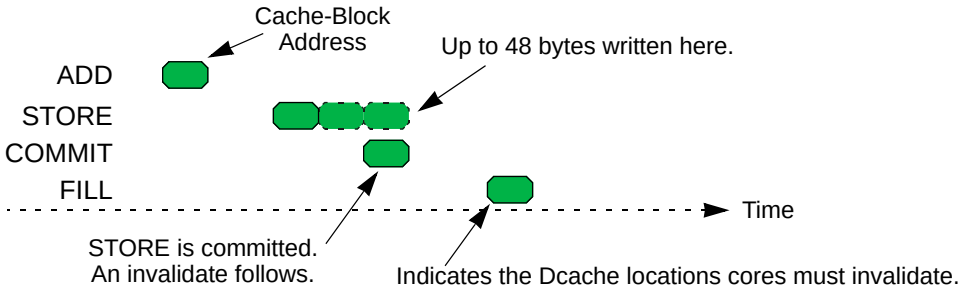


Figure 2–4 Store with Invalidate

Coherent Memory Bus Transaction Example

Store/IOBDMA Reflection

Figure 2–5 shows a store/IOBDMA reflection. Effectively, a reflection transaction is simply a way to transfer commands/results between the IOB and the cores. This can be required, for example, when a core does a store destined to either the IOB or the opposite side of the IOB. In these reflection transactions, the L2C buffers the ADD and STORE bus values, sends an early fill code indicating a reflection

transaction response, and copies the ADD/STORE values onto the FILL bus three cycles later. The destination, (the IOB in the previous example) latches the ADD/STORE values from the FILL bus and processes the operation.

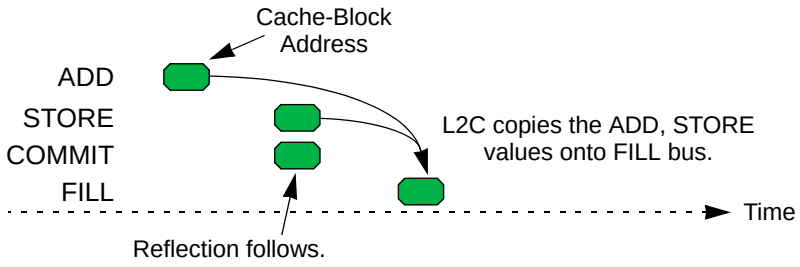


Figure 2–5 Store ADD from Core or IOB

Coherent Memory Bus Transaction Example

Load Reflection From Core

Figure 2–6 shows a load reflection. This is required when a core does a load destined to either the IOB or the opposite side of the IOB. The transaction is the same as the Figure 2–4 store/IOBDMA case, except that there is no STORE bus cycles.

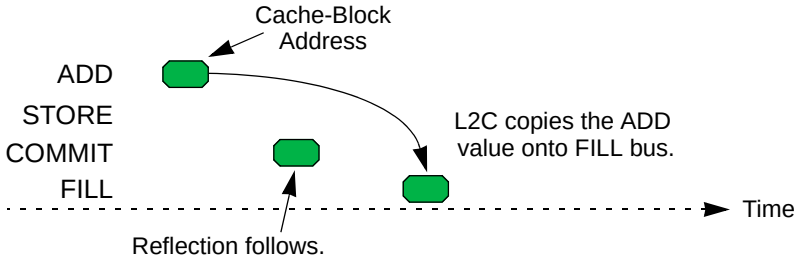


Figure 2–6 Load Reflection from Core

With any load/IOBDMA operation initiated by a core, there will later be an accompanying response transaction, initiated by the IOB, to return the result of the load/IOBDMA operation to the requesting core. This response transaction is a store reflection, similar to that shown in Figure 2–5.

2.1.5 CMB Transactions

Table 2–1 shows the CMB's transactions.

Table 2–1 Coherent Memory Bus Transactions

Transaction	Description
Data Cache Fill (LDD)	The cache block is filled from L2/DRAM. The block will be filled into the Dcache of the requesting core.A subsequent store from another core or IOB will cause an invalidate. The block is put into the L2 cache. These transactions can only be issued by cores.
Data Cache Fill-Through (PSL1)	The cache block is filled from L2/DRAM. The block will be filled into the Dcache of the requesting core. A subsequent store from another core or IOB will cause an invalidate. The block will not be put into the L2 cache. These transactions can only be issued by cores.
Instruction Cache / IO Fill (LDI)	The cache block is filled from L2/DRAM. The block is put into the L2 cache. These transactions can be issued either by cores or by the IOB. A subsequent store from another core or IOB will NOT cause an invalidate.
Instruction Cache / IO Fill-Through (LDT)	The cache block is filled from L2/DRAM. The block will not be put into the L2 cache. These transactions can be issued either by cores or by the IOB.
Prefetch Into L2 (PL2)	The cache block will be put into the L2 cache. This is an ADD-only transaction on the CMB. These transactions are only issued by cores.
Store-Partial (STP)	Some of the bytes in the cache block will be stored. The value of the bytes in the cache block that are not transferred on the STORE bus or are masked off are not modified. All data-cache copies of the block will be invalidated, except for the data cache of an initiating core. The cache block will be put into the L2 cache. These transactions are issued either by cores or by the IOB.
Store-Full (STF)	Store to all bytes in the cache block. The value of the bytes in the cache block that are not transferred on the STORE bus or are masked off will be written to 0. All data cache copies of the block will be invalidated, except for the data cache of an initiating core. The cache block will be put into the L2 cache. These transactions are issued either by cores or by the IOB.
Store-Through (STT)	Store to all bytes in the cache block. The value of the bytes in the cache block that are not transferred on the STORE bus or are masked off will be written to 0. All data cache copies of the block will be invalidated, except for the data cache of an initiating core. The cache block will not be put into the L2 cache. These transactions are only issued by the IOB.
Store-Conditional (STC)	Store to either 32 or 64-bits if the block is currently held in the data cache of the requesting core. If the block is in the data cache of the requesting core, the store happens and a commit indication is returned. All data-cache copies of the block will be invalidated in the other cores. If the block is not in the data-cache of the requesting core, a failure indication is returned, no invalidate occurs, neither does the store. The cache block will be put into the L2 cache. These transactions are only issued by cores.

Table 2–1 Coherent Memory Bus Transactions (Continued)

Transaction	Description
Store-Atomic Add (SAA)	Store atomic add to either 32 or 64 bits. Acts like an unmerged STP transaction, except that the valid bytes are added to the memory location, not stored to it. All data-cache copies of the block are invalidated in all cores, including the requesting core. The cache block is put into the L2 cache. These transactions are issued only by cores.
Don't-Write-Back (DWB)	Clear the dirty bit in the L2 tags if the cache block is present in the L2 cache. This is an ADD-only transaction on the CMB.
Store Reflection (IOBST, IOBDMA, IOBRSP)	Reflect an ADD cycle and corresponding STORE cycle(s) onto the FILL bus. This is used for core ⇔ IOB communication. A store reflection transaction is generated by the cores to transfer a store/IOBDMA to IOB. An IOBDMA must have exactly 64 bits of (aligned) STORE data. A store may have 64, 32, 16, or 8 bits of aligned STORE data. These transactions are destined to IOB. A store reflection transaction is generated by the IOB to respond to a prior load/IOBDMA request from a core. A load response always contains exactly 64 bits of (aligned) STORE data. An individual IOBDMA response contains between one and sixteen 64 bit (aligned) words of data. These transactions are destined to a core.
Load Reflection (IOBLD8, IOBLD16, IOBLD32, IOBLD64)	An ADD cycle will be reflected onto the FILL bus. This is used for communication from the cores to the IOB. A load reflection transaction is generated by the core to transfer a load to IOB.

2.2 Level-2 Cache Controller (L2C)

This section discusses the L2C and contains the following subjects:

- L2 Cache and Data Store
- L2C Memory Coherence
- L2 Cache Indexing (Set Selection)
- L2 Cache Replacement and Way-Partitioning
- L2 Cache-Block Locking
- Cache-Block Flush and Unlocking
- Memory Input Queue Arbitration
- COMMIT and FILL Bus Arbitration

2.2.1 L2 Cache and Data Store

The L2C block diagram is shown in Figure 2–7. The L2C contains CN50XX's shared on-chip cache. The L2 cache has the following specifications:

- 128KB
- eight-way set-associative with a 128-byte cache block
- write-back
- both the on-chip data and tags are protected by SECDED ECC.

The L2 cache is shared by both the cores and the I/O components on CN50XX, though it can be bypassed using particular CMB transactions and can also be partitioned.

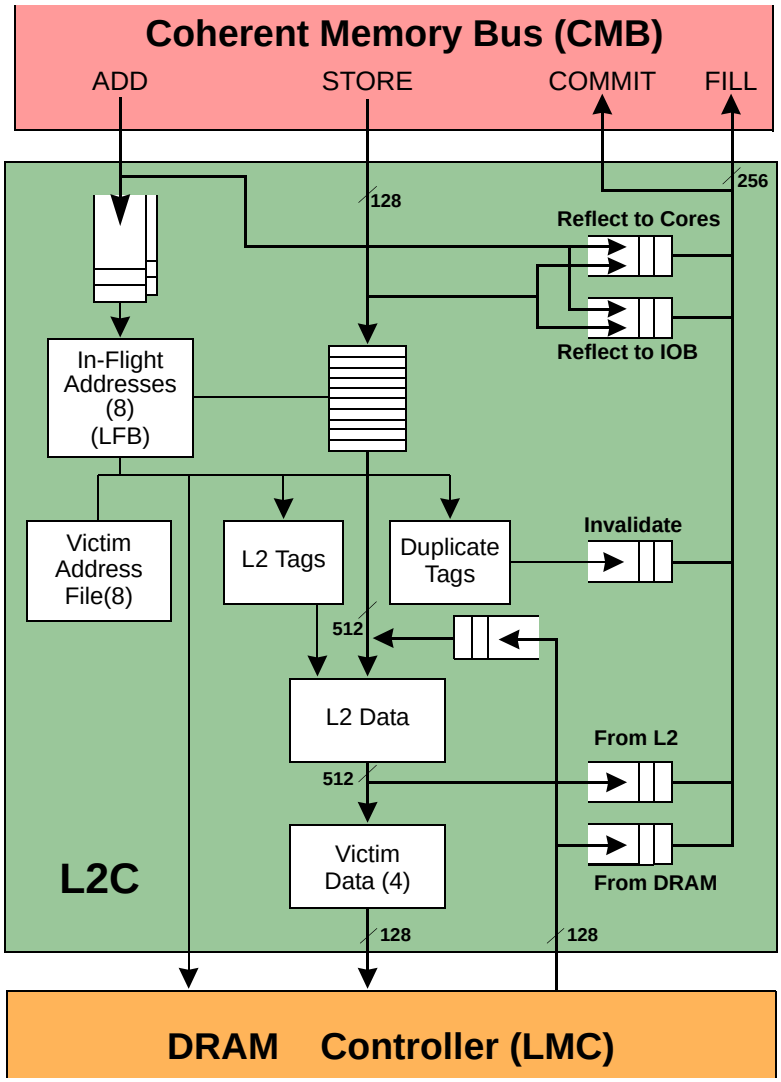


Figure 2–7 L2C Block Diagram

2.2.2 L2C Memory Coherence

L2C maintains CN50XX's memory-reference coherence.

- It returns the latest copy of the block for every fill request, whether the block is in the L2 cache, in DRAM, or in-flight somewhere.
- It also contains a duplicate copy of the tags for each core's data cache. It compares the addresses of cache-block store requests against the data cache tags, and invalidates (both copies) a data-cache tag for a core whenever the store is from another core or from an I/O component, via the IOB.

The L2C services CMB reflections, that is, non-memory transactions that are necessary to transfer commands and/or data between the cores and the IOB.

Figure 2–7 is a block diagram of the L2C. The figure shows the CMB interface on the top and the DRAM controller interface on the bottom. The CMB interface is 384 bits wide, the DRAM interface is 256 bits wide, and the internal L2 cache data interfaces are 512 bits wide.

L2C has two memory-input queues that receive ADD bus memory transactions: one for transactions initiated by the cores, and one for transactions initiated by the IOB. Having the separate IOB queue improves L2C response to IOB-initiated transactions. Without two queues, IOB memory references would have priority equal to the two cores, so it could be starved when both cores are saturating L2/DRAM.

L2C contains two reflection queues that hold the ADD/STORE bus information to be reflected. Two different reflection queues are necessary for deadlock avoidance:

- one is for reflections destined to the cores
- the other is for reflections destined for the IOB.

L2C holds and processes up to 8 simultaneous memory transactions in its in-flight address buffer (LFB). It can also manage up to 8 in-flight L2 cache victims, and up to four of these victims may reside in the victim-data file.

For fills, L2C returns data from either the L2 cache or memory. L2C first deposits memory STORE bus data into a file associated with the in-flight addresses. It then updates the cache (STP, STF, STC, or SAA in Table 2–1), or writes the store data straight to memory (STT in Table 2–1). STF and STT are full cache-block writes. The partial cache-block writes (STP, STC and SAA) require a DRAM fill (to first read on the old data in the block) if the store misses in the cache.

All data movement transactions between the L2C and LMC are 128-byte, full-cache blocks. The L2C buffers LMC fills in one or both of two queues:

- one for data destined to be written into the L2 cache
- the other for data destined for the FILL bus.

The L2C buffers LMC stores in the victim address / data files until the LMC accepts them.

L2C buffers all the COMMIT/FILL bus commands needed from each possible source: the two reflection queues, fills from L2/DRAM, and invalidates. These two queues allows the arbiter to saturate the FILL bus when a heavy load requires it.

2.2.3 L2 Cache Indexing (Set Selection)

There are 128 sets in the L2 cache. Each set contains eight 128-byte cache blocks, one per available way. Every L2/DRAM reference must select a set.

L2C implements two set-indexing algorithms, an unaliased algorithm (L2C_CFG[IDXALIAS] = 0), and an aliased one (L2C_CFG[IDXALIAS] = 1). The aliased algorithm is recommended for most applications.

- The unaliased algorithm is the typical set-selection algorithm:

  index<6:0> = address<13:7>

  where address is the physical byte address of the L2/DRAM location.

- The aliased set-selection algorithm additionally bit-wise exclusive-ORs upper address bits into the index:

  index<6:0> = address<13:7> ⊕ address<20:14>

Both the unaliased and aliased algorithms spread contiguous cache blocks across sequential sets. The aliased algorithm also spreads cache blocks with different upper address bits across different sets. This is advantageous when memory addressing is not strictly contiguous or not random.

For example, if packet buffers are 2KB and always naturally aligned, but only the first 256 bytes of every packet buffer is typically used, the unaliased algorithm can only ever use two sets out of every sixteen, but the aliased algorithm evenly distributes the blocks across all sets, provided the naturally aligned 2KB buffers cover many 16KB aligned regions. This can have a very large performance effect.

NOTE: L2C_CFG[IDXALIAS] must change value only when the L2 cache is known to be completely empty, such as at boot time.

2.2.4 L2 Cache Replacement and Way-Partitioning

L2C implements its replacement policy using one USED bit per L2 cache block. There are eight USED bits per each of the 128 sets, one per way. (All eight ways are normally “available”, but way-partitioning, described below, can remove ways from consideration.)

When L2C references a block, it sets the USED bit for the block to 1. When the USED bits for all available ways in a set are all 1s, L2C clears to 0 the USED bits for all the available ways in the set, except for the last block referenced. When L2C replaces a cache block, it picks the first available way that has a USED bit of 0 and does not hold a locked block. If the USED bits of all available ways in a set are all 1s on a replacement operation, L2C replaces the first available way that does not hold a locked block. (Section 2.2.5 covers locking.)

Way-partitioning can prevent specific cores or I/O devices from polluting the L2 cache. Eight-bit UMSK values configure the way-partitioning. There are a total of three 8-bit UMSK CSR fields for the different sources of L2/DRAM transactions on the CMB bus:

- two for the cores: one for each of the cores (L2C_SPAR0[UMSK0/1])
- one for all the IOB-initiated transactions (L2C_SPAR4[UMSKIOB])

When [UMSK0/1<*i*>] is set, the selected source cannot place a block into way *i*. Way *i* is not available to the source for replacements.

NOTE: Way partitioning does not affect cache coherence and does not directly affect cache-hit processing in any way. Way-partitioning only affects cache replacement.

2.2.5 L2 Cache-Block Locking

L2C can lock individual cache blocks into the L2 cache. A locked block is not replaced (until it is later flushed from the cache via the procedure described in Section 2.2.6, or until the chip is reset), so fast access to the block is guaranteed when it is locked.

When L2C_LCKBASE[LCK_ENA] is set, L2C locks cache blocks referenced by an LDD or LDI transaction on the CMB bus (i.e. and instruction or data cache fill, refer to Section 2.1.5) as long as both the following occur:

- The selected core (L2C_DBG[PPNUM]) sourced the LDD or LDI
- The index-aliased LDD or LDI physical address falls in the range selected by L2C_LCKBASE[LCK_BASE] and L2C_LCKOFF[LCK_OFFSET].

If the locking LDD/LDI does not find the block in the cache, the locked block resides in the way selected by the normal replacement algorithm for the source core after the transaction. If the locking LDD/LDI finds the block already in the cache, the locked block resides in the way that previously held the block after the transaction. It is difficult to predict the exact way that a locked block was placed in, though the selection can be influenced with way-partitioning (refer to Section 2.2.4) while locking.

L2C_LCKOFF[LCK_OFFSET] should always be updated appropriately before L2C_LCKBASE[LCK_ENA] is set. The CSR write that sets L2C_LCKBASE[LCK_ENA] also writes L2C_LCKBASE[LCK_BASE]. Note also that L2C_LCKOFF[LCK_OFFSET] can cover at most 128KB. Larger address ranges must be split into multiple different ranges.

To lock a different DRAM range from the current one:

- Clear L2C_LCKBASE[LCK_ENA] to 0
- Update L2C_LCKOFF[LCK_OFFSET] for the new range
- Set L2C_LCKBASE[LCK_BASE] for the new range and set L2C_LCKBASE[LCK_ENA] = 1 (Both can be done with a single CSR write.)

Note that software should always follow a write operation to L2C_LCKBASE that changes the L2C_LCKBASE[LCK_ENA] value with a read operation to L2C_LCKBASE (and wait for the result). Like other CN50XX CSRs, L2C_LCKBASE CSR write operations are posted (refer to Section 4.9), so the read operation is necessary to ensure that L2C has the updated L2C_LCKBASE value before any subsequent operations.

Refer to Section 2.2.4 regarding ways and way partitioning. For every source, there should be at least one block in every set that is both available and not locked. When this condition is violated, some combination of the L2T_ERR[LCKERR,LCKERR2] error bits will set, and some blocks that were desired to be locked are not locked.

Note that L2C_LCKBASE[LCK_BASE] is an index-aliased physical-cache-block address. When L2C_CFG[IDXALIAS] is set to 1, the hardware first index-aliases the LDD/LDI physical address before it compares it to L2C_LCKBASE[LCK_BASE] and L2C_LCKOFF[LCK_OFFSET]. Index-aliasing simply replaces address<13:7> with index<6:0>, where Section 2.2.3 defines index. An LDD/LDI is a locking candidate when its index-aliased physical byte address falls within the range:

from: L2C_LCKBASE[LCK_BASE]<<7
to: (L2C_LCKBASE[LCK_BASE] + L2C_LCKOFF[LCK_OFFSET])<<7 + 127

Note also that L2C discards bit<34> of the physical-byte addresses, so the address ranges mentioned above are based solely on <33:0> of a L2/DRAM byte address.

2.2.6 Cache-Block Flush and Unlocking

During normal operation, L2C only flushes cache blocks from the L2 cache when it is necessary to create space for new blocks. L2C contains a special cache-flush mode that can force cache blocks to be flushed. This is the only way to flush locked blocks from the cache, so this mode is also a cache block unlocking mode (Section 2.2.5 discusses block locking). It is also one way to flush a block with an ECC error out of the cache, and it is the only way if the block is locked into the cache.)

L2C is in the cache-flush mode whenever L2C_DBG[FINV] is set to 1. In this mode, L2C treats cached full cache block write transactions (STFs, refer to Section 2.1.5) initiating from the selected core (L2C_DBG[PPNUM]) specially. These flushing STFs evict blocks from the cache and do not perform any actual write nor validate a new block. L2C flushes the block whether it is locked or not. Note that L2C discards any store data associated with the flushing STF when it is in flush mode. If the flushed block is dirty, L2C writes it to DRAM. In any case, before the transaction completes, L2C invalidates the selected cache location.

L2C flushes the block in the cache location in the set selected by the index of the flushing STF (see Section 2.2.3 for the index calculation), and in the way selected by L2C_DBG[SET]. Note that the index of the flushed block must match the index of the flushing STF, but the address of the flushed block is otherwise unrelated to the address of the flushing STF.

NOTE: Software should always execute a SYNC instruction immediately before writing the L2C_DBG register to change the L2C_DBG[FINV] value, or to change the L2C_DBG[SET,PPNUM] values when L2C_DBG[FINV] is set. Software should also follow the L2C_DBG write operation with a L2C_DBG read operation (and wait for the result).

SYNC flushes prior STF transactions from the write buffer before the L2C_DBG update (refer to Section 4.10 for a description of the write buffer, and to Appendix A for a description of the SYNC instruction). Like other CN50XX CSRs, L2C_DBG CSR write operations are posted (refer to Section 4.9), so the L2C_DBG read operation is necessary to ensure that L2C has the updated L2C_DBG values before any subsequent operations.

Cavium Networks recommends that the flushing STF transactions are generated on the cores via a sequence of (at least) two instructions:

- PREF 30 (i.e. prepare for store) to generate the STF, with the appropriate physical address to generate the desired L2 cache index. (Beware that PREF instructions become NOPs if they would otherwise cause TLB misses.)
- SYNCW to force the STF transaction out of the core-write buffer.

NOTE: Refer to the PREF and SYNCW instruction descriptions in Appendix A.

Note that the selected core may generate STF transactions any time the software writes every byte in a cache line. Care is necessary to ensure that only the desired STF transactions are created whenever L2C_DBG[FINV] is set, since L2C discards any store data included with a flushing STF transaction. Unwanted STF transactions while L2C is in flush mode cause corruption.

If a particular cache block should be flushed/unlocked from the L2 cache, but the way that the cache block resides in the cache is not known (the index of the block

is assumed known since the physical address of the block is assumed known), all possible ways can be flushed by changing L2C_DBG[SET]. (Note the discussion above regarding restrictions on changing L2C_DBG[SET] while L2C_DBG[FINV] is set.) Similarly, the entire L2 cache can be flushed/unlocked by flushing all possible ways and indexes with a sequence of many flushing STF transactions.

2.2.7 Memory Input Queue Arbitration

As shown in Figure 2–7, L2C has two queues for ADD-bus transactions. This configuration automatically improves response time for IOB-initiated traffic when the cores heavily use the CMB. This is important for real-time packet transfers. The two queues alone eliminate many potential performance problems, but the exact algorithm that selects between queues can further increase/decrease the priority of IOB-generated traffic.

L2C processes transactions from the two queues in one of two programmable arbitration algorithms:

- fixed priority (L2C_CFG[LRF_ARB_MODE] = 0)
- round-robin (L2C_CFG[LRF_ARB_MODE] = 1).

With fixed priority, IOB-initiated transactions are always higher priority. For systems with very high I/O bandwidth requirements, fixed priority may be preferred.

2.2.8 COMMIT and FILL Bus Arbitration

The COMMIT/FILL bus arbiter can operate in one of the following modes:

- round-robin mode (L2C_CFG[RSP_ARB_MODE] = 1)
- static-priority mode (L2C_CFG[RSP_ARB_MODE] = 0).

Round-robin is more fair, but the fixed priority mode prioritizes reflections above other FILL bus usages, and this may be advantageous to minimize core-initiated reflection latencies.

2.2.9 L2C ECC Codes

Table 2–2 shows the L2T 23-bit ECC code for cache-data references, and Table 2–3 shows the L2D 128-bit ECC code for tag references.

Table 2–2 L2T 23-Bit ECC Code

CCCCCCC 5:0	22:20	19:16	15:12	11:8	7:4	3:0	
000001	110	0001	0010	1100	1011	0111	0x612CB7
000010	111	0010	0101	0101	0101	1011	0x72555B
000100	111	0100	1001	1010	0110	1101	0x749A6D
001000	111	1000	1110	0011	1000	1110	0x78E38E
010000	011	1111	0000	0011	1111	0000	0x3F03F0
100000	101	1111	1111	1100	0000	0000	0x5FFC00
000000	000	0000	0000	0000	0000	0000	
xxxxxx	xxx	xxxx	xxxx	xxxx	xxxx	xxxx	
210000	213	3333	2222	2211	1111	0000	← Syndromes
008421	FFE	8421	CA96	53CA	9653	EDB7	

Table 2–3 L2D 128-Bit ECC Code

CCCCCCC 9:0	59:56	55:52	51:48	47:44	43:40	39:36	35:32	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0	
0000000001	1011	0000	0100	0010	0010	0101	1000	0100	0100	1011	0001	0010	1100	1011	0111	0xB04225844B12CB7
0000000010	0101	0000	1000	0100	0100	1010	1000	1000	1001	0101	0010	0101	0101	0101	1011	0x50844A88952555B
0000000100	0110	0001	0000	1000	1001	0011	0001	0001	0010	0110	0100	1001	1010	0110	1101	0x610893112649A6D
0000001000	1000	0010	0001	0001	0001	1100	0010	0010	0011	1000	1000	1110	0011	1000	1110	0x82111C22388E38E
0000010000	0000	0100	0010	0001	1110	0000	0100	0011	1100	0000	1111	0000	0011	1111	0000	0x0421E043C0F03F0
0000100000	0000	1000	0011	1110	0000	0000	0111	1100	0000	0000	1111	1111	1100	0000	0000	0x083E007C00FFC00
0001000000	0000	1111	1100	0000	0000	0000	0111	1111	1111	1111	0000	0000	0000	0000	0000	0x0FC0007FFF00000
0010000000	0000	1111	1111	1111	1111	1111	1000	0000	0000	0000	0000	0000	0000	0000	0000	0x0FFFFF800000000
0100000000	1111	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0xF00000000000000
1000000000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0x000000000000000
0000000000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
xxxxxxxxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	
2100000000	1111	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	← Syndromes
0084210000	0000	EDCC	CCBA	AAA9	9998	8888	8766	6655	5544	4444	3333	2222	2211	1111	0000	
0000008421	9653	0084	2108	4218	421C	A965	3084	2184	21CA	9653	8421	CA93	53CA	9653	EDB7	

127:124	123:120	119:116	115:112	111:108	107:104	103:100	99:96	95:92	91:88	87:84	83:80	79:76	75:72	71:68	67:64	63:60	
0000	0001	0000	0010	0000	1000	0111	0100	1100	1011	0000	0000	0010	0000	1000	0100	0100	0x01020874CB0020844
0000	0010	0000	0100	0001	0000	1001	0101	0101	0101	0001	0000	0100	0001	0000	1000	1001	0x02041095551041089
0000	0100	0000	1000	0010	0001	0010	0110	0110	0110	0010	0000	1000	0010	0001	0001	0010	0x04082126662082112
0000	1000	0001	0000	0100	0010	0011	1000	0111	1000	0100	0001	0000	0100	0010	0010	0011	0x08104238784104223
0001	0000	0010	0000	1000	0100	0011	1111	1000	0000	1000	0010	0000	1000	0100	0011	1100	0x1020843F80820843C
0010	0000	0100	0001	0000	0111	1100	0000	0000	0000	1111	0100	0001	0000	0111	1100	0000	0x204107C000F4107C0
0100	0000	1000	0001	1111	1000	0000	0000	0000	0000	1111	1000	0001	1111	1000	0000	0000	0x4081F80000F81F800
1000	0000	1111	1110	0000	0000	0000	0000	0000	0000	1111	1111	1110	0000	0000	0000	0000	0x80FE000000FFE0000
1111	1111	0000	0000	0000	0000	0000	0000	0000	0000	1111	1111	1111	1111	1111	1111	1111	0xFF00000000FFFFFFF
1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	0000	0000	0000	0000	0000	0000	0000	0xFFFFFFFFFF0000000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	
3333	3333	2222	2222	2222	2222	2222	2222	2222	2222	1111	1111	1111	1111	1111	1111	1111	← Syndromes
8421	0000	CA98	8886	5444	4322	2211	1111	1000	0000	FEEE	CB98	8886	5444	4322	2211	1100	
0000	8421	0008	4210	0842	1084	21DB	8742	1FCA	9653	0842	0008	4210	0842	1084	2184	21CA	

2.3 DRAM Controller (LMC)

The LMC is fully programmable to support the JEDEC DDR2 SDRAM specification (refer to the JEDEC standard: *DDR2 SDRAM Specification JESD79-2B (Revision of JESD79-2) January 2005*). The LMC has a 32/16-bit-wide interface (36/18 bits wide with optional SECDED ECC), and can architecturally support up to 16GB of SDRAM with data rates of up to 667 MHz.

The 32/36-bit interface can support up to two DIMMs. There is a single address bus shared by all the DIMMs/parts.

The SDRAM controller has the following features:

- Support for ×8 and ×16 components from 256Mb to 2Gb (×4 support only for ECC)
- Memory can be assembled as registered or unbuffered DIMMs or a similar combination
- Support for up to four independent ranks, addressable with four chip-select signals
- Low-address cache-block bank selection with optional hashing
- An aggressive reordering scheduler
  - Entire 128-byte cache-block read/write operations (data masking not supported)
  - Bank autoprecharge following each read/write operation
  - four or eight banks supported
- 1T and 2T address/command support (Micron Technical Note TN-47-01)
- Burst-length 4 and 8 support
- Posted-CAS support
- Support for nonhomogeneous ranks/chip selects
- Separate DDR2 ODT enables for read operations and write operations
- Write-data mask not required; optional RDQS
- Support for data rates up to 667 MHz
- Programmable settings to meet various timing specifications
- Programmable settings to compensate for board delays with quarter-cycle granularity
- Programmable settings per-byte with quarter-cycle granularity to compensate for board delays
- No support for DDR OCD calibration
- No support for DDR2 power-down

The CN50XX LMC supports a variety of combinations of memory devices. The combinations are shown in Table 2–4.

Table 2–4 CN50XX Supported Memory Devices

Device Size	Device Configuration × DQ Pins	Row/ Column/ Banks Bits	Physical Banks
256 Mb	32Mb × 8	13/10/2	4
	16Mb × 16	13/9/2	4
512 Mb	64Mb × 8	14/10/2	4
	32Mb × 16	13/10/2	4
1Gb	128Mb × 8	14/10/3	8
	64Mb × 16	13/10/3	8

Table 2–4 CN50XX Supported Memory Devices (Continued)

Device Size	Device Configuration × DQ Pins	Row/ Column/ Banks Bits	Physical Banks
2Gb	256Mb × 8	15/10/3	8
	128Mb × 16	14/10/3	8

Figure 2–8 shows a high-level block diagram of the CN50XX LMC. Read and write requests are received from L2C along with address and data (in write requests) for 128-byte cache-block accesses to the memory. Asynchronous FIFOs (MRF for memory-read requests and MWF for memory-write requests) queue up requests in the core-clock (ECLK) domain, and a bank-based controller picks requests out of order on the memory-clock (DCLK) domain (while maintaining coherence) to minimize bank conflicts.

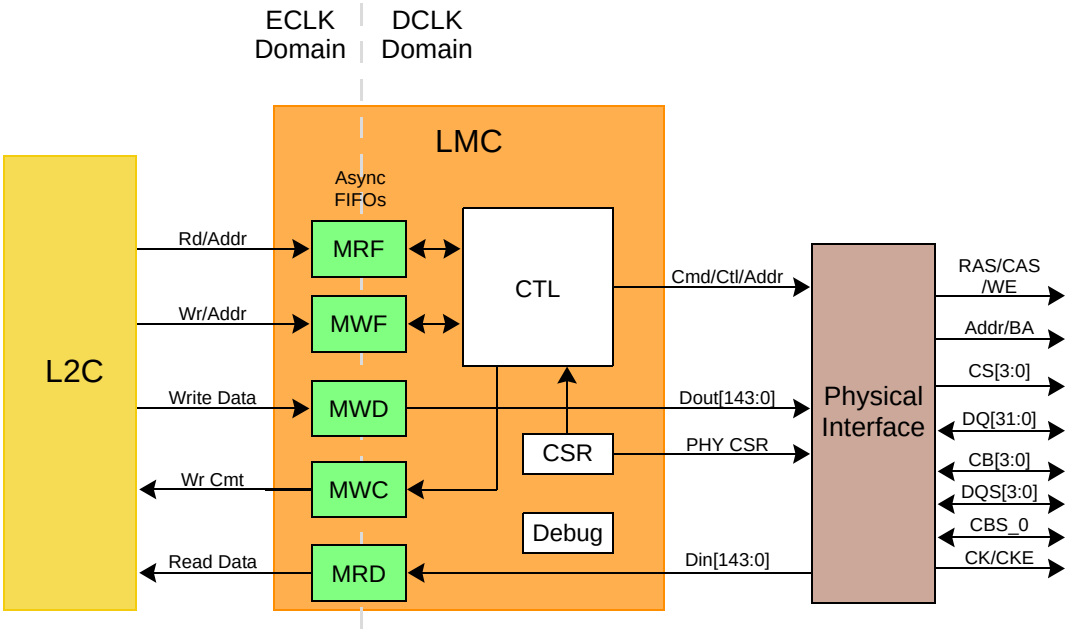


Figure 2–8 LMC Block Diagram

A cache-block address from L2C is decomposed into row-, column-, and bank-address based on programmable parameters stored in LMC_MEM_CFG0, LMC_CTL, and LMC_DDR2_CTL Commands are issued to the memory while meeting timing specifications as programmed in the CSR. All transactions autoclose each row after a cache block has been read/written. Refresh counters keep track of refresh intervals in order to refresh the memory within the maximum allowable time.

Once the command is committed to the bus, read and write commits are issued by the MRF and the memory-write-commit block (MWC) respectively, to L2C for bookkeeping. L2C pushes write data into the asynchronous memory-write-data register (MWD), which is written out at double-data rate along with generated ECC code. Read data from memory is accumulated in the asynchronous memory-read-data FIFO (MRD) and returned to L2C after ECC correction (SEC/DED) along with payload information.

Figure 2–9 shows a system configuration with CN50XX and two dual-ranked DIMMs (only Rank 0 is shown).

For a description of the memory signals, see Table 25–3.

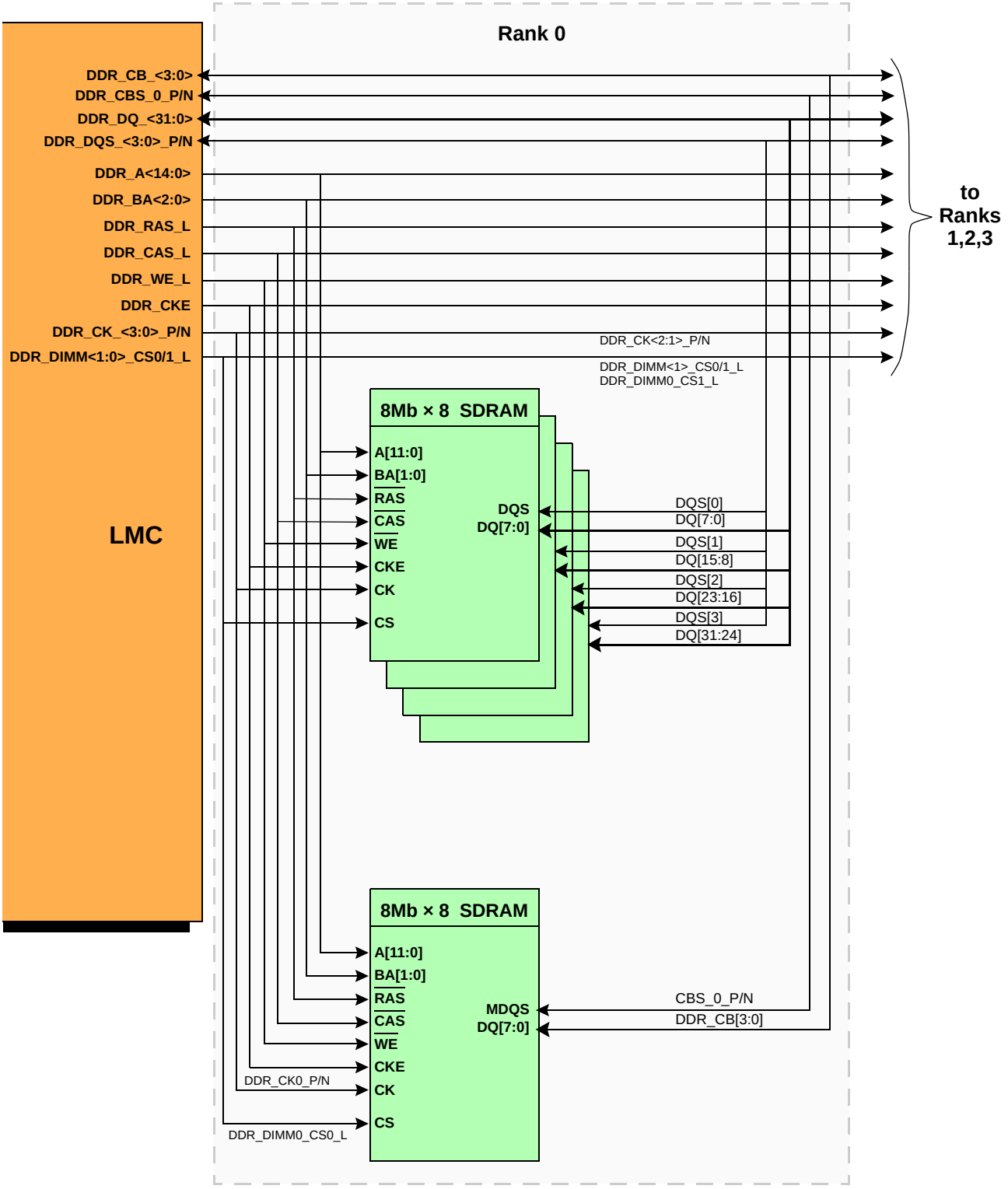


Figure 2–9 Memory System Configuration

2.3.1 Main Memory DRAM Addressing

All cached DRAM memory resides in an architected 36-bit address space that is in the lowest physical addresses. The first CN50XX implementation can contain up to 16GB of physical addresses available in the following physical address ranges:

DR0: 0x0 0000 0000 0000 to 0x0 0000 0FFF FFFF = low 256MB of SDRAM

DR1: 0x0 0004 1000 0000 to 0x0 0004 1FFF FFFF = mid 256MB of SDRAM

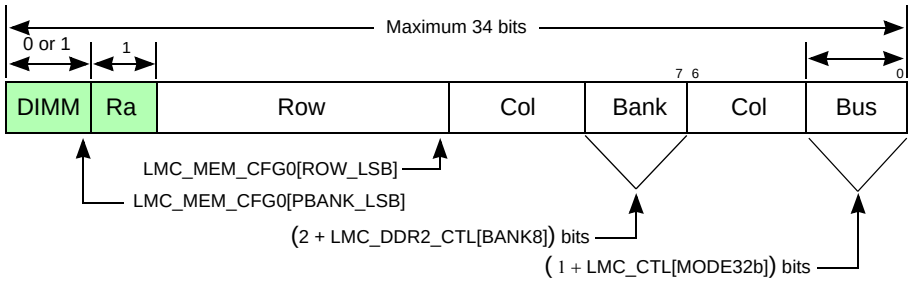
DR2: 0x0 0000 2000 0000 to 0x0 0003 FFFF FFFF = upper 15.5GB of SDRAM

- With 256MB of SDRAM or less, only DR0 is present.
- With 256MB – 512MB of SDRAM, DR0 and DR1 are present.
- With more than 512MB of SDRAM, DR0 and DR1 are present in entirety, and the lower portion of DR2 is present, according to the amount of SDRAM attached.

2.3.2 DRAM Part Addressing

Although a CN50XX physical address is 49 bits, the CN50XX addresses the attached main-memory DRAM parts with at most a 34-bit address. The CN50XX discards the upper 15 address bits from the DR0, DR1, or DR2 physical address to create the 34-bit DRAM-part address. The DRAM-part address is contiguous from 0 up to the size of the attached DRAM.

Figure 2–10 shows the partitioning of the (up to) 34-bit DRAM-part byte address.



Bus = selects the byte on the 18- or 36-bit DDR2 bus
Col = column address for the DDR2 part (9, 10, or 11 bits)
Bank = bank address for the DDR2 part (2 or 3 bits)
Row = row address for the DDR2 part (13, 14, or 15 bits)
Ra = optional rank address for a dual-rank DIMMs (present when LMC_CTL[BUNK_ENA] is set to 1)
DIMM = optional DIMM address (present with more than one DIMM with a 36-bit bus)

Figure 2–10 SDRAM Physical Address

The least-significant one or two bits of the DRAM part address determine the bus width.

- Bits <8:7> or <9:7> select the bank, depending whether there are four or eight banks in the attached parts, respectively. (LMC_DDR2_CTL[BANK8] selects the number of banks.)
- The column bits for the DRAM parts fill in the remainder of the lowest address bits, followed by the row bits for the DRAM parts.

- The most-significant bits of the DRAM part address select:
  - the rank within DIMM (if one is present, as selected by LMC_CTL[BUNK_ENA])
  - the DIMM (if two DIMMs are present with a 32/36-bit bus).

The bank selected to service a request is not just the Bank field shown in Figure 2–10. CN50XX also supports hashed bank selection when LMC_CTL[XOR_BANK] is set to 1. Table 2–5 shows the calculations to select the bank in different configurations:

Table 2–5 Bank Select

LMC_DDR2_CTL [BANK8]	LMC_CTL [XOR_BANK]	Selected Bank
0	0	byte_address<8:7>
0	1	byte_address<8:7> ⊕ byte_address<13:12>
1	0	byte_address<9:7>
1	1	byte_address<9:7> ⊕ byte_address<14:12>

Most applications will find it advantageous to use hashed bank selection (i.e. to set LMC_CTL[XOR_BANK] = 1).

CN50XX's bank selection has the following properties:

- All the locations within a single cache block reside in the same row.
- Sequential cache blocks are always/usually in different banks.

  Always when LMC_CTL[XOR_BANK] = 0

  Usually when LMC_CTL[XOR_BANK] = 1.
- Sequential cache blocks are usually in the same rank/DIMM.
- Bank distribution for skewed access when LMC_CTL[XOR_BANK] = 1.
  - Suppose an application uses many buffers whose size is 4KB or larger and a power-of-two.
  - Suppose, further, that the buffers are naturally-aligned based on their size and that the application only ever uses the first two (128B) cache blocks in any given buffer.

  With LMC_CTL[XOR_BANK] = 0, only banks 0 and 1 are used by the described application.

  With LMC_CTL[XOR_BANK] = 1, however, all four/eight banks are used by the described application.

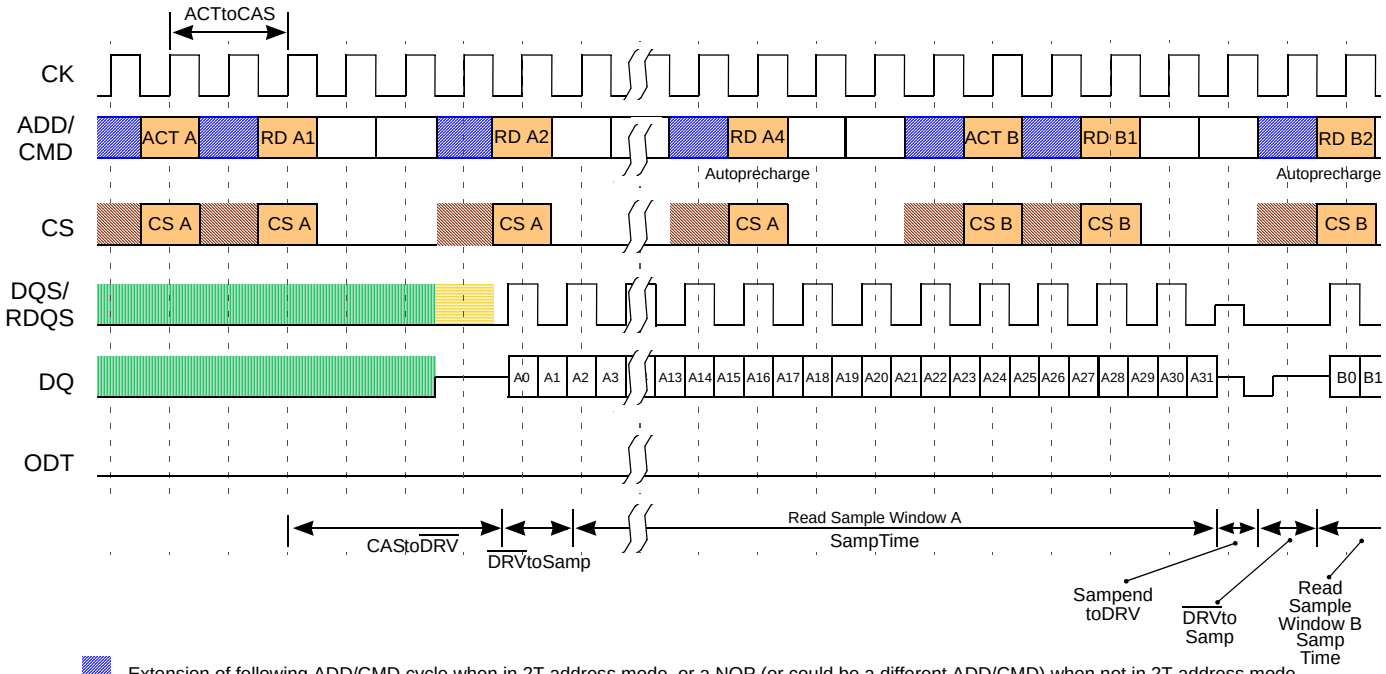
2.3.3 DRAM Transaction Examples

CN50XX always reads/writes entire 128-byte cache blocks on the main-memory DDR2 interface. There is a single address/command bus to drive all commands to all DIMMs/parts. The entire interface (32/36-bit) operates as one.

Figure 2–11shows two cache-block read transactions (A and B). A cache-block read transaction consists of one bank activate, one (or more) column-address commands, and the data transfer of the 128-byte cache block on the DRAM data bus.

Parameters For This Example:

T_{RCD} = 2	LMC_MEM_CFG0[SILO_QC=0]
T_{CL} = 3	LMC_MEM_CFG1[TRCD=2, CASLAT=3]
Board Delay = ¾CK	LMC_DDR2_CTL[POCAS=0, BURST8=1, SILO_HC=0]
	LMC_CTL[TSKW=1, RDIMM_ENA=0, FPRCH2=0, BPRCH=0, MODE32b=1]



Extension of following ADD/CMD cycle when in 2T address mode, or a NOP (or could be a different ADD/CMD) when not in 2T address mode

Always deasserted in 2T address mode

CN50XX drives deasserted

CN50XX does not drive DQS and forces DQS to read as 0s

minACTtoCAS = LMC_MEM_CFG1[TRCD] – (LMC_DDR2_CTL[POCAS] ? LMC_DDR2_CTL[ADDLAT] : 0) (can never be less than 2)

CAStoDRV = (LMC_DDR2_CTL[POCAS] ? LMC_DDR2_CTL[ADDLAT] : 0) + LMC_MEM_CFG1[CASLAT] – 1.5 + LMC_CTL[TSKW] + LMC_CTL[RDIMM_ENA] – LMC_CTL[FPRCH2]

DRVtoSamp + SampTime + SampendtoDRV = 10 + LMC_CTL[FPRCH] + LMC_CTL[BPRCH]

DRVtoSamp = 1 + LMC_CTL[FPRCH2] + 0.5 × LMC_DDR2_CTL[SILO_HC] + 0.25 × LMC_MEM_CFG0[SILO_QC]

SampTime = 16

SampendtoDRV = 1 – 0.5 × LMC_DDR2_CTL[SILO_HC] – 0.25 × LMC_MEM_CFG0[SILO_QC] + LMC_CTL[BPRCH]

Figure 2–11 Example of Two Consecutive Read Operations

CN50XX stops driving the data bus (and data strobes) during a programmable window after the column-address command, called the read no-drive window. (CN50XX is the default driver of the data bus and data strobes. CN50XX always drives 0s when it default-drives.)

LMC_CTL[TSKW] delays the entire read no-drive window to adjust for different board delays. LMC_CTL[FPRCH2] increases the read no-drive window by one clock cycle and starts it one clock cycle earlier. LMC_CTL[BPRCH] increases the read no-drive window by one cycle (but does not start it sooner).

Within the encompassing read no-drive window, there is a read-sample window, during which CN50XX obeys received-data-bus strobes. LMC_CTL[TSKW] delays the read-sample window by whole clock cycles, and LMC_DDR2_CTL[SILO_HC] and LMC_MEM_CFG0[SILO_QC] delay the read sample window by an additional ¼, ½, or ¾ clock cycle to compensate for the board delay (refer to Section 2.3.4 for more on programming for board delays). All edges of the read-data strobe that CN50XX receives from the DRAM part must reside entirely in the read-sample window. (CN50XX supports either differential, bidirectional DQS or single-ended, unidirectional RDQS, selected by lMC_DDR2_CTL[RDQS].)

The Figure 2–11 example shows two consecutive read operations that are not back-to-back. CN50XX can schedule back-to-back read operations when the two read operations reside in the same rank/DIMM and LMC_CTL[R2R_SLOT] = 0, unlike is shown in the figure. If LMC_CTL[R2R_SLOT] = 1, CN50XX forces a one clock bubble between all consecutive read operations, like the two read operations shown in Figure 2–11.

Figure 2–11 directly displays the CN50XX behavior when 2T addressing mode is not enabled (i.e. when LMC_DDR2_CTL[DDR2T] = 0), and also indicates the 2T addressing mode behavior. When 2T addressing is on, the timing could be the same, except that the address/command bus holds a stable value for an extra cycle prior to the chip-select assertion.

Though the specific example in Figure 2–11 does not use the DDR2 posted-CAS feature, CN50XX does support posted CAS, enabled and configured via the LMC_DDR2_CTL[POCAS,ADDLAT] fields. CN50XX cannot schedule a column-address command earlier than two cycles behind a bank activation, so the largest possible posted-CAS delay of $T_{MRD} - 1$ should not be used, as it results in poorer performance (higher latency, lower bandwidth) on CN50XX than a smaller posted-CAS delay. CN50XX fully supports posted-CAS delays ranging from 1 through $T_{MRD} - 2$.

CN50XX always precharges (via autoprecharges included in the last column-address command for the cache block) a BANK after it reads/writes an individual cache block from/to the bank.

The specific Figure 2–11 example does not use it, but CN50XX supports registered DIMMs. Registered DIMMs add a cycle of latency between the address/command bus and the data bus.

Figure 2–11 and the subsequent examples in this section have the following characteristics:

- They show CN50XX's ODT behavior on read operations. Refer to Section 2.3.6 for more details.
- They show behavior with a burst size of 8 (i.e. with LMC_DDR2_CTL[BURST8] set = 1).

  When the burst size is four, CN50XX similarly transfers cache blocks on the bus contiguously, but there are differences:

  - CN50XX issues twice as many column-address commands (RD A/RD B in Figure 2–11) to move a cache block with a burst of four.
  - When 2T address mode is used (i.e. when LMC_DDR2_CTL[DDR2T] = 1), performance is considerably worse with a burst of four than with a burst of 8. (100% of the ADD/CMD-bus bandwidth is needed to saturate the data bus for 2T address mode with a burst of four, so each bank activate causes a data-bus bubble in this case.)

    A burst length of eight is recommended whenever 2T address mode is used.
- They show behavior with a 32/36-bit bus. For a 16/18-bit bus, the CN50XX issues twice times as many column-address commands.

The number of column-address commands needed to move a cache block are listed in Table 2–6.

Table 2–6 Column-Address Commands

LMC_CTL [MODE32b]	LMC_DDR2_CTL [BURST8]	Number of Column-Address Commands
0	0	16
0	1	8
1	0	8
1	1	4

CN50XX always positions the column-address commands for a cache block for contiguous transfer of the cache block on the data bus. (The column-address commands for an individual cache block are exactly two cycles apart with a burst of four, and four cycles apart with a burst of eight and a 32/36-bit bus.) CN50XX always and only issues an autoprecharge with the final column-address command for a cache block.

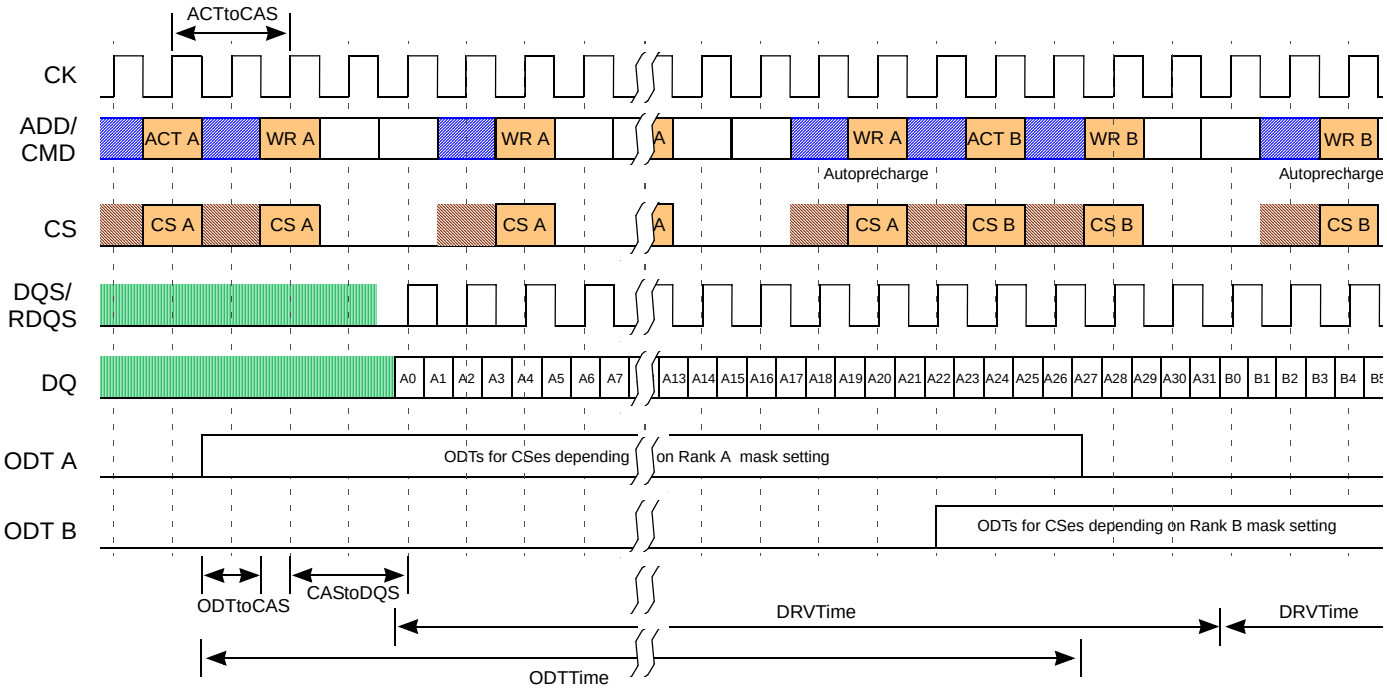
Figure 2–12 shows an example with two back-to-back cache-block write operations. As CN50XX is the default data-bus driver, write operations are simpler. CN50XX implements all of the following features for write operations:

- 2T address mode
- posted CAS
- registered DIMM
- burst size
- bus-width

When these features are used for read operations, the DDR2 parts require that they also be used for write operations. CN50XX also autoprecharges for the last column-address write operation as for read operations. CN50XX always attempts to schedule write data back-to-back.

Parameters For This Example:

T_{RCD} = 2	LMC_MEM_CFG0[SILO_QC=0]
T_{CL} = 3	LMC_MEM_CFG1[TRCD=2, CASLAT=3]
Board Delay = ¾CK	LMC_DDR2_CTL[POCAS=0, BURST8=1, SILO_HC=0]
	LMC_CTL[TSKW=1, RDIMM_ENA=0, FPRCH2=0, BPRCH=0, MODE32b=1]



Extension of following ADD/CMD cycle when in 2T address mode, or a NOP (or could be a different ADD/CMD) when not in 2T address mode

Always deasserted in 2T address mode

CN50XX drives deasserted (CN50XX is the default driver)

CN50XX does not drive DQS and forces DQS to read as 0s

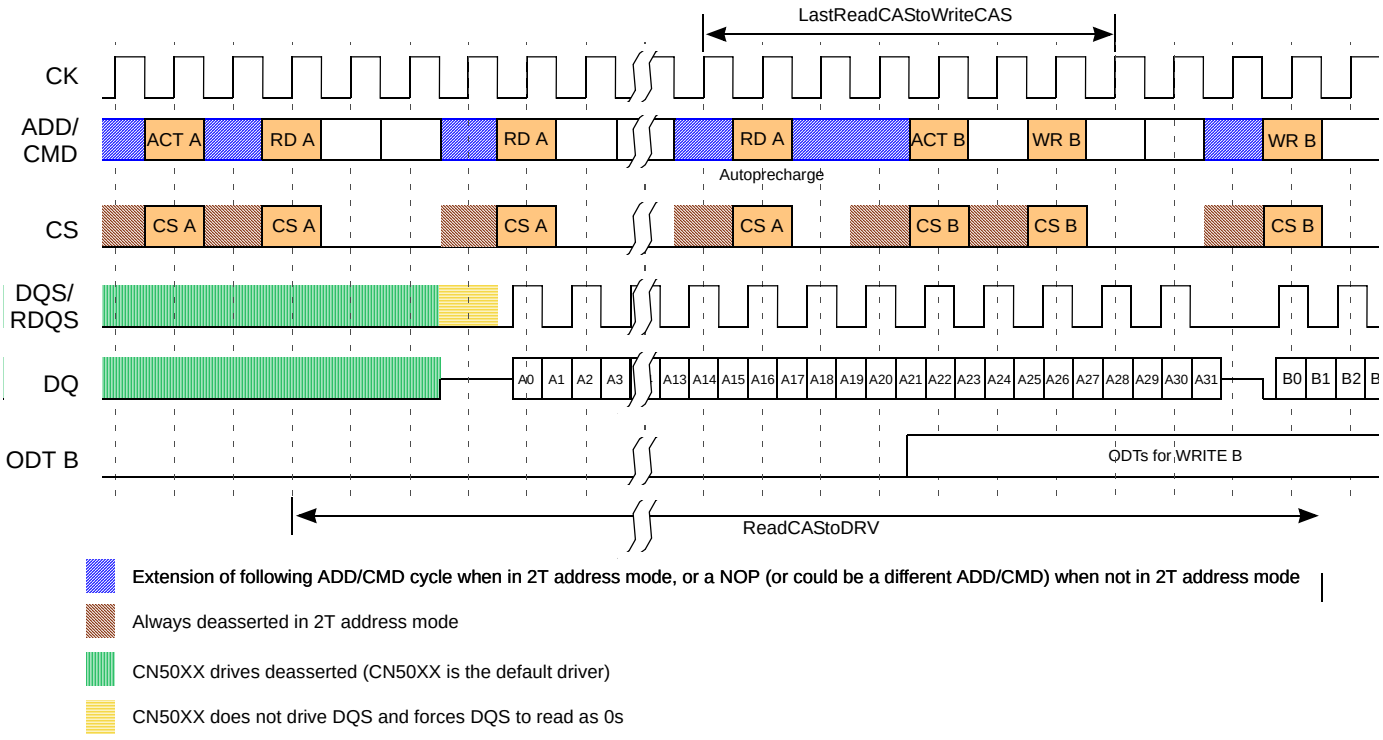
minACTtoCAS	= LMC_MEM_CFG1[TRCD] – (LMC_DDR2_CTL[POCAS] ? LMC_DDR2_CTL[ADDLAT] : 0) (can never be less than 2)
CAStoDQS	= (LMC_DDR2_CTL[POCAS] ? LMC_DDR2_CTL[ADDLAT] : 0) + LMC_MEM_CFG1[CASLAT] – 1 + LMC_CTL[RDIMM_ENA]
DRVTime	= 16
ODTtoCAS	= 4 – (LMC_DDR2_CTL[POCAS] ? LMC_DDR2_CTL[ADDLAT] : 0) – LMC_MEM_CFG1[CASLAT]
ODTTime	= 17

Figure 2–12 Example of Two Back-to-Back Cache-Block Write Operations

Figure 2–13 shows a cache-line read followed the earliest possible cache-line write.

Parameters For This Example:

T_{RCD} = 2	LMC_MEM_CFG0[SILO_QC=0]
T_{CL} = 3	LMC_MEM_CFG1[TRCD=2, CASLAT=3]
Board Delay = ¾CK	LMC_DDR2_CTL[POCAS=0, BURST8=1, SILO_HC=0]
	LMC_CTL[TSKW=1, RDIMM_ENA=0, FPRCH2=0, BPRCH=0, MODE32b=1]



minLastReadCAStoWriteCAS = LMC_CTL[TSKW] + LMC_CTL[BPRCH] + 2 × LMC_DDR2_CTL[BURST8] + 4

ReadCAStoDRV = (LMC_DDR2_CTL[POCAS] ? LMC_DDR2_CTL[ADDLAT] : 0) + LMC_MEM_CFG1[CASLAT] + LMC_CTL[TSKW] + LMC_CTL[RDIMM_ENA] + 2 × LMC_DDR2_CTL[BURST8] + LMC_CTL[BPRCH] + 2.5

Figure 2–13 Example of Cache-Line Read Operation Followed by Cache-Line Write Operation

2.3.4 DRAM Programming

Most of the parameters to the DRAM controller come from DRAM part and DIMM specifications. If desired, the TWSI interface can be used to read the SPD of the DIMMs, refer to Chapter 17. These part parameters go into fields in the LMC_MEM_CFG0, LMC_MEM_CFG1, LMC_CTL, and LMC_DDR2_CTL CSRs.

The most important remaining parameters account for the board delay inherent in any design. The LMC_CTL[TSKW], LMC_DDR2_CTL[SILO_HC], and LMC_MEM_CFG0[SILO_QC] parameters specify this delay.

Table 2–7 lists recommended parameters for different board delays:

Table 2–7 Recommended Parameters

Board Delay (DDR_CK Cycles)	LMC_CTL [TSKW]	LMC_DDR2_CTL [SILO_HC]	LMC_MEM_CFG0 [SILO_QC]	LMC_CTL [FPRCH2]	LMC_CTL [BPRCH]
0 – 0.25	0	1	0	1	0/1
0.25 – 0.5	0	1	1	1	1
0.5 – 0.75	1	0	0	1	0/1
0.75 – 1.0	1	0	1	1	0/1
1.0 – 1.25	1	1	0	1	0/1
1.25 – 1.5	1	1	1	1	1
1.5 – 1.75	2	0	0	1	0/1
1.75 – 2.0	2	0	1	1	0/1

2.3.5 DRAM Refreshes

LMC_MEM_CFG0[REF_INT] configures the DRAM refresh interval. When the refresh-interval counter expires, CN50XX does the following:

- stops creating new read and write operations
- completes the in-flight read and write operations
- generates a single DDR2 REF command.

2.3.6 DRAM Scheduler Performance

The DRAM scheduler aggressively reorders both read and write operations for optimal performance. The oldest references are higher priority, but if bank conflicts prevent CN50XX from scheduling the oldest, then CN50XX selects a younger reference.

The DRAM scheduler typically prioritizes read operations over write operations. It does, however, prioritize write operations over read operations if one of the following scenarios occur:

- The most recent transaction was a write operation, and there have not been LMC_CTL[MAX_WRITE_BATCH] consecutive write operations
- The L2C wants to access a cache block that is currently being written to memory.

Regardless of the priority between read and write operations, CN50XX still may schedule either read or write operations if bank conflicts prevent one type from being scheduled.

2.3.7 DRAM Chip Selects and ODT

CN50XX has four chip selects (DDR_*n*_CS_<1:0>_L) and supports up to two DIMMs, each of which can be dual-rank (i.e. when LMC_CTL[BUNK_ENA] = 1).

DIMM *n* uses the pair of chip-select signals DDR_*n*_CS_<1:0>_L.

- when LMC_CTL[BUNK_ENA] = 1, each chip-select signal in the pair asserts independently
- when LMC_CTL[BUNK_ENA] = 0, both chip-select signals in the pair assert together

The LMC assumes that all two DIMMs are present. If fewer are present, software should not reference the addresses that are not present (see Figure 2–10).

CN50XX has four ODT output signals (DDR_ODT_*m*) to drive DDR2-part ODT inputs, one output signal per chip select or DIMM/rank. Figures 2–11, 2–12, and 2–13 show where it is possible for CN50XX to assert its ODT outputs for different example transactions.

Table 2–8 shows the correspondence between the ODT signals and chip selects / DIMMs / ranks.

Table 2–8 ODT Signals and Corresponding Chip Select/DIMM/Rank Signals

ODT Signal	Chip Select	Comment
DDR_ODT_<0>	DDR_0_CS_<0>_L	<31:0> DIMM 0, rank 0
DDR_ODT_<1>	DDR_0_CS_<1>_L	<31:0> DIMM 0, rank 1
DDR_ODT_<2>	DDR_1_CS_<0>_L	<31:0> DIMM 1, rank 0
DDR_ODT_<3>	DDR_1_CS_<1>_L	<31:0> DIMM 1, rank 1

CN50XX does not assert an ODT signal for a write operation when the transaction references the corresponding chip select. Typically, a system may desire all other ODT signals to assert for a transaction. CN50XX can provide this and other patterns depending on the LMC_RODT_COMP_CTL and LMC_WODT_CTL configuration. CN50XX can produce different ODT signal combinations for write operations, and for transactions that reference different DIMMs/ranks.

Table 2–9 shows the configuration variables that determine whether each ODT signal asserts for write operations to different DIMMs/ranks.

Table 2–9 Write ODT Configuration Variables

Write on Chip Select #	ODT[7:4]	ODT[3:0]
DDR_0_CS_0_L	WODT_HI3	WODT_LO0
DDR_0_CS_1_L	WODT_HI3	WODT_LO1
DDR_1_CS_0_L	WODT_HI3	WODT_LO2
DDR_1_CS_1_L	WODT_HI3	WODT_LO3

The ODT-enable interval for back-to-back write operations can overlap. These two write operations may be to different DIMMs/ranks, so may have different ODT values. CN50XX asserts an ODT signal whenever it should assert due to any ODT-enable interval.

If ODT is not used, all of LMC_CTL[QS_DIC], LMC_RODT_COMP_CTL, and LMC_WODT_CTL should be cleared to 0s.

During a memory read operation, read ODT control is available at the receivers of CN50XX. Through LMC_RODT_COMP_CTL, three different read ODT conditions are provided:

- no ODT
- weak ODT
- strong ODT

Read ODT is automatically off during a memory write operation, even if a weak-ODT or strong-ODT condition is chosen. Table 2–10 shows the configuration of read ODT setting through LMC_RODT_COMP_CTL register.

Table 2–10 Configuration of Read ODT Setting

ODT Setting	LMC_RODT_COMP_CTL		
	[ENABLE]	[PCTL]	[NCTL]
No ODT	0	—	—
Weak ODT	1	00011	0001
Strong ODT	1	00111	0010

2.3.8 DRAM Controller Initialization

There are three parts to the LMC-initialization procedure:

1. DCLK initialization
2. DRESET initialization
3. LMC initialization

During a cold reset, all three initializations should be executed in sequence. The DCLK initialization need only be performed once if the DCLK speed and parameters do not change. Subsequently, it is possible to execute only DRESET and LMC initializations, or to execute only the LMC initialization.

2.3.8.1 DCLK Initialization Sequence

Perform the following steps to initialize the DCLK.

1. Write LMC_CTL[DRESET]=1, LMC_DDR2_CTL[QDLL_ENA]=0.
2. Write LMC_PLL_CTL[CLKR, CLKF, EN*] with the appropriate values, while writing LMC_PLL_CTL[RESET_N] = 0, LMC_PLL_CTL[DIV_RESET] = 1. LMC_PLL_CTL[CLKR, CLKF, EN*] values must not change after this point without restarting the DCLK initialization sequence.

   Section 2.3.9 describes the DDR_CK frequencies resulting from different reference-clock values and programmings.
3. Read L2D_BST0 and wait for the result.
4. Wait 5 µsec.
5. Write LMC_PLL_CTL[RESET_N] = 1 while keeping LMC_PLL_CTL[DIV_RESET] = 1. LMC_PLL_CTL[RESET_N] must not change after this point without restarting the DCLK initialization sequence.
6. Read L2D_BST0 and wait for the result.
7. Wait 500 × (LMC_PLL_CTL[CLKR] + 1) reference-clock cycles.
8. Write LMC_PLL_CTL[DIV_RESET] = 0. LMC_PLL_CTL[DIV_RESET] must not change after this point without restarting the DCLK initialization sequence.
9. Read L2D_BST0 and wait for the result.

The DDR address clock frequency (DDR_CK_<5:0>_P/N) should be stable at that point.

2.3.8.2 DRESET Initialization Sequence

The DRESET initialization sequence cannot start unless DCLK is stable due to a prior DCLK initialization sequence. Perform the following steps to initialize DRESET.

1. Write LMC_CTL[DRESET] = 1 and LMC_DDR2_CTL[QDLL_ENA] = 0.
2. Write LMC_DDR2_CTL[QDLL_ENA] = 1. LMC_DDR2_CTL[QDLL_ENA] must not change after this point without restarting the LMC and/or DRESET initialization sequence.
3. Read L2D_BST0 and wait for the result.
4. Wait 10 µsec.
5. Write LMC_CTL[DRESET] = 0. LMC_CTL[DRESET] must not change after this point without restarting the DRAM-controller and/or DRESET initialization sequence.
6. Read L2D_BST0 and wait for the result.

2.3.8.3 LMC Initialization Sequence

The LMC initialization sequence must be preceded by a DCLK and DRESET initialization sequence.

1. Software must ensure there are no pending DRAM transactions.
2. Write LMC_CTL, LMC_CTL1, LMC_MEM_CFG1, LMC_DDR2_CTL, LMC_RODT_COMP_CTL, LMC_DUAL_MEMCFG, and LMC_WODT_CTL with appropriate values, if necessary. Refer to Sections 2.3.4, 2.3.5, and 2.3.7 regarding these registers (and LMC_MEM_CFG0).
3. Write LMC_MEM_CFG0 with appropriate values and LMC_MEM_CFG0[INIT_START] = 0.
4. Write LMC_MEM_CFG0 with appropriate values and LMC_MEM_CFG0[INIT_START] = 1. At that point, CN50XX hardware initiates the standard DDR2 init sequence shown in Figure 2–14.

   CN50XX activates DDR_CKE (if it has not already been activated). DDR_CKE remains activated from that point until a subsequent DRESET.

   CN50XX then follows with the standard DDR2 initialization sequence, not using OCD. While CN50XX performs this initialization sequence, it cannot perform other DDR2 transactions.

   Note that if there is not a DRESET between two LMC initialization sequences, DDR_CKE remains asserted through the second sequence. The hardware initiates the same DDR2 initialization sequence as the first, except that DDR_CKE does not deactivate. If DDR_CKE deactivation and reactivation is desired for a second controller reset, a DRESET sequence is required.

5. Read L2D_BST0 and wait for the result.

After this point, the LMC is fully functional. LMC_MEM_CFG0[INIT_START] should not transition from 0→1 during normal operation.

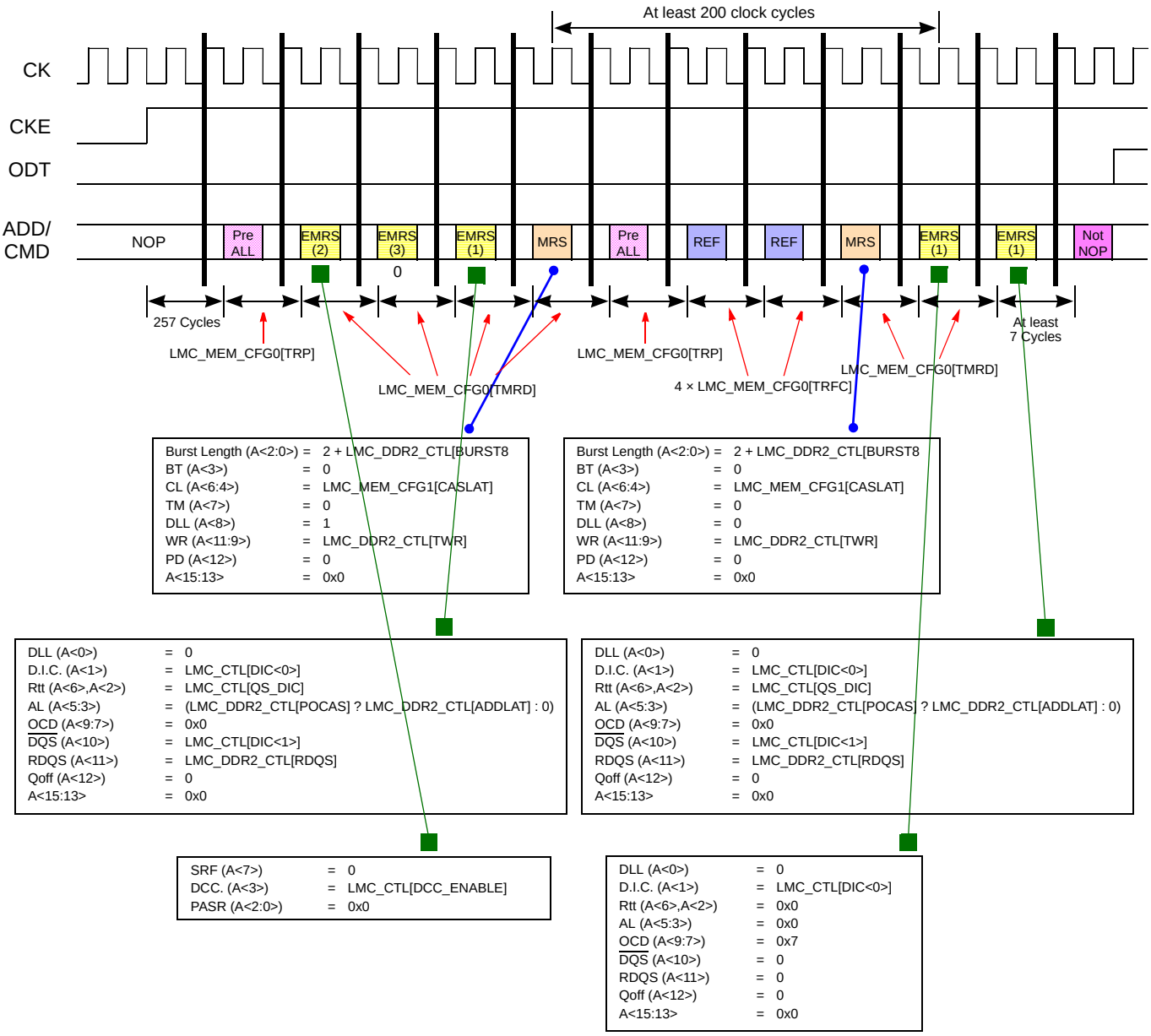


Figure 2–14 Standard DDR2 Initialization Procedure

2.3.9 DDR Clock-Speed Programming Tables

The resultant address-clock frequency (DDR_CK_<5:0>_P/N) is determined by the following formula:

$$\text{DDR\_CK frequency (MHz)} = 50 \times \frac{(\text{CLKF} + 1)}{(\text{CLKR} + 1) \times \text{EN}(2,4,6,8,12,16)}$$

where:

CLKR = LMC_PLL_CTL[CLKR]

50 = PLL_REF_CLK frequency

CLKF = LMC_PLL_CTL[CLKF]

EN(2,4,6,8,12,16) = 2, 4, 6, 8, 12, or 16, depending on the LMC_PLL_CTL[EN*] programming.

The PLL frequency is determined by the following formula:

$$\text{PLL frequency (MHz)} = 50 \times \frac{\text{CLKF} + 1}{\text{CLKR} + 1}$$

The PLL frequency must reside between 1.2 GHz and 2.5 GHz. A PLL frequency closer to 2.5 GHz is desirable if there is a choice.

The following example shows sample programming settings.

Example 2–1 Sample Programming Settings

Frequency	LMC0_PLL_CTL		
DDR_CK/data rate	[CLKF]	[EN*]	[CLKR]
200/400 MHz	95	EN12	1
250/500 MHz	79	EN8	1
267/533 MHz	63	EN6	1
300/600 MHz	71	EN6	1
333/667 MHz	79	EN6	1

NOTE: LMC_PLL_CTL[FASTEN_N] changes the allowed CLKF values.

CLKF limits are controlled by [FASTEN_N] as follows:

1 = 128 < CLKF ≤ 256;
0 = 0 < CLKF ≤ 128.

For more details on DDR clock-speed programming, see Section 2.3.8.

2.3.10 DRAM ECC Codes

Table 2–11 shows the DRAM 64-bit ECC code for core 64-bit references.

Table 2–11 DRAM 64-Bit ECC Code

cccccccc 7:0	63:60	59:59	55:52	51:48	47:44	43:40	39:36	35:32	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0	
00000001	1011	0100	1101	0001	1011	0100	1101	0001	0100	1011	0010	1110	0100	1011	0010	1110	0xB4D1B4D14B2E4B2E
00000010	0001	0101	0101	0111	0001	0101	0101	0111	0001	0101	0101	0111	0001	0101	0101	0111	0x1557155715571557
00000100	1010	0110	1001	1001	1010	0110	1001	1001	1010	0110	1001	1001	1010	0110	1001	1001	0xA699A699A699A699
00001000	0011	1000	1110	0011	0011	1000	1110	0011	0011	1000	1110	0011	0011	1000	1110	0011	0x38E338E338E338E3
00010000	1100	0000	1111	1100	1100	0000	1111	1100	1100	0000	1111	1100	1100	0000	1111	1100	0xC0FCC0FCC0FCC0FC
00100000	1111	1111	0000	0000	1111	1111	0000	0000	1111	1111	0000	0000	1111	1111	0000	0000	0xFF00FF00FF00FF00
01000000	1111	1111	0000	0000	0000	0000	1111	1111	1111	1111	0000	0000	0000	0000	1111	1111	0xFF0000FFFF0000FF
10000000	0000	0000	1111	1111	1111	1111	0000	0000	1111	1111	0000	0000	0000	0000	1111	1111	0x00FFFF00FF0000FF
00000000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
xxxxxxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	

Table 2–11 DRAM 64-Bit ECC Code

cccccccc 7:0	63:60	59:59	55:52	51:48	47:44	43:40	39:36	35:32	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0	
84210000	7766	6666	9999	9988	BBAA	AAAA	5555	5544	FFEE	EEEE	1111	1100	3322	2222	DDDD	DDCC	← Syndromes
00008421	50DB	8742	DB87	42AF	50DB	8742	DB87	42AF	41CA	9653	CA96	53BE	41CA	9653	CA96	53BE	

2.4 L2C Registers

The L2C registers are listed in Table 2–12.

Table 2–12 Level 2 Cache Registers

Register	Address	CSR Type[1]	Detailed Description
L2C_CFG	0x0001180080000000	RSL	See page 85
L2T_ERR	0x0001180080000008	RSL	See page 86
L2D_ERR	0x0001180080000010	RSL	See page 87
L2D_FADR	0x0001180080000018	RSL	See page 87
L2D_FSYN0	0x0001180080000020	RSL	See page 88
L2D_FSYN1	0x0001180080000028	RSL	
L2C_DBG	0x0001180080000030	RSL	See page 89
L2C_LFB0	0x0001180080000038	RSL	See page 90
L2C_LFB1	0x0001180080000040	RSL	See page 91
L2C_LFB2	0x0001180080000048	RSL	See page 91
L2C_DUT	0x0001180080000050	RSL	See page 92
L2C_LCKBASE	0x0001180080000058	RSL	See page 93
L2C_LCKOFF	0x0001180080000060	RSL	See page 94
L2C_SPAR0	0x0001180080000068	RSL	See page 94
L2C_SPAR4	0x0001180080000088	RSL	See page 94
L2C_PFCTL	0x0001180080000090	RSL	See page 95
L2C_PFC0 ... L2C_PFC3	0x0001180080000098 ... 0x00011800800000B0	RSL	See page 97
L2C_LFB3	0x00011800800000B8	RSL	See page 91
L2D_BST0	0x0001180080000780	RSL	See page 98
L2D_BST1	0x0001180080000788	RSL	See page 98
L2D_BST2	0x0001180080000790	RSL	See page 99
L2D_BST3	0x0001180080000798	RSL	See page 99
L2D_FUS0	0x00011800800007A0	RSL	See page 100
L2D_FUS1	0x00011800800007A8	RSL	See page 100
L2D_FUS2	0x00011800800007B0	RSL	See page 101
L2D_FUS3	0x00011800800007B8	RSL	See page 101
L2C_BST2	0x00011800800007E8	RSL	See page 104
L2C_BST1	0x00011800800007F0	RSL	See page 103
L2C_BST0	0x00011800800007F8	RSL	See page 103

1. RSL-type registers are accessed indirectly across the I/O bus.

Level 2 Cache Configuration Register
L2C_CFG

See Table 2–12 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:20>	—	RAZ	—	—	Reserved.
<19>	BSTRUN	RO	0	0	L2 data store BIST running. Indicates when the L2C hardware BIST sequence (short or long) is running. [L2C ECC BIST FSM is not in the RESET/DONE state.]
<18>	LBIST	R/W	0	0	L2C data store long BIST sequence. When the previous state was 0 and software writes a 1, the long BIST sequence (enhanced 13N March) is performed. Software can then read the L2C_CFG[BSTRUN] which will indicate that the long BIST sequence is running. When BSTRUN=0, the state of the L2D_BST[0–3] registers contain information which reflects the status of the recent long BIST sequence. **NOTE:** Software must never write LBIST=0 while Long BIST is running (i.e.: when BSTRUN=1 never write LBIST=0).
<17:15>	—	R/W	0x0	0x0	Spare bits.
<14>	—	RAZ	—	—	Reserved.
<13:10>	FPEXP	R/W	0x0	0x0	MBZ.
<9>	FPEMPTY	R/W	0	0	MBZ.
<8>	FPEN	R/W	0	0	MBZ.
<7>	IDXALIAS	R/W	0	0	L2C index alias enable. When set, the L2 tag/data store aliases the 7-bit index with the low-order 7 bits of the tag. index[13:7] = (tag[20:14] ⊕ index[13:7]) **NOTE:** This bit must be modified only at boot time, when it can be guaranteed that no blocks have been loaded into the L2 cache. The index aliasing is a performance-enhancement feature that reduces the L2-cache thrashing experienced for regular stride references.
<6:3>	MWF_CRD	R/W	2	2	MWF credit threshold. When the remaining MWF credits become less than or equal to the MWF_CRD, the L2C asserts L2C__LMI_MWD_HIWATER_A to signal the LMC to give write operations (victims) higher priority.
<2>	RSP_ARB_MODE	R/W	1	1	RSP arbitration mode: 0 = Fixed Priority [HP=RFB, RMCF, RHCF, STRSP, LP=STRSC] 1 = Round Robin: [RFB(reflected I/O), RMCF(RdMiss), RHCF(RdHit), STRSP(ST RSP w/ invalidate), STRSC (ST RSP no invalidate)]
<1>	RFB_ARB_MODE	R/W	1	1	RFB arbitration mode: 0 = Fixed Priority – IOB→PP requests are higher priority than core→IOB requests 1 = Round Robin – I/O requests from core and IOB are serviced in round robin

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<0>	LRF_ARB_MODE	R/W	1	1	RF arbitration mode: 0 = Fixed Priority – IOB memory requests are higher priority than core memory requests. 1 = Round Robin – Memory requests from core and IOB are serviced in round robin.

Level 2 Cache Tag Errors
L2T_ERR

L2 tag ECC SEC/DED errors and interrupt enable. See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:28>	—	RAZ	0	0	Reserved
<27>	LCK_INTENA2	R/W	0	1	L2 tag lock error2 interrupt enable bit
<26>	LCKERR2	R/W1C	0	0	Hardware detected a case where a Rd/Wr Miss from core *n* could not find an available/unlocked set (for replacement). Most likely, this is a result of software mixing SET PARTITIONING with ADDRESS LOCKING. If software allows another core to LOCKDOWN all SETs available to core *n*, then a Rd/Wr Miss from core *n* will be unable to determine a valid replacement set (since LOCKED addresses should never be replaced). If such an event occurs, the hardware selects the smallest available SET (specified by UMSK*n*) as the replacement set, and the address is unlocked.
<25>	LCK_INTENA	R/W	0	1	L2 tag lock error interrupt enable bit
<24>	LCKERR	R/W1C	0	0	Software attempted to lock down the last set of the index (which is ignored by hardware but reported to software). **NOTE:** Available sets takes the L2C_SPAR*n*[UMSK*m*] into account. For example, if the diagnostic core has UMSK*m* defined to only use SETs [1:0], and SET1 had been previously LOCKED, then an attempt to LOCK the last available SET0 would result in a LCKERR.
<23:21>	FSET	RO	0	0	Failing L2 tag set number (1 of 8)
<20:18>	—	RAZ	0x0	0x0	Reserved
<17:11>	FADR	RO	0	0	Failing L2 tag address (7-bit index). When L2T_ERR[SEC_ERR] or L2T_ERR[DED_ERR] are set, the FADR contains the lower 7-bit cacheline index into the L2 tag store.
<10:5>	FSYN	RO	0	0	When L2T_SEC_ERR or L2T_DED_ERR are set, this field contains the failing L2 tag ECC 5-bit syndrome. (A DED error always overwrites a SEC error SYNDROME and FADR).
<4>	DED_ERR	R/W1C	0	0	L2T double-bit error detected (DED)
<3>	SEC_ERR	R/W1C	0	0	L2T single-bit error corrected (SEC)
<2>	DED_INTENA	R/W	0	0	L2 tag ECC double-error detect (DED) interrupt enable bit. When set, allows interrupts to be reported on double-bit (uncorrectable) errors from the L2 tag arrays.
<1>	SEC_INTENA	R/W	0	0	L2 Tag ECC single-error correct (SEC) interrupt enable bit. When set, allows interrupts to be reported on single-bit (correctable) errors from the L2 tag arrays.
<0>	ECC_ENA	R/W	0	0	L2 tag ECC enable. When set, enables 6-bit SEC/DED codeword for 23-bit L2 tag arrays {V,D,L,TAG[33:14]}

Level 2 Cache Data Errors Register
L2D_ERR

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	0x0	0x0	Reserved
<5>	BMHCLSEL	R/W	0	0	L2 bit-map half cache line ECC selector. When L2C_DBG[L2T] = 1 and L2D_ERR[ECC_ENA] = 0, the BMHCLSEL selects which half cacheline to conditionally latch into the L2D_FSYN0/L2D_FSYN1 registers when an LDD is detected from the diagnostic core (determined by L2C_DBG[PPNUM]). 0 = OW[0-3] ECC (from first ½ cacheline) is selected to be conditionally latched into the L2D_FSYN0/1 CSRs. 1 = OW[4-7] ECC (from last ½ cacheline) is selected to be conditionally latched into the L2D_FSYN0/1 CSRs.
<4>	DED_ERR	R/W1C	0	0	L2D double-bit error detected (DED)
<3>	SEC_ERR	R/W1C	0	0	L2D single-bit error corrected (SEC)
<2>	DED_INTENA	R/W	0	1	L2 data ECC double error detect (DED) interrupt enable bit. When set, allows interrupts to be reported on double-bit (uncorrectable) errors from the L2 data arrays.
<1>	SEC_INTENA	R/W	0	1	L2 data ECC single error correct (SEC) interrupt enable bit. When set, allows interrupts to be reported on single-bit (correctable) errors from the L2 data arrays.
<0>	ECC_ENA	R/W	0	1	L2 data ECC enable. When set. enables 10-bit SEC/DED codeword for 128-bit L2 data arrays.

Level 2 Cache Failing Address
L2D_FADR

L2 data ECC SEC/DED failing address. When L2D_SEC_ERR or L2D_DED_ERR are set, this CSR contains the failing L2 data-store index. A DED error always overwrites an SEC error SYNDROME and FADR bits. See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:18>	—	RAZ	0x0	0x0	Reserved
<17:14>	FOWMSK	RO	0x0	0x0	Failing OW mask. Indicates which one of four OWs contained SEC/DED error.
<13:11>	FSET	RO	0x0	0x0	Failing SET number.
<10:8>	—	RAZ	0x0	0x0	Reserved
<7:0>	FADR	RO	0x0	0x0	Failing L2 data-store lower index bits FADR[7:1]: cacheline index[13:7] FADR[0]: 1/2 cacheline index **NOTE:** L2 Data Store Index is for each 1/2 cacheline.

Level 2 Cache Failing Syndrome (OW0/1/4/5) Register L2D_FSYN0

L2 data ECC SEC/DED failing syndrome for lower cache line. When L2D_SEC_ERR or L2D_DED_ERR are set, this CSR contains the failing L2 data ECC 10-bit syndrome. A DED error always overwrites an SEC Error SYNDROME and FADR bits. See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:20>	—	RAZ	0	0	Reserved
<19:10>	FSYN_OW1	RO	0	0	Failing L2 data store SYNDROME OW1 or OW5. When L2D_ERR[ECC_ENA] = 1 and either L2D_ERR[SEC_ERR] = 1 or L2D_ERR[DED_ERR] = 1, this field represents the failing OWECC syndrome for the half cache line indexed by L2D_FADR[FADR]. **NOTE:** The L2D_FADR[FOWMSK] further qualifies which OW lane (one of four) detected the error. When L2C_DBG[L2T] = 1 and L2D_ERR[ECC_ENA] = 0, an LDD from the diagnostic core conditionally latches the raw OWECC for the selected half cache line. (see L2D_ERR[BMHCLSEL.
<9:0>	FSYN_OW0	RO	0	0	Failing L2 data store SYNDROME OW0 or OW4. When L2D_ERR[ECC_ENA] = 1 and either L2D_ERR[SEC_ERR] = 1 or L2D_ERR[DED_ERR] = 1, this field represents the failing OWECC syndrome for the half cache line indexed by L2D_FADR[FADR]. **NOTE:** The L2D_FADR[FOWMSK] further qualifies which OW lane (one of four) detected the error. When L2C_DBG[L2T] = 1 and L2D_ERR[ECC_ENA] = 0, an LDD (L1 load-miss) from the diagnostic core conditionally latches the raw OWECC for the selected half cache line. (see L2D_ERR[BMHCLSEL].

Level 2 Cache Failing Syndrome (OW2/3/6/7) Register L2D_FSYN1

L2 data ECC SEC/DED failing syndrome for upper cache line. See Table 2–12 for address. When L2D_SEC_ERR or L2D_DED_ERR are set, this field contains the failing L2 data ECC 10-bit syndrome. (A DED error always overwrites an SEC Error SYNDROME and FADR).

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:20>	—	RAZ	0	0	Reserved
<19:10>	FSYN_OW3	RO	0	0	Failing L2 data store SYNDROME OW3 or OW7.
<9:0>	FSYN_OW2	RO	0	0	Failing L2 data store SYNDROME OW2 or OW6.

Level 2 Cache Debug Register
L2C_DBG

L2C tag/data store debug register. See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:14>	—	RAZ	0x0	0x0	Reserved
<13:11>	LFB_ENUM	R/W	0x0	0x0	Specifies the L2C inflight buffer (LFB) entry number that is to be captured.
<10>	LFB_DMP	R/W	0	0	LFB dump enable. When written to 1, the contents of the LFB specified by LFB_ENUM[2:0] are captured into the L2C_LFB(0/1/2) registers.
<9:7>	—	RAZ	0x0	0x0	Reserved
<6>	PPNUM	R/W	0	0	When L2T, L2D, or FINV are enabled, this field determines which 1-of-2 cores is selected as the diagnostic core.
<5:3>	SET	R/W	0	0	When L2T, L2D, or FINV are enabled, this field determines 1-of-4 targeted sets to act upon.
<2>	FINV	R/W	0	0	Flush-invalidate. When set to 1, all STF write commands generated from the diagnostic core (determined by PPNUM) invalidate the specified set (determined by SET) at the index specified in the STF address[13:7]. If a dirty block is detected (D=1), it is written back to memory. The contents of the invalid L2 cache line is also “scrubbed” with the STF write data. **NOTE:** If L2C_CFG[IDXALIAS] = 1, the index specified in STF address <13:7> refers to the aliased address. **NOTE:** An STF command with write data = 0s can be generated by software using the Prefetch instruction with Hint=30d “prepare for Store”, followed by a SYNCW. What is seen at the L2C as an STF w/wrdcnt=0 with all of its mask bits clear (indicates zero-fill data). A flush-invalidate will “force-hit” the L2 cache at [index, set] and invalidate the entry (V=0/D=0/L=0/U=0). If the cache block is dirty, it is also written back to memory. The DuTag state is probed/updated as normal for an STF request. Typical Applications: • L2 Tag/Data ECC software recovery • Cache unlocking **NOTE:** If the cache line had been previously locked, a flush-invalidate operation explicitly unlocks the set/index specified. **NOTE:** The diagnostic core can generate STF commands to L2C whenever all 128 bytes in a block are written. Software must take this into consideration to avoid errant flush-invalidate operations.
<1>	L2D	R/W	0	0	When enabled (and L2T = 0), fill data is returned directly from the L2 data store (regardless of hit/miss) when an LDD instruction is issued from a core determined by the PPNUM field. The selected set# is determined by the SET field, and the index is determined from the address[13:7] associated with the LDD instruction. This “force-hit” does not alter the current L2 tag state or the DuTag state.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<0>	L2T	R/W	0	0	When enabled, L2 tag information {V,D,L,U,TAG[33:14]} is returned on the low-order 2 bits of OW2, OW6 when an LDD instruction is issued from a core determined by the PPNUM field. The selected set# is determined by the SET field, and the index is determined from the address[13:7] associated with the LDD instruction. This “force-hit” does not alter the current L2 tag state or the DuTag state. **NOTE:** The diagnostic core should issue a D-stream load command to an aligned cacheline + 0x20 (+ 0x60) in order to have the return VDLUTAG information (in OW2) written directly into the proper core register. The diagnostic core should also flush its local L1 cache after use (to ensure data coherency). **NOTE:** (For L2C bitmap testing of L2 data store OW ECC): If L L2D_ERR[ECC_ENA] = 0, the OW ECC from the selected half cacheline (see L2D_ERR[BMHCLSEL]) is also conditionally latched into the L2D_FSYN0/1 CSRs if an LDD is detected from the diagnostic core (PPNUM).

Notes:

- When using the L2T, L2D, or FINV debug probe feature, the LDD instruction does not update the DuTags.
- L2T, L2D, and FINV must be mutually exclusive (only one set).
- Force Invalidate is intended as a means for software to invalidate the L2 cache while also writing back dirty data to memory to maintain coherency.
- L2 cache lock down feature must be disabled (L2C_LCKBASE[LCK_ENA]=0) if any of the L2C debug features (L2T, L2D, FINV) are enabled.

Level 2 Cache In-Flight Address Buffer Debug 0 Register L2C_LFB0

Contains L2C in-flight address buffer (LFB) contents (status bits). See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	0x0	0x0	Reserved.
<31>	STCPND	RO	0	0	LFB STC pending status
<30>	STPND	RO	0	0	LFB ST* pending status
<29>	STINV	RO	0	0	LFB ST* invalidate status
<28>	STCFL	RO	0	0	LFB STC = FAIL status
<27>	VAM	RO	0	0	Valid full address match status
<26>	—	RAZ	0	0	Reserved.
<25:23>	INXT	RO	0x0	0x0	Next LFB pointer
<22>	ITL	RO	0	0	LFB tail of list indicator
<21>	IHD	RO	0	0	LFB head of list indicator
<20:18>	SET	RO	0x0	0x0	SET number used for DS-OP (hit = hset/miss = rset)
<17>	—	RAZ	0	0	Reserved
<16:14>	VABNUM	RO	0x0	0x0	VAB number used for LMC miss launch
<13:5>	SID	RO	0x0	0x0	LFB source ID
<4:1>	CMD	RO	0x0	0x0	LFB command
<0>	VLD	RO	0	0	LFB valid

Level 2 Cache In-Flight Address Buffer Debug 1 Register L2C_LFB1

Contains L2C LFB contents (wait bits). See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:19>	—	RAZ	0x0	0x0	Reserved
<18>	DSGOING	RO	0	0	LFB DS going (in flight).
<17:16>	BID	RO	0	0	LFB DS bid#
<15>	WTRSP	RO	0	0	LFB waiting for RSC response [FILL,STRSP] completion
<14>	WTDW	RO	0	0	LFB waiting for DS-WR completion
<13>	WTDQ	RO	0	0	LFB waiting for LFB-DQ
<12>	WTWHP	RO	0	0	LFB waiting for write-hit partial L2 DS-WR completion
<11>	WTWHF	RO	0	0	LFB waiting for write-hit full L2 DS-WR completion
<10>	WTWRM	RO	0	0	LFB waiting for write-miss L2 DS-WR completion
<9>	WTSTM	RO	0	0	LFB waiting for write-miss L2 DS-WR completion
<8>	WTRDA	RO	0	0	LFB waiting for read-miss L2 DS-WR completion
<7>	WTSTDT	RO	0	0	LFB waiting for all ST write data to arrive on XMD bus
<6>	WTSTRSP	RO	0	0	LFB waiting for ST RSC/RSD to be issued on RSP (invalidates)
<5>	WTSTRSC	RO	0	0	LFB waiting for ST RSC-only to be issued on RSP (no-invalidates)
<4>	WTVTM	RO	0	0	LFB waiting for victim read L2 DS-RD completion
<3>	WTMFL	RO	0	0	LFB waiting for memory fill completion to MRB
<2>	PRBRTY	RO	0	0	Probe retry detected – waiting for probe completion.
<1>	WTPRB	RO	0	0	LFB waiting for probe
<0>	VLD	RO	0	0	LFB valid

Level 2 Cache In-Flight Address Buffer Debug 2 Register L2C_LFB2

Contains L2C LFB contents tag/index. See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:27>	—	RAZ	0x0	0x0	Reserved
<26:7>	LFB_TAG	RO	0x0	0x0	LFB TAG[33:14]
<6:0>	LFB_IDX	RO	0x0	0x0	LFB IDX[13:7]

Level 2 Cache In-Flight Address Buffer Debug 3 Register L2C_LFB3

Contains L2C LFB high-water mark. See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	0x0	0x0	Reserved
<4>	STPARTDIS	R/W	0	0	STP/C performance enhancement disable. When clear, all STP/C (store partial) commands take two cycles to complete (power-on default). When set, all STP/C commands take four cycles to complete. **NOTE:** Cavium recommends you always keep this bit = 0.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<3>	—	RAZ	0	0	Reserved
<2:0>	LFB_HMW	R/W	0x7	0x7	LFB high-water mark. Determines the number of LFB entries in use before backpressure is asserted: 0 = 1 LFB entry is available, ..., 7 = 8 LFB entries are available

Level 2 Cache DuTag Register L2C_DUT

L2C duplicate-tag state register. See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	0	0	Reserved
<31>	DTENA	R/W	0	0	DuTag diagnostic read enable. When set to 1, all LDD (D-stream load) commands issued from the diagnostic core (L2C_ DBG[PPNUM]) capture the DuTag state (V|L1TAG) of the corespecified in the LDD address <26> into L2C_DUT. The DuTag set# to capture is extracted from the LDD address<25:20>. The diagnostic core would issue the LDD then read the L2C_DUT register (one at a time). This LDD “force-hit” does not alter the current L2 tag state or the DuTag state. The fill data is returned directly from the L2 data store (regardless of hit/miss) when an LDD instruction is issued from the core determined by the L2C_DBG[PPNUM] field. The selected L2 set# is determined by L2C_DBG[SET], and the index is determined from the address[13:7] associated with the LDD instruction. This “force-hit” does not alter the current L2 tag state or the DuTag state. **NOTE:** In order for the diagnostic core to generate an LDD command to the L2C, it must first force an L1 Dcache flush. **NOTE:** The DuTag SIZE is 16KB, where each DuTag is organized as 2 × 64-way entries. The LDD address[7] determines which one (of two) internal 64-ways to select.
<30>	—	RAZ	0	0	Reserved
<29>	DT_VLD	RO	0	0	Duplicate L1 tag valid bit, latched in for previous LDD instruction sourced by diagnostic core.
<28:0>	DT_TAG	RO	0	0	Duplicate L1 Tag[35:7] latched in for previous LDD instruction sourced by diagnostic core.

Notes:

- When using the L2T, L2D, or FINV debug probe feature, an LDD instruction issued by the diagnostic core does not update the DuTags.
- L2T, L2D, and FINV must be mutually exclusive (only one enabled at a time).
- Force Invalidate is intended as a means for software to invalidate the L2 cache while also writing back dirty data to memory to maintain coherency. (A side effect of FINV is that an LDD miss fill is launched, which fills data into the L2 DS).

Level 2 Cache Lock Down Base Register L2C_LCKBASE

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:31>	—	RAZ	0x0	0x0	Reserved
<30:4>	LCK_BASE	RO	0x0	0x0	Base memory block address[33:7]. Specifies the starting address of the lockdown region.
<3:1>	—	RAZ	0x0	0x0	Reserved
<0>	LCK_ENA	R/W	0	0	L2 cache lock enable. When set to 1, all LDI (I-stream Load) or LDD (D-stream load) commands issued from the core that fall within a predefined lockdown address range (specified by: [lck_base:lck_base+lck_offset]) are LOCKED in the L2 cache. The LOCKED state is denoted using an explicit L2 Tag bit (L = 1). The LOCKED state is encoded in the L2 Tag as (V=0/D=1/U=1). If the LOCK request L2-Hits, then data is returned from the L2 and the hit set is updated to the LOCKED state. If the LOCK request L2-Misses, a replacement set is chosen. If the replacement set contains a dirty-victim, it is written back to memory. If the LOCK request L2-Misses, a replacement set is chosen (from the available sets (UMSKx). If the replacement set contains a dirty-victim it is written back to memory. Memory read data is then written into the replacement set, and the replacement SET is updated to the LOCKED state(L=1). **NOTE:** SETs that contain LOCKED addresses are excluded from the replacement set selection algorithm. **NOTE:** The LDD allocates the DuTag as normal.

Notes:

- Software restriction #1: Software must manage the L2 data store lockdown space such that at least one set per cache line remains in the unlocked (normal) state to allow general caching operations. If software violates this restriction, a status bit is set (LCK_ERR) and an interrupt is posted. This limits the total lockdown space to 7/8ths of the total L2 data store = 896KB.
- IOB-initiated LDI commands are ignored (only core-initiated LDI/LDD commands are considered for lockdown).
- To unlock a locked cache line, software can use the FLUSH-INVAL CSR mechanism (see L2C_DBG[FINV]).
- LCK_ENA must only be activated when debug modes are disabled (L2C_DBG[L2T], L2C_DBG[L2D], L2C_DBG[FINV]).

Level 2 Cache Lock Down OFFSET Register
L2C_LCKOFF

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:10>	—	RAZ	0x0	0x0	Reserved
<9:0>	LCK_OFFSET	R/W	0x0	0x0	Lock down block offset. Used in determining the ending block address of the lockdown region: End lockdown block address<33:7> = L2C_LCKBASE[LCK_BASE] + LCK_OFFSET

Notes:

- The generation of the end lockdown block address will wrap.
- The minimum granularity for lockdown is 1 cache line (= 128-byte block).

Level 2 Cache Set Partitioning Registers (Core Set 0)
L2C_SPAR0

When a bit is set in the UMSK*n* register, a memory command issued from core *n* does not select that set for replacement. There should always be at least 1 bit clear in each UMSK*n* register for proper L2 cache operation.

NOTE: When L2C FUSE[136] is blown (CRIP_64K), then SETS 7–4 are SET in all UMSK*n* registers. When L2C FUSE[137] is blown (CRIP_32K), then SETS 7–2 are SET in all UMSK*n* registers.

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	0x0	0x0	Reserved
<15:8>	UMSK1	R/W	0x0	0x0	Core 1 L2 “DO NOT USE” set partition mask
<7:0>	UMSK0	R/W	0x0	0x0	Core 0 L2 “DO NOT USE” set partition mask

Level 2 Cache Set Partitioning Register (IOB)
L2C_SPAR4

When a bit is set in the UMSK*n* register, a memory command issued from core *n* does not select that set for replacement. There should always be at least 1 bit clear in each UMSK*n* register for proper L2 cache operation.

NOTE: When L2C FUSE[136] is blown (CRIP_64K), then SETS 7–4 are SET in all UMSK*n* registers. When L2C FUSE[137] is blown (CRIP_32K), then SETS 7–2 are SET in all UMSK*n* registers.

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	0x0	0x0	Reserved
<7:0>	UMSKIOB	R/W	0x0	0x0	IOB L2 “DO NOT USE” set partition mask

Level 2 Cache Performance Counter Control Register L2C_PFCTL

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	0x0	0x0	Reserved
<35>	CNT3RDCLR	R/W	0	0	Performance counter 3 read clear. When set, all CSR reads of the L2C_PFC3 register will autoclear the counter. This allows SW to maintain “cumulative” counters in SW. **NOTE:** If the CSR read occurs in the same cycle as the “event” to be counted, the counter will properly reflect the event.
<34>	CNT2RDCLR	R/W	0	0	Performance counter 2 read clear. When set, all CSR reads of the L2C_PFC2 register will autoclear the counter. This allows SW to maintain “cumulative” counters in SW. **NOTE:** If the CSR read occurs in the same cycle as the “event” to be counted, the counter will properly reflect the event.
<33>	CNT1RDCLR	R/W	0	0	Performance counter 1 read clear. When set, all CSR reads of the L2C_PFC1 register will autoclear the counter. This allows SW to maintain “cumulative” counters in SW. **NOTE:** If the CSR read occurs in the same cycle as the “event” to be counted, the counter will properly reflect the event.
<32>	CNT0RDCLR	R/W	0	0	Performance counter 0 read clear. When set, all CSR reads of the L2C_PFC0 register will autoclear the counter. This allows SW to maintain “cumulative” counters in SW. **NOTE:** If the CSR read occurs in the same cycle as the “event” to be counted, the counter will properly reflect the event.
<31>	CNT3ENA	R/W	0	0	Performance counter 3 enable. When the CSR write occurs, if this bit is set, the performance counter is cleared.
<30>	CNT3CLR	R/W	0	0	Performance counter 3 clear. When the CSR write occurs, if this bit is set, the performance counter is cleared. Otherwise, it will resume counting from its current value.
<29:24>	CNT3SEL	R/W	0x0	0x0	Performance counter 3 event selector. See the list of selectable events to count in Table 2–13.
<23>	CNT2ENA	R/W	0	0	Performance counter 2 enable. When the CSR write occur, if this bit is set, the performance counter is cleared.
<22>	CNT2CLR	R/W	0	0	Performance counter 2 clear. When the CSR write occurs, if this bit is set, the performance counter is cleared. Otherwise, it will resume counting from its current value.
<21:16>	CNT2SEL	R/W	0x0	0x0	Performance counter 2 event selector. See the list of selectable events to count in Table 2–13.
<15>	CNT1ENA	R/W	0	0	Performance counter 1 enable. When the CSR write occur, if this bit is set, the performance counter is cleared.
<14>	CNT1CLR	R/W	0	0	Performance counter 1 clear. When the CSR write occurs, if this bit is set, the performance counter is cleared. Otherwise, it will resume counting from its current value.
<13:8>	CNT1SEL	R/W	0x0	0x0	Performance counter 1 event selector. See the list of selectable events to count in Table 2–13.
<7>	CNT0ENA	R/W	0	0	Performance counter 0 enable. When the CSR write occur, if this bit is set, the performance counter is cleared.
<6>	CNT0CLR	R/W	0	0	Performance counter 0 clear. When the CSR write occurs, if this bit is set, the performance counter is cleared. Otherwise, it will resume counting from its current value.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<5:0>	CNT0SEL	R/W	0x0	0x0	Performance counter 0 event selector. See the list of selectable events to count in Table 2–13.

Note:

- There are four 36-bit performance-counter registers that can simultaneously count events. Each counter's event, described in Table 2–13, is programmably selected via the corresponding CNT*n*SEL field:

Table 2–13 List of Selectable Events to Count

CNT*n*SEL[5:0]	Event
00 0000	Cycles
00 0001	L2 instruction miss
00 0010	L2 instruction hit
00 0011	L2 data miss
00 0100	L2 data hit
00 0101	L2 miss (I/D)
00 0110	L2 hit (I/D)
00 0111	L2 victim buffer hit (retry probe)
00 1000	LFB-NQ index conflict
00 1001	L2 tag probe (issued - could be VB-retried)
00 1010	L2 tag update (completed). **Note**: Some CMD types do not update.
00 1011	L2 tag probe completed (beyond VB-RTY window)
00 1100	L2 tag dirty victim
00 1101	L2 data store NOP
00 1110	L2 data store READ
00 1111	L2 data store WRITE
01 0000	Memory fill data valid
01 0001	Memory write request
01 0010	Memory read request
01 0011	Memory write data valid
01 0100	XMC NOP
01 0101	XMC LDT
01 0110	XMC LDI
01 0111	XMC LDD
01 1000	XMC STF
01 1001	XMC STT
01 1010	XMC STP
01 1011	XMC STC
01 1100	XMC DWB
01 1101	XMC PL2
01 1110	XMC PSL1
01 1111	XMC IOBLD
10 0000	XMC IOBST
10 0001	XMC IOBDMA
10 0010	XMC IOBRSP
10 0011	XMD bus valid (all)

Table 2–13 List of Selectable Events to Count (Continued)

CNT*n*SEL[5:0]	Event
10 0100	XMD bus valid (DST=L2C) memory data
10 0101	XMD Bus valid (DST=IOB) REFL data
10 0110	XMD Bus valid (DST=PP) IOBRSP data
10 0111	RSC NOP
10 1000	RSC STDN
10 1001	RSC FILL
10 1010	RSC REFL
10 1011	RSC STIN
10 1100	RSC SCIN
10 1101	RSC SCFL
10 1110	RSC SCDN
10 1111	RSD data valid
11 0000	RSD data valid (FILL)
11 0001	RSD data valid (STRSP)
11 0010	RSD data valid (REFL)
11 0011	LRF-REQ (LFB-NQ)
11 0100	DT RD-ALLOC (LDD/PSL1 commands)
11 0101	DT WR-INVA (ST* commands)

Level 2 Cache Performance Counter Registers L2C_PFC(0..3)

Level-2 cache performance counters. See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	0x0	0x0	Reserved
<35:0>	PFCNT*n*	RO	0x0	0x0	Performance counter 0..3

Level 2 Cache Data Store QUAD0 BIST Status Register L2D_BST0

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:35>	—	RAZ	0x0	0x0	Reserved
<34>	FTL	RO	0	0	L2C data store fatal defect (across all QUADs): two or more columns were detected bad across all QUADs[0-3]. Please refer to individual quad failures for bad column = 0x7E to determine which QUAD was in error.
<33:0>	Q0STAT	RO	0x0	0x0	BIST results for QUAD0: Failure #1 Status [16:14] Unused [13:7] bad high column [6:0] bad low column Failure #2 Status [33:31] Unused [30:24] bad high column [23:17] bad low column **NOTE:** For bad high/low column reporting: 0x7F: No failure 0x7E: Fatal Defect: two or more bad columns **NOTE:** 0–0x45: Bad column

Level 2 Cache Data Store QUAD1 BIST Status Register L2D_BST1

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:34>	RSVD	RAZ	0x0	0x0	Reserved
<33:0>	Q1STAT	RO	0x0	0x0	BIST results for QUAD1: Failure #1 Status [16:14] Unused [13:7] bad high column [6:0] bad low column Failure #2 Status [33:31] Unused [30:24] bad high column [23:17] bad low column **NOTE:** For bad high/low column reporting: 0x7F: No failure 0x7E: Fatal Defect: two or more bad columns **NOTE:** 0–0x45: Bad column

Level 2 Cache Data Store QUAD2 BIST Status Register L2D_BST2

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:34>	RSVD	RAZ	0x0	0x0	Reserved
<33:0>	Q2STAT	RO	00x	0x0	BIST results for QUAD2: Failure #1 Status [16:14] Unused [13:7] bad high column [6:0] bad low column Failure #2 Status [33:31] Unused [30:24] bad high column [23:17] bad low column **NOTE:** For bad high/low column reporting: 0x7F: No failure 0x7E: Fatal Defect: two or more bad columns **NOTE:** 0–0x45: Bad column

Level 2 Cache Data Store QUAD3 BIST Status Register L2D_BST3

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:34>	RSVD	RAZ	0	0	Reserved
<33:0>	Q3STAT	RO	0	0	BIST results for QUAD3: Failure #1 Status [16:14] Unused [13:7] bad high column [6:0] bad low column Failure #2 Status [33:31] Unused [30:24] bad high column [23:17] bad low column **NOTE:** For bad high/low column reporting: 0x7F: No failure 0x7E: Fatal Defect: two or more bad columns **NOTE:** 0–0x45: Bad column

Level 2 Cache Data Store QUAD0 Fuse Register L2D_FUS0

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:34>	—	RAZ	0	0	Reserved
<33:0>	Q0FUS	RO	0	0	Fuse register for QUAD0. This is purely for debug and not needed in the general manufacturing flow. Note that the fuses are complementary (Assigning a fuse to 1 will read as a 0). This means the case where no fuses are blown results in these CSRs showing all 1s. Failure #1 fuse mapping [16:14] Unused [13:7] bad high column [6:0] bad low column Failure #2 fuse mapping [33:31] Unused [30:24] bad high column [23:17] bad low column

Level 2 Cache Data Store QUAD1 Fuse Register L2D_FUS1

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:34>	—	RAZ	0	0	Reserved
<33:0>	Q1FUS	RO	0	0	Fuse register for QUAD1. This is purely for debug and not needed in the general manufacturing flow. Note that the fuses are complementary (Assigning a fuse to 1 will read as a 0). This means the case where no fuses are blown results in these CSRs showing all 1s. Failure #1 fuse mapping [16:14] Unused [13:7] bad high column [6:0] bad low column Failure #2 fuse mapping [33:31] Unused [30:24] bad high column [23:17] bad low column

Level 2 Cache Data Store QUAD2 Fuse Register L2D_FUS2

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:34>	—	RAZ	0	0	Reserved
<33:0>	Q2FUS	RO	0	0	Fuse register for QUAD2. This is purely for debug and not needed in the general manufacturing flow. Note that the fuses are complementary (Assigning a fuse to 1 will read as a 0). This means the case where no fuses are blown results in these CSRs showing all 1s. Failure #1 fuse mapping [16:14] Unused [13:7] bad high column [6:0] bad low column Failure #2 fuse mapping [33:31] Unused [30:24] bad high column [23:17] bad low column

Level 2 Cache Data Store QUAD3 Fuse Register L2D_FUS3

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:40>	—	RAZ	0x0	0x0	Reserved
<39:37>	EMA_CTL	RO	0x0	0x0	L2 data-store EMA control. These bits are used to observe the EMA[1:0] inputs for the L2 data-store RAMs, which are controlled either by FUSES[142:140] or by MIO_FUSE_EMA[EMA] CSR. From power-up (DC_OK), EMA_CTL is driven by FUSE[142:140]. However after the first CSR write to MIO_FUSE_EMA[EMA] bits, the EMA_CTL are driven by MIO_FUSE_EMA[EMA] (until DC_OK).
<36>	FUS_SPARE	RO	0	0	This is purely for debug and not needed in the general manufacturing flow. If the FUSE is not blown, then this bit should read as 0. If the FUSE is blown, then this bit should read as 1.
<35>	CRIP_32K	RO	0	0	This is purely for debug and not needed in the general manufacturing flow. If the FUSE is not blown, then this bit should read as 0. If the FUSE is blown, then this bit should read as 1.
<34>	CRIP_64K	RO	0	0	This is purely for debug and not needed in the general manufacturing flow. If the FUSE is not blown, then this bit should read as 0. If the FUSE is blown, then this bit should read as 1.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<33:0>	Q3FUS	RO	0x0	0x0	Fuse register for QUAD3. This is purely for debug and not needed in the general manufacturing flow. Note that the fuses are complementary (Assigning a fuse to 1 will read as a 0). This means the case where no fuses are blown results in these CSRs showing all 1s. Failure #1 fuse mapping [16:14] Unused [13:7] bad high column [6:0] bad low column Failure #2 fuse mapping [33:31] Unused [30:24] bad high column [23:17] bad low column

Level 2 Cache BIST 0 Control and Status Register L2C_BST0

This register specifies the RSL base addresses for the block. See Table 2–12 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:24>	—	RAZ	0x0	0x0	Reserved
<23>	DTBNK	RO	0	0	DuTag Bank number. When DT = 1 (BAD), this field provides additional information about which DuTag Bank (0/1) failed.
<22:19>	WLB_MSK	RO	0x0	0x0	BIST results for WLB-MSK RAM [DP0-3] 0 = GOOD (or BIST in progress/never run) 1 = BAD
<18:16>	—	RAZ	0x0	0x0	Reserved
<15:6>	DTCNT	RO	0x0	0x0	DuTag BiST Counter (used to help isolate the failure) [9]: i (0=FORWARD/1=REVERSE pass) [8:7]: j (Pattern# 1 of 4) [6:1]: k (DT Index 1 of 64) [0]: l (DT# 1 of 2 DTs)
<5>	DT	RO	0	0	BIST results for DuTAG RAMs 0 = GOOD (or BIST in progress/never run) 1 = BAD
<4>	STIN_MSK	RO	0	0	BIST results for STIN-MSK RAM 0 = GOOD (or BIST in progress/never run) 1 = BAD
<3:0>	WLB_DAT	RO	0x0	0x0	BIST results for WLB-DAT RAM [DP0-3] 0 = GOOD (or BIST in progress/never run) 1 = BAD

Level 2 Cache BIST 1 Control and Status Register L2C_BST1

See Table 2–12 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	0	0	Reserved
<15:12>	VWDF	RO	0	0	BIST results for VWDF RAMs 0 = GOOD (or BIST in progress/never run) 1 = BAD
<11:10>	LRF	RO	0	0	BIST results for LRF RAMs (PLC+ILC) 0 = GOOD (or BIST in progress/never run) 1 = BAD
<9>	VAB_VWCF	RO	0	0	BIST results for VAB VWCF_MEM 0 = GOOD (or BIST in progress/never run) 1 = BAD

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<8:0>	L2T	RO	0	0	BIST results for L2T (USE+8SET RAMs) 0 = GOOD (or BIST in progress/never run) 1 = BAD

Level 2 Cache BIST 2 Control and Status Register L2C_BST2

See Table 2–12 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	0	0	Reserved
<15:12>	MRB	RO	0	0	BIST results for MRB RAMs 0 = GOOD (or BIST in progress/never run) 1 = BAD
<11:8>	RMDF	RO	0	0	BIST results for RMDF RAMs 0 = GOOD (or BIST in progress/never run) 1 = BAD
<7:4>	—	RAZ	0	0	Reserved
<3>	IPCBST	RO	0	0	BIST results for RFB IPC RAM 1 = BAD
<2>	—	RAZ	0	0	Reserved
<1>	XRDMSK	RO	0	0	BIST results for RFB XRD-MSK RAM 0 = GOOD (or BIST in progress/never run) 1 = BAD
<0>	XRDDAT	RO	0	0	BIST results for RFB XRD-DAT RAM 0 = GOOD (or BIST in progress/never run) 1 = BAD

2.5 LMC Registers

The LMC registers are listed in Table 2–14.

Table 2–14 LMC Registers

Register	Address	CSR Type[1]	Detailed Description
LMC_MEM_CFG0	0x0001180088000000	RSL	See page 106
LMC_MEM_CFG1	0x0001180088000008	RSL	See page 109
LMC_CTL	0x0001180088000010	RSL	See page 111
LMC_DDR2_CTL	0x0001180088000018	RSL	See page 113
LMC_FADR	0x0001180088000020	RSL	See page 115
LMC_COMP_CTL	0x0001180088000028	RSL	See page 115
LMC_WODT_CTL	0x0001180088000030	RSL	See page 116
LMC_ECC_SYND	0x0001180088000038	RSL	See page 117
LMC_IFB_CNT_LO	0x0001180088000048	RSL	See page 117
LMC_IFB_CNT_HI	0x0001180088000050	RSL	See page 118
LMC_OPS_CNT_LO	0x0001180088000058	RSL	See page 118
LMC_OPS_CNT_HI	0x0001180088000060	RSL	See page 118
LMC_DCLK_CNT_LO	0x0001180088000068	RSL	See page 118
LMC_DCLK_CNT_HI	0x0001180088000070	RSL	See page 119
LMC_RODT_CTL	0x0001180088000078	RSL	See page 119
LMC_DELAY_CFG	0x0001180088000088	RSL	See page 119
LMC_CTL1	0x0001180088000090	RSL	See page 120
LMC_DUAL_MEM_CONFIG	0x0001180088000098	RSL	See page 121
LMC_RODT_COMP_CTL	0x00011800880000A0	RSL	See page 123
LMC_PLL_CTL	0x00011800880000A8	RSL	See page 123
LMC_PLL_STATUS	0x00011800880000B0	RSL	See page 124
LMC_BIST_CTL	0x00011800880000F0	RSL	See page 124
LMC_BIST_RESULT	0x00011800880000F8	RSL	See page 124

1. RSL-type registers are accessed indirectly across the I/O Bus.

LMC Memory Configuration Register0 LMC_MEM_CFG0

This register controls certain parameters required for memory configuration. See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31>	RESET	RAZ	—	—	Reset one-shot pulse for refresh counter, and LMC_OPS_CNT_*, LMC_IFB_CNT_*, and LMC_DCLK_CNT_* CSRs. Software should write this to a 1, then rewrite it to a 0 to cause the reset.
<30>	SILO_QC	R/W	0	—	Adds a quarter cycle granularity to generate dqs pulse generation for silo. Combination of SILO_HC and SILO_QC gives the ability to position the read enable with quarter cycle resolution. This is applied on all the bytes uniformly.
<29>	BUNK_ENA	R/W	0	0	Bunk or rank enable. Used with dual-rank DIMMs. Enables the drive of the CS_N[1:0] pins based on the (PBANK_LSB–1) address bit. Write 0 for single-rank DIMMs.
<28:25>	DED_ERR[3:0]	R/W1C	0x0	0x0	Double-bit error detected on read data. In 32-bit mode, ECC is calculated on four cycles' worth of data (c represents cycle and p represents phase). [25] corresponds to {DQ[31:0], c 0, p 1, DQ[31:0], c 0, p 0} [26] corresponds to {DQ[31:0], c 1, p 1, DQ[31:0], c 1, p 0} [27] corresponds to {DQ[31:0], c 2, p 1, DQ[31:0], c 2, p 0} [28] corresponds to {DQ[31:0], c 3, p 1, DQ[31:0], c 3, p 0} In 16-bit mode, ECC is calculated on eight cycles' worth of data (c represents cycle and p represents phase). [25] corresponds to {DQ[15:0], c 1, p 1, DQ[15:0], c 1, p 0, DQ[15:0], c 0, p 1, DQ[15:0], c 0, p 0} [26] corresponds to{ DQ[15:0], c 3, p 1, DQ[15:0], c 3, p 0, DQ[15:0], c 2, p 1, DQ[15:0], c 2, p 0} [27] corresponds to{ DQ[15:0], c 5, p 1, DQ[15:0], c 5, p 0, DQ[15:0], c 4, p 1, DQ[15:0], c 4, p 0} [28] corresponds to {DQ[15:0], c 7, p 1, DQ[15:0], c 7, p 0, DQ[15:0], c 6, p 1, DQ[15:0], c 6, p 0} Writing 1 clears the corresponding error bit.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<24:21>	SEC_ERR	R/W1C	0x0	0x0	Single-bit error (corrected) on read data. In 32-bit mode, ECC is calculated on four cycles' worth of data (c represents cycle and p represents phase). [21] corresponds to {DQ[31:0], c 0, p 1, DQ[31:0], c 0, p 0} [22] corresponds to {DQ[31:0], c 1, p 1, DQ[31:0], c 1, p 0} [23] corresponds to {DQ[31:0], c 2, p 1, DQ[31:0], c 2, p 0} [24] corresponds to {DQ[31:0], c 3, p 1, DQ[31:0], c 3, p 0} In 16-bit mode, ECC is calculated on eight cycles' worth of data (c represents cycle and p represents phase). [21] corresponds to {DQ[15:0], c 1, p 1, DQ[15:0], c 1, p 0, DQ[15:0], c 0, p 1, DQ[15:0], c 0, p 0} [22] corresponds to{ DQ[15:0], c 3, p 1, DQ[15:0], c 3, p 0, DQ[15:0], c 2, p 1, DQ[15:0], c 2, p 0} [23] corresponds to{ DQ[15:0], c 5, p 1, DQ[15:0], c 5, p 0, DQ[15:0], c 4, p 1, DQ[15:0], c 4, p 0} [24] corresponds to {DQ[15:0], c 7, p 1, DQ[15:0], c 7, p 0, DQ[15:0], c 6, p 1, DQ[15:0], c 6, p 0} Writing 1 clears the corresponding error bit.
<20>	INTR_DED_ENA	R/W	0	0	ECC double error detect (DED) interrupt enable bit. When set, allows interrupts to be reported on double bit (uncorrectable) errors.
<19>	INTR_SEC_ENA	R/W	0	1	ECC single error correct (SEC) interrupt enable bit. When set, allows interrupts to be reported on single bit (correctable) errors.
<18:15>	TCL	R/W	0x3	—	Not used. Set to 0x0.
<14:9>	REF_INT	R/W	0x1	0x2	Refresh interval, represented in number of 512 DCLK increments. 000000 = Reserved 000001 = 1 × 512 = 512 DCLK cycles ... 111111 = 63 × 512 = 32256 DCLK cycles
<8:5>	PBANK_LSB	R/W	0x5	—	Physical Bank address select. Refer to Figure 2–10.
<4:2>	ROW_LSB	R/W	0x3	—	Encoding used to determine which memory address bit position represents the low order DDR ROW address: Value / Address bit is LSB: 000 [14]; 001 [15]; 010 [16]; 011 [17]; 100 [18]; 101 reserved; 110 [12]; 111 [13] Refer to Figure 2–10.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<1>	ECC_ENA	R/W	0	0	ECC enable. Enables the 8-bit ECC check/correct logic. Should be 1 when used with DIMMs with ECC. 0, otherwise. • When this mode is turned on, writes contain the ECC code generated for the 64 bits of data that will written in the memory and then later on reads, are used to check for single-bit errors (which are autocorrected) and double-bit errors (which are reported). • When not turned on, writes will not contain ECC, and the ECC bits are driven to 0. Refer to SEC_ERR, DED_ERR, and the LMC_FADR, and LMC_ECC_SYND registers for diagnostics information when there is an error.
<0>	INIT_START	R/W	0	0	Start initialization. Starts the DDR memory initialization sequence required prior to using the memory.

LMC Memory Configuration Register1 LMC_MEM_CFG1

This register controls the external memory configuration timing parameters. See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31>	COMP_BYPASS	R/W	0	0	Compensation bypass.
<30:28>	TRRD	R/W	0x2	0x2	T_{RRD} cycles. Specifies ACT-ACT timing parameter for different banks (represented in T_{CYC} cycles = 1DCLK cycle). For DDR2, typical = 7.5ns 000 = Reserved 001 = 1 T_{CYC} 010 = 2 T_{CYC} 011 = 3 T_{CYC} 100 = 4 T_{CYC} 101 = 5 T_{CYC} 110 = 6 T_{CYC} 111 = 7 T_{CYC}
<27:25>	CASLAT	R/W	0x4	0x4	CAS latency encoding, which is loaded into each DDR SDRAM device (MRS[6:4]) upon power-up (INIT_START=1). Represented in T_{CYC} cycles = 1 DCLK cycle. 000 = Reserved 001 = Reserved 010 = 2.0 T_{CYC} 011 = 3.0 T_{CYC} 100 = 4.0 T_{CYC} 101 = 5.0 T_{CYC} 110 = 6.0 T_{CYC} 111 = Reserved **Example**: The parameters TSKW, SILO_HC, and SILO_QC can account for ¼ cycle granularity in board/etch delays.
<24:22>	TMRD	R/W	0x2	0x2	T_{MRD} cycles = RNDUP[T_{RFC}(ns) / T_{CYC}(ns)]. Represented in T_{CYC} cycles = 1 DCLK). Typically, it is 2 × T_{CYC}. 000 = Reserved 001 = 1 010 = 2 011 = 3 100 = 4 101-111 = Reserved
<21:17>	TRFC	R/W	0x6	0x7	¼ T_{RFC} cycles = RNDUP[T_{RFC}(ns) / 4 × T_{CYC}(ns)]. Represented in T_{CYC} cycles = 1 DCLK cycle. For example, for **2Gb, 667MHz** parts, typically T_{RFC} = 195 ns: RNDUP[195/(4×3)] = 17 = 0x11 = TRFC 0x0 and 0x1 are reserved.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<16:13>	TRP	R/W	0x5	0x4	T_{RP} cycles = RNDUP[T_{RP}(ns) / T_{CYC}(ns)]. Represented in T_{CYC} cycles = 1 DCLK cycle. Typical = 15ns (66MHz=1, 167MHz=3, 400MHz=6) 0000 = Reserved 0001 = 1 ... 1001 = 9 1010–1111 = Reserved When using parts with 8 banks (LMC_DDR2_CTL[BANK8] = 1), load T_{RP} cycles + 1 into this register.
<12:9>	TWTR	R/W	0x2	0x2	Last write data to read command time. T_{WTR} cycles = RNDUP[T_{WTR}(ns) / T_{CYC}(ns)]. Represented in T_{CYC} cycles = 1 DCLK cycle. Typical = 15ns (66MHz=1, 167MHz=3, 400MHz=6) 0000 = Reserved 0001 = 1 ... 0111 = 7 1000–1111 = Reserved
<8:5>	TRCD	R/W	0x4	0x4	T_{RCD} cycles = RNDUP[T_{RAS}(ns) / T_{CYC}(ns)]. Represented in T_{CYC} cycles = 1 DCLK cycle) Typical = 15ns (66MHz=1,167MHz=3,400MHz=6 for TYP) 0000 = Reserved 0001 = 2 (2 is the smallest value allowed) 0010 = 2 ... 1001 = 9 1010–1111 = Reserved
<4:0>	TRAS	R/W	0xC	0xC	T_{RAS} cycles = RNDUP[T_{RAS}(ns) / T_{CYC}(ns)]. Represented in T_{CYC} cycles = 1 DCLK). 00000-0001 = Reserved 00010 = 2 ... 11111 = 31

LMC Control Register
LMC_CTL

This register provides various control fields used by the LMC. See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:28>	DDR_NCTL	RO	—	—	DDR NMOS control from compensation circuit. The encoded value on this adjusts the drive strength of the DDR DQ pulldown resistors.
<27:24>	DDR_PCTL	RO	—	—	DDR PMOS control from compensation circuit. The encoded value on this adjusts the drive strength of the DDR DQ pullup resistors.
<23>	—	R/W	0	0	Must be 0.

<22>	XOR_BANK	R/W	0	1	If (XOR_BANK = 1), then
					bank[n:0] = address[n+7:7] ⊕ address[n+7+5:7+5]
					else
					bank[n:0] = address[n+7:7]
					where n=1 for a four-bank part and n=2 for an eight-bank part

<21:18>	MAX_WRITE_ BATCH	R/W	0x8	0x8	Maximum number of consecutive writes to service before allowing reads to interrupt.
<17>	—	R/W	0	0	Must be 0.
<16>	PLL_BYPASS	R/W	0	0	PLL bypass.
<15>	RDIMM_ENA	R/W	0	0	Registered DIMM enable. When this bit is set, it allows the use of JEDEC Registered DIMMs, which require write data to be registered in the controller.
<14>	R2R_SLOT	R/W	0	0	Read-to-read slot enable. When set, all read-to-read transactions slot an additional one-cycle data bus bubble to avoid DQ/DQS bus contention. **NOTE:** This bit should not be needed in normal operation. It is provided in case the built-in DIMM and BUNK crossing logic (which autodetects and slots read-to-reads to the same DIMM/BUNK) does not work.
<13>	INORDER_MWF	RAZ	0	0	Reads as 0.
<12>	INORDER_MRF	R/W	0	0	Always clear to 0.
<11>	DRESET	R/W	1	0	DCLK domain reset. The DCLK domain is reset whenever this bit is set, or whenever the ECLK domain is reset.
<10>	MODE32b	R/W	1	1	32-bit data-path mode. 1 = 32 DQ pins are used, 0 = 16 DQ pins are used.
<9>	FPRCH2	R/W	0	1	Front porch enable. When this bit is set, the turn-off time for the DDR_DQ/DQS drivers is one DCLK cycle earlier. Typically, this should be set to 1.
<8>	BPRCH	R/W	0	—	Back porch enable. When this bit is set, the turn-on time for the DDR_DQ/DQS drivers is delayed an additional DCLK cycle. This should be set to 1 whenever both SILO_HC and SILO_QC are set.
<7:6>	SIL_LAT	R/W	0x1	0x1	Silo latency. On reads, this field determines how many additional DCLK cycles to wait (on top of TCL+1+TSKW) before pulling data out of the pad silos. 00 = illegal 01 = 1 DCLK cycle 10 = 2 DCLK cycles 11 = illegal This should always be set to 1.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<5:4>	TSKW	R/W	0x0	0x1	Time of skew. Specifies the total board delay on DQ/DQS during reads (used to determine the R→W spacing to avoid DQ/DQS bus conflicts). Enter the largest per-byte board delay. 00 = 0 DCLK cycles 01 = 1 DCLK cycle 10 = 2 DCLK cycles 11 = 3 DCLK cycles
<3:2>	QS_DIC	R/W	0x2	0x2	DDR2 termination-resistor setting. A non-zero value in this register enables the on-die termination (ODT). This field is loaded into the RTT section of the EMRS (bits [A6,A2] for DDR termination of the memory's DQ/DQS/DM pads. 00 = ODT disabled 01 = 75Ω termination 10 = 150Ω termination 11 = 50Ω termination On write operations, CN50XX, by default, drives the ODT pins based on what the masks (LMC_WODT_CTL) are programmed to. LMC_DDR2_CTL[ODT_ENA] enables CN50XX to drive ODT pins for read operations. LMC_RODT_CTL must be programmed based on the system's needs for ODT.
<1:0>	DIC	R/W	0x0	0x0	Drive strength control. DIC[0] is loaded into the EMRS [A1] bit during initialization. 0 = Normal 1 = Reduced. DIC[1] is loaded into the EMRS DSQN enable/disable field (bit[A10]). 0 = DSQN enable 1 = DSQN disable (if high impedance)

LMC DDR2 and DLL Control Register
LMC_DDR2_CTL

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	0	0	Reserved
<31>	BANK8	R/W	0	—	DDR2 eight-bank: 1 = eight banks (BA is 3 bits), 0 = four banks (BA is 2 bits)
<30>	BURST8	R/W	0	1	Eight-burst mode: 1 = DDR transfer burst is 8, 0 = DDR transfer burst is 4. BURST8 should be set when DDR2T = 1 to minimize the command bandwidth loss.
<29:27>	ADDLAT	R/W	0x0	0x0	Additional latency for posted CAS. When posted CAS is enabled, this configures the additional latency. This should be set between 1 and LMC_MEM_CFG1[TRCD] – 2. Note the implication that posted CAS should not be used when T_{RCD} = 2.
<26>	POCAS	R/W	0	0	Enable the posted CAS feature of DDR2.
<25>	BWCNT	R/W	0	0	Clear bus utilization counters. Clears the LMC_OPS_CNT_*, LMC_IFB_CNT_*, and LMC_DCLK_CNT_* registers. Software should first write this field to a 1, then write this field to a 0 to clear the CSRs.
<24:22>	TWR	R/W	0x3	0x1	DDR write recovery time (T_{WR}). The last write burst to predelay. This is not a direct encoding of the values are specified in the DDR2 specification. The value shown is RNDUP(T_{WR}(ns) / T_{CYC}(ns)). Typical is 15 ns. 000 = Reserved 100 = 5 001 = 2 101 = 6 010 = 3 110 = 7 011 = 4 111 = 8
<21>	SILO_HC	R/W	1	—	Silo half cycle. Delays the read-sample window by a half cycle.
<20:17>	DDR_EOF	R/W	0x0	0x0	Early fill counter initialization. L2C needs to know a few cycle before a fill completes so it can get its control pipe started (for better overall performance). This counter contains an initialization value that is a function of ECLK/DCLK ratio, to account for the asynchronous boundary between L2 cache and the DMC. This initialization value determines when to safely let the L2C know that a fill termination is coming up. Set DDR_EOF according to the following rule: ECLKFreq/DCLKFreq = DCLKPeriod/ECLKPeriod = RATIO RATIO < 6/6 → illegal 6/6 ≤ RATIO < 6/5 → DDR_EOF=3 6/5 ≤ RATIO < 6/4 → DDR_EOF=3 6/4 ≤ RATIO < 6/3 → DDR_EOF=2 6/3 ≤ RATIO < 6/2 → DDR_EOF=1 6/2 ≤ RATIO < 6/1 -→ DDR_EOF=0 6/1 ≤ RATIO → DDR_EOF=0
<16:12>	TFAW	R/W	0x0	0x9	Four access window time. Relevant only in DDR and eight-bank parts. T_{FAW} – Cycles = RNDUP(T_{FAW}/T_{CYC}) – 1 T_{FAW} = 0x0 for DDR2-4bank T_{FAW} = RNDUP(T_{FAW}/T_{CYC}) – 1 in DDR2-8bank
<11>	CRIP_MODE	R/W	0	0	Cripple mode. 1 = allow one inflight transaction, 0 = normal mode (allow 8 inflight transactions). **NOTE:** This bit can only be programmed at power-on and should not be set for normal use.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<10>	DDR2T	R/W	0	0	DDR 2T mode (2-cycle window for CMD and address): 1 = enabled, 0 = disabled This mode helps relieve setup time pressure on the address/command bus, which nominally has a very large fanout. Refer to Micron's technical note tn_47_01 “DDR2-533 Memory Design Guide for Two Dimm Unbuffered Systems” for physical details. BURST8 should be asserted when DDR2T is set, to minimize add/cmd loss.
<9>	ODT_ENA	R/W	0	0	Enable Obsolete ODT on read operations. Obsolete Read ODT wiggles DDR_ODT_* pins on reads. This field should normally be cleared to zero. When this bit is 1, the following registers must also be programmed: LMC_CTL[QS_DIC]: programs the termination value LMC_RODT_CTL: programs the ODT I/O mask for reads.
<8>	QDLL_ENA	R/W	0	0	DDR quad DLL enable: A 0→1 transition on this bit after DCLK initialization sequence resets the DDR 90 DLL. This should be done at startup, before any DDR activity. DRESET should be asserted before and for 10 µs following the 0→1 transition on QDLL_ENA.
<7:3>	DLL90_VLU	R/W	0x0	—	DDR DLL90 value. Contains the open-loop setting value for the DDR90 delay line.
<2>	DLL90_BYP	R/W	0	0	DDR DLL90 bypass: When set, the DDR90 DLL is bypassed and the setting is defined by DLL90_VLU.
<1>	RDQS	R/W	0	0	DDR2 RDQS mode. When set, configures memory subsystem to use unidirectional DQS pins. RDQS/DM — RCV, DQS — XMIT
<0>	DDR2	R/W	0x1	0x1	DDR2 enable: When set, configures memory subsystem for DDR2 SDRAMs. This field should be set.

LMC Failing-Address Register (SEC/DED) LMC_FADR

This register captures and holds only the first transaction with an ECC error. If it is a single-bit error, however, the address can be overwritten with the failing address of a double-bit error.

Writing LMC_MEM_CFG0[SEC_ERR, DED_ERR] clears the error bits and unlocks this register, allowing the capture of the next failing address.

The physical mapping is a function of the number of column bits and the number of row bits. See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31:30>	FDIMM	RO	0x0	0x0	Failing DIMM number.
<29>	FBUNK	RO	0	0	Failing rank number.
<28:26>	FBANK	RO	0x0	0x0	Failing bank number. Bits[2:0]
<25:12>	FROW	RO	0x0	0x0	Failing row address. Bits[13:0]
<11:0>	FCOL	RO	0x0	0x0	Failing column start address. Bits[11:0]. Represents the starting column address of the failing read operation (and not the exact column address in which the SEC/DED was detected)

LMC Compensation Control Register LMC_COMP_CTL

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	0x0	0x0	Reserved.
<31:28>	NCTL_CSR	R/W	0xF	—	NMOS control CSR compensation-control bits.
<27:24>	—	R/W	0x0	—	Must be 0.
<23:20>	—	R/W	0x0	—	Must be 0.
<19:16>	NCTL_DAT	R/W	0x0	—	NMOS control data compensation-control bits.
<15:12>	PCTL_CSR	R/W	0xf	—	PMOS control CSR compensation-control bits.
<11:8>	PCTL_CLK—	R/W	0x0	—	Must be 0.
<7:5>	PCTL_CMD—	R/W	0x0	—	Must be 0.
<4:0>	PCTL_DAT	R/W	0x0	—	PMOS control data compensation-control bits

LMC Write ODT Mask Register
LMC_WODT_CTL

It may be advantageous to terminate DQ/DQS/DM lines for higher-frequency DDR operations (667MHz and faster), especially on a multirank system. DDR2 DQ/DM/DQS I/O pins have built-in termination resistors that can be turned on or off by the controller, after meeting T_{AOND} and T_{AOF} timing requirements.

Each rank has its own ODT pin that fans out to all the memory parts in that DIMM. You may prefer different combinations of ODT ONs for read and write into different ranks. CN50XX supports full programmability in this area with the LMC_WODT_CTL mask register. Each rank position has its own programmable field. When the controller does a write operation to that rank, it sets the ODT pins to the MASK pins in LMC_WODT_CTL.

For example, when writing into Rank0, you may desire to terminate the lines with the resistor on Rank1. The mask WODT_HI0 and WODT_LO0 would then each be 0010 and 0010.

If ODT feature is not desired, the DDR parts can be programmed to not look at these pins by writing 0x0 into LMC_CTL[QS_DIC]. CN50XX drives the appropriate mask values on the ODT pins by default. If this feature is not required, write all 0s into this register.

When a given RANK in position *N* is selected, the WODT _LO mask for that position is used. Mask[3:0] is used for WODT control of the RANKs in positions 3, 2, 1, and 0, respectively.

DIMMs are assumed to be ordered in the following order:

- position 3: {DIMM1_RANK1_LO}
- position 2: {DIMM1_RANK0_LO}
- position 1: {DIMM0_RANK1_LO}
- position 0: {DIMM0_RANK0_LO}

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	0	0	Reserved.
<31:28>	WODT_HI3	R/W	0xF	0xF	Reserved.
<27:24>	WODT_HI2	R/W	0xF	0xF	Reserved.
<23:20>	WODT_HI1	R/W	0xF	0xF	Reserved.
<19:16>	WODT_HI0	R/W	0xF	0xF	Reserved.
<15:12>	WODT_LO3	R/W	0xF	0xF	Write ODT mask for position 3.
<11:8>	WODT_LO2	R/W	0xF	0xF	Write ODT mask for position 2.
<7:4>	WODT_LO1	R/W	0xF	0xF	Write ODT mask for position 1.
<3:0>	WODT_LO0	R/W	0xF	0xF	Write ODT mask for position 0.

MRD ECC Syndromes Register
LMC_ECC_SYND

This register provides a view of the ECC syndrome bits. In 32-bit mode, ECC is calculated on four cycles' worth of data; while in 16-bit mode, it is calculated on eight cycles of data. See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	RSVD	RAZ	—	—	Reserved.
<31:24>	MRDSYN3	RO	0x0	0x0	MRD ECC syndrome bits Quad3. In 32-bit mode: MRDSYN3 corresponds to {DQ[31:0]_c3_p1, DQ[31:0]_c3_p0}. In 16-bit mode: MRDSYN3 corresponds to {DQ[15:0]_c7_p1, DQ[15:0]_c7_p0 DQ[15:0]_c6_p1, DQ[15:0]_c6_p0}.
<23:16>	MRDSYN2	RO	0x0	0x0	MRD ECC syndrome bits Quad2. In 32-bit mode: MRDSYN2 corresponds to {DQ[31:0]_c2_p1, DQ[31:0]_c2_p0}. In 16-bit mode: MRDSYN2 corresponds to {DQ[15:0]_c5_p1, DQ[15:0]_c5_p0 DQ[15:0]_c4_p1, DQ[15:0]_c4_p0}.
<15:8>	MRDSYN1	RO	0x0	0x0	MRD ECC syndrome bits Quad1. In 32-bit mode: MRDSYN1 corresponds to {DQ[31:0]_c1_p1, DQ[31:0]_c1_p0}. In 16-bit mode: MRDSYN1 corresponds to {DQ[15:0]_c3_p1, DQ[15:0]_c3_p0 DQ[15:0]_c2_p1, DQ[15:0]_c2_p0}.
<7:0>	MRDSYN0	RO	0x0	0x0	MRD ECC syndrome bits Quad0. In 32-bit mode: MRDSYN0 corresponds to {DQ[31:0]_c0_p1, DQ[31:0]_c0_p0}. In 16-bit mode: MRDSYN0 corresponds to {DQ[15:0]_c1_p1, DQ[15:0]_c1_p0 DQ[15:0]_c0_p1, DQ[15:0]_c0_p0}.

LMC IFB Performance Counter Low Register
LMC_IFB_CNT_LO

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	IFBCNT_LO	RO	0x0	0x0	Performance counter to measure bus utilization. Lower 32-bits of the 64-bit counter that increments every cycle in which there is something in the inflight buffer

LMC IFB Performance Counter High Register
LMC_IFB_CNT_HI

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	IFBCNT_HI	RO	0x0	0x0	Performance counter to measure bus utilization. Upper 32-bits of the 64-bit counter that increments every cycle in which there is something in the inflight buffer

LMC Operations Performance Counter Low Register
LMC_OPS_CNT_LO

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	OPSCNT_LO	R/W	0x0	0x0	Lower 32-bits of the 64-bit performance counter that measures DRAM bus utilization. $\text{Bus utilization} = \frac{\text{LMC\_OPS\_CNT\_HI, LMC\_OPS\_CNT\_L}}{\text{LMC\_DCLK\_CNT\_HI, LMC\_DCLK\_CNT\_}}$

LMC Operations Performance Counter High Register
LMC_OPS_CNT_HI

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	OPSCNT_HI	R/W	0x0	0x0	Upper 32-bits of the 64-bit performance counter that measures DRAM bus utilization. $\text{Bus utilization} = \frac{\text{LMC\_OPS\_CNT\_HI, LMC\_OPS\_CNT\_L}}{\text{LMC\_DCLK\_CNT\_HI, LMC\_DCLK\_CNT\_}}$

LMC DCLK Counter Low Register
LMC_DCLK_CNT_LO

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	OPSCNT_LO	R/W	0x0	0x0	Lower 32-bits of the 64-bit performance counter that counts DCLK cycles. Used with LMC_OPS_CNT_* to measure DRAM bus utilization.

LMC DCLK Counter High Register
LMC_DCLK_CNT_HI

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	OPSCNT_HI	R/W	0x0	0x0	Upper 32-bits of the 64-bit performance counter that counts DCLK cycles. Used with LMC_OPS_CNT_* to measure DRAM bus utilization.

LMC Read ODT Control Register
LMC_RODT_CTL

On read operations, CN50XX supports turning on ODTs only in the lower two DIMMs with the masks specified in this register. Refer to the description in LMC_WODT_CTL. LMC_DDR2_CTL[ODT_ENA] must be set to 1 for CN50XX to wiggle the ODT pins on reads.

When a given RANK in position *N* is selected, the RODT _LO mask for that position is used. Mask[3:0] is used for RODT control of the RANKs in positions 3, 2, 1, and 0, respectively.

DIMMs are assumed to be ordered in the following order:

- position 3: {DIMM1_RANK1_LO}
- position 2: {DIMM1_RANK0_LO}
- position 1: {DIMM0_RANK1_LO}
- position 0: {DIMM0_RANK0_LO}

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:28>	RODT_HI3	R/W	0xF	0xF	Reserved.
<27:24>	RODT_HI2	R/W	0xF	0xF	Reserved.
<23:20>	RODT_HI1	R/W	0xF	0xF	Reserved.
<19:16>	RODT_HI0	R/W	0xF	0xF	Reserved.
<15:12>	RODT_LO3	R/W	0xF	0xF	Read ODT mask for position 3.
<11:8>	RODT_LO2	R/W	0xF	0xF	Read ODT mask for position 2.
<7:4>	RODT_LO1	R/W	0xF	0xF	Read ODT mask for position 1.
<3:0>	RODT_LO0	R/W	0xF	0xF	Read ODT mask for position 0.

LMC Open-Loop Delay Configuration Register
LMC_DELAY_CFG

This register provides the open-loop delay-line settings. The usage scenario is as follows:

- There is too much delay on command signals and setup-on-command is not met. The user can then delay the clock until setup is met.
- At the same time, however, dq/dqs should be delayed because there is also a DDR specification tying dqs with clock. If clock is too much delayed with respect to dqs, writes start to fail.

This scheme eliminates the board need of adding routing delay to clock signals to make high frequencies work.

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:15>	—	RAZ	—	—	Reserved.
<14>	—	R/W	0	0	Reserved. Must be 0.
<13:10>	DQ	R/W	0x0	0x0	DQ delay-line setting. DQ adds outgoing delay only to: dq, dqs_{p,n}, cb, cbs_{p,n}, dqm. Delay is approximately 50–80ps per setting, depending on process/ voltage. There is no need to add incoming delay since by default all strobe bits are delayed internally by 90 degrees (as was always the case in previous passes and past chips.
<9>	—	R/W	0	0	Reserved. Must be 0.
<8:5>	CMD	R/W	0x0	0x0	CMD delay-line setting. CMD adds delay to all command bits: DDR_RAS, DDR_CAS, DDR_A<15:0>, DDR_BA<2:0>, DDR_n_CS<1:0>_L, DDR_WE, DDR_CKE, DDR_ODT_<7:0>. Delay is 50–80ps per tap.
<4>	—	R/W	0	0	Reserved. Must be 0.
<3:0>	CLK	R/W	0x0	0x0	CLK delay-line setting. CLK adds delay to all clock signals: DDR_CK_<5:0>_P and DDR_CK_<5:0>_N. Delay is 50–80ps per tap.

LMC Control Register 1
LMC_CTL1

This register provides various control fields used by the LMC. See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:10>	—	RAZ	—	—	Reserved.
<9>	SIL_MODE	R/W	0	1	Read Silo mode: 0 = envelope, 1 = self-timed.
<8>	DCC_ENABLE	R/W	0	0	Duty-cycle corrector enable. 0 = disable, 1 = enable If the memory part does not support DCC, then this bit must be set to 0.
<7:2>	—	RAZ	0x0	—	Reserved.

<table>
<tr><th>Bit Pos</th><th>Field Name</th><th>Field Type</th><th>Reset Value</th><th>Typical Value</th><th>Field Description</th></tr>
<tr><td><1:0></td><td>DATA_LAYOUT</td><td>R/W</td><td>0x0</td><td>0x0</td><td>Selects data, ECC, and strobe signals used in 16-bit mode.
In 32-bit mode, this field has no effect and the ECC/data/strobe signals used are DDR_CB<3:0> / DDR_DQ<31:0> / DDR_CBS_0_P/N, DDR_DQS_<3:0>_P/N
In 16-bit mode, the ECC/data/strobe signals used are the following:
<table>
<tr><th>Field Value</th><th>Signals Used for ECC Strobe</th><th>Signals Used for ECC Bits</th><th>Signals Used for Data Strobe</th><th>Signals Used for 16 Data Bits</th></tr>
<tr><td>0x0 =</td><td>DDR_DQS_<2></td><td>DDR_DQ<17:16></td><td>DDR_DQS_<1:0></td><td>DDR_DQ<15:0></td></tr>
<tr><td>0x1 =</td><td>DDR_DQS_<3></td><td>DDR_DQ<25:24></td><td>DDR_DQS_<2:1></td><td>DDR_DQ<23:8></td></tr>
<tr><td>0x2 =</td><td>DDR_CBS_0</td><td>DDR_CB<1:0></td><td>DDR_DQS_<3:2></td><td>DDR_DQ<31:16></td></tr>
</table>
0x3 = Reserved
This assumes that ECC is enabled. If ECC is not enabled, the ECC columns are not used.</td></tr>
</table>

LMC Dual Memory Configuration Register LMC_DUAL_MEMCFG

This register controls certain parameters of dual-memory configuration. See Table 2–14 for the address.

This register enables the design to have two, separate memory configurations, selected dynamically by the reference address. Note however, that both configurations share LMC_CTL[MODE32b, XOR_BANK], LMC_MEM_CFG0[PBANK_LSB, BUNK_ENA], and all timing parameters.

In this description:

- **config0** refers to the normal memory configuration that is defined by the LMC_MEM_CFG0[ROW_LSB] and LMC_DDR2_CTL[BANK8] parameters
- **config1** refers to the dual (or second) memory configuration that is defined by this register.

Memory config0 must be programmed for the part with the most strict timing requirements. If a mix of four-bank and eight-bank parts is used, then config0 must be used for the eight-bank part (because the timing requirements of T_{FAW} and T_{RP} are more strict for eight-bank parts than they are for four-bank parts).

Enable mask to chip select mapping is shown below:

CS_MASK[3] = DDR_1_CS_<1>
CS_MASK[2] = DDR_1_CS_<0>
CS_MASK[1] = DDR_0_CS_<1>
CS_MASK[0] = DDR_0_CS_<0>

The DIMMs are arranged in one of these arrangements:

DIMM1_RANK1 (highest address)
DIMM1_RANK0
DIMM0_RANK1
DIMM0_RANK0 (lowest address)

DIMM0/1 uses the pair of chip selects DDR_n_CS_<1:0>.

- When LMC_MEM_CFG0[BUNK_ENA] = 1, each chip select in the pair asserts independently.
- When LMC_MEM_CFG0[BUNK_ENA] = 0, both chip selects in the pair assert together.

Programming restrictions for CS_MASK:

1. when LMC_MEM_CFG0[BUNK_ENA] = 0, CS_MASK[2n + 1] = CS_MASK[2n], where 0 ≤ n ≤ 3
2. when LMC_MEM_CFG0[MODE128b] = 1, CS_MASK[n + 4] = CS_MASK[n], where 0 ≤ n ≤ 3

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:20>	—	R/W	0x0	—	Must be zero.
<19>	BANK8	R/W	0	0	DDR2 8-bank: 1 = 8 banks (BA is 3 bits), 0 = 4 banks (BA is 2 bits)
<18:16>	ROW_LSB	R/W	0x3	—	Encoding used to determine which memory address bit position represents the low order DDR ROW address. The processor's memory address<33:7> needs to be translated to DRAM addresses (bnk,row,col,rank and dimm) and that is a function of the following: **1.** # Banks (4 or 8) - specified by BANK8 **2.** Datapath Width (32 or 16) - MODE32b **3.** # Ranks in a DIMM - specified by BUNK_ENA **4.** # DIMM's in the system **5.** # Column Bits of the memory part - specified indirectly by this register. **6.** # Row Bits of the memory part - specified indirectly by PBANK_LSB. Refer to Figure 2–10.
<15:4>	—	R/W	0x0	—	Must be zero.
<3:0>	CS_MASK	R/W	0x0	—	Chip select mask. This mask corresponds to the four chip selects for a memory configuration. Each reference address asserts one of the chip selects. If that chip select has its corresponding CS_MASK bit set, then the config1 parameters are used, otherwise the config0 parameters are used.

LMC Read ODT Control Register LMC_RODT_COMP_CTL

During a memory-read operation, read ODT control is available at the receivers. LMC_RODT_COMP_CTL provides three different read ODT conditions:

no ODT, weak ODT, or strong ODT.

Read ODT is automatically off during a memory-write operation, even if a weak-ODT or strong-ODT condition is chosen. Table 2–10 shows the configuration of read ODT settings.

Table 2–15 Configuration of Read ODT Setting

ODT Setting	[ENABLE]	[PCTL]	[NCTL]
No ODT	0	—	—
Weak ODT	1	00011	0001
Strong ODT	1	00111	0010

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:17>	—	R/W	0x0	—	Must be set to 0.
<16>	ENABLE	R/W	0	—	Read ODT enable: 0 = not enabled, 1 = enabled
<15:12>	—	R/W	0x0	—	Must be set to 0.
<11:8>	NCTL	R/W	0x0	—	On-die termination control bits for read operations. Refer to Table 2–10.
<7:5>	—	R/W	0x0	—	Must be set to 0.
<4:0>	PCTL	R/W	0x0	—	On-die termination control bits. Refer to Table 2–10.

LMC PLL Control Register LMC_PLL_CTL

This register controls the DDR_CK frequency. For details, refer to Section 2.3.9. See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:29>	—	R/W	0x0	—	Must be set to 0.
<28>	FASTEN_N				This field changes the allowed CLKF values. 1 = 128 < CLKF ≤ 256. 0 = 0 < CLKF ≤ 128.
<27>	DIV_RESET	R/W	1	0	Analog PLL divider reset Deassert at least 500 × (CLKR + 1) reference-clock cycles following RESET_N deassertion.
<26>	RESET_N	R/W	0	1	Analog PLL reset Deassert at least 5 µs after CLKF, CLKR, and EN* are set up.
<25:14>	CLKF	R/W	0x1F	—	Multiply reference by CLKF + 1. CLKF constraints are specified by FASTEN_N
<13:8>	CLKR	R/W	0x0	—	Divide reference by CLKR + 1.
<7:6>	—	R/W	0x0	—	Must be set to 0.
<5>	EN16	R/W	0	—	Divide output by 16.
<4>	EN12	R/W	0	—	Divide output by 12.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<3>	EN8	R/W	1	—	Divide output by 8.
<2>	EN6	R/W	0	—	Divide output by 6.
<1>	EN4	R/W	0	—	Divide output by 4.
<0>	EN2	R/W	0	—	Divide output by 2.

LMC PLL Status Register
LMC_PLL_STATUS

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	R/W	0x0	—	Must be zero.
<31:27>	DDR__NCTL	RO	—	—	DDR NCTL from compensation circuit.
<26:22>	DDR__PCTL	RO	—	—	DDR PCTL from compensation circuit.
<21:2>	—	R/W	0x0	—	Must be zero.
<1>	RFSLIP	R/W1C	0	—	Reference clock slip.
<0>	FBSLIP	R/W1C	0	—	Feedback clock slip.

LMC BIST Control Register
LMC_BIST_CTL

This register controls BIST only for the memories that operate on DCLK. The normal chip-wide BIST flow controls BIST for the memories that operate on core clock. See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	X	0x0	Reserved. Must be zero.
<0>	START	R/W	0	0	A 0 → 1 transition causes BIST to run.

LMC BIST Result Register
LMC_BIST_RESULT

This register provides access to the internal BIST results. Each bit is the BIST result of an individual memory. For each bit, 0 = pass or BIST in progress/never run; 1 = fail.

See Table 2–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:9>	—	RAZ	X	0x0	Reserved. Must be zero.
<8>	MWF	RO	X	0	BIST result of MWF memories.
<7:5>	MWD	RO	X	0x0	BIST result of MWD memories.
<4>	MWC	RO	X	0	BIST result of MWC memories.
<3>	MRF	RO	X	0	BIST result of MRF memories.
<2:0>	MRD	RO	X	0x0	BIST result of MRD memories.

Chapter 3

I/O Busing, I/O Bridge (IOB) and Fetch and Add Unit (FAU)

This chapter describes the following topics:

3.1 CN50XX I/O Busing

3.1.1 I/O Busing Overview

Figure 3–1 shows the CN50XX I/O and coprocessor components and their physical connections, i.e. the I/O buses. The I/O buses include

- an inbound bus (IOBI)
- an outbound bus (IOBO)
- a packet output bus (POB)
- a PKO-specific bus (PKOB)
- an input packet-data bus (IPDB)
- associated controls.

The I/O buses all run at the core-clock frequency. I/O and coprocessor devices use these buses to communicate among themselves and, through the I/O bridge (IOB), to communicate with the L2 Cache (L2C) and cores on the coherent memory bus (CMB).

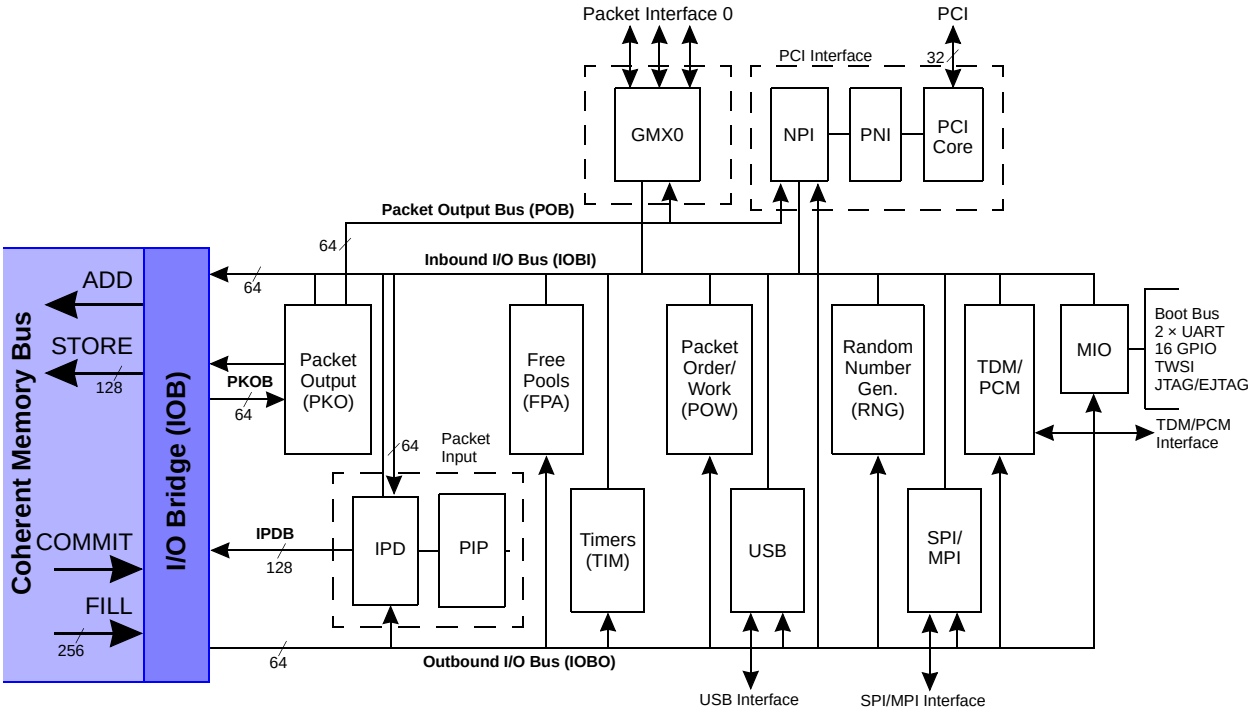


Figure 3–1 I/O Bus Block Diagram

3.1.2 I/O Bus Flow Examples

The following figures are examples of transactions on the CN50XX internal I/O buses to illustrate their usages.

Figure 3–2 shows I/O bus flows for a packet data input example. The packet data first arrives on either the RGMII, GMII, MII, or PCI interfaces and is captured by the RGMII/GMII/MII (GMX) or PCI input interfaces, respectively. This is not shown in the figure. Figure 3–2 does, however, show the packet input data pieces (each of up to 64-bits) eventually placed by one of the I/O interfaces onto the CN50XX-internal IOBI bus. The I/O interface also had to previously arbitrate for the IOBI bus cycles, as do all other IOBI drivers. The CN50XX central packet-input hardware (IPD) latches the packet-data pieces directly from the IOBI bus for processing. There is much processing that occurs, but one particular aspect regarding this example is that the IPD hardware accumulates the 64-bit packet-data pieces from the IOBI bus into full (128 byte) cache blocks. IPD then forwards the cache block writes on the IPDB bus. The IOB later forwards the cache block write onto the CMB.

NOTE: A full 64-bit (per core cycle) data path, using only the IOBI and IPDB buses, exists internally to capture store input packet data into L2/DRAM. This gives a raw hardware packet input processing capability in excess of 30 Gbits/sec.

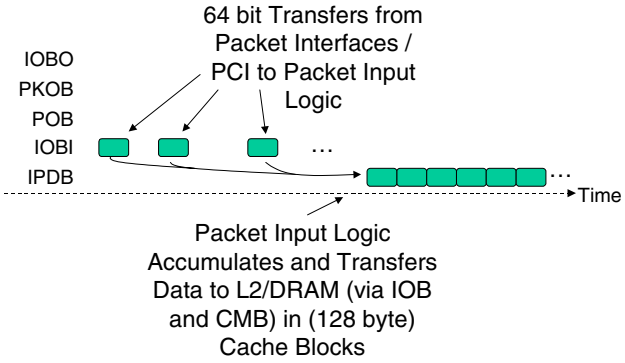


Figure 3–2 I/O Bus Flows for a Packet Data Input

Figure 3–3 shows I/O bus flows for packet data output. The IOB first fills data using the CMB (not shown). An individual fetch is up to 128 bytes; the cache block size. The IOB returns the filled data to the centralized packet-output hardware (PKO) using the PKOB bus. PKO buffers the PKOB data and eventually forwards it onto the Packet Output Bus (POB), though PKO has large buffers (because it supports in-line TCP/UDP checksum generation), so there may be considerable time delay between the PKOB and POB buses.

NOTE: A full 64-bit (per core cycle) data path, using only the PKOB and POB buses, exists internally to send packet-output data off-chip. This gives a raw hardware packet-output processing capability in excess of 30 Gbits/sec. This bandwidth is available independently of the packet-input data flow. This is for internal busing only. External interfaces may limit the sustained bandwidth.

Figure 3–4 has an I/O bus flow example that uses the hardware free pools (FPA).

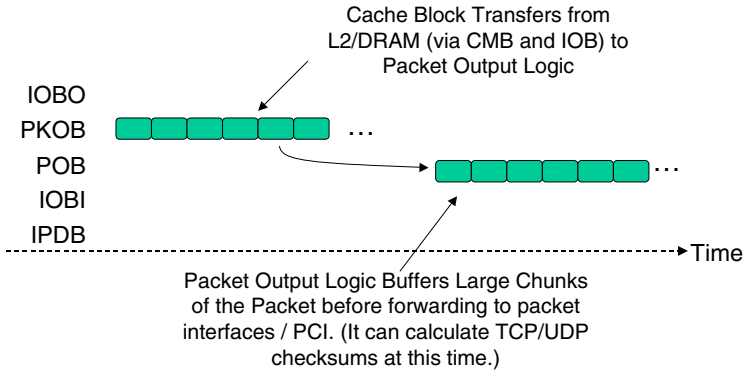


Figure 3–3 I/O Bus Flows for Packet Data Output

This example shows transactions resulting from an allocation request to a free pool. First (not shown), the core issues a load/IOBDMA to an address corresponding to FPA, and the address is reflected through the CMB. Later (shown), the IOB bridges the command and address onto the IOBO bus (in green) and the FPA hardware recognizes that it is the destination for the address. The FPA hardware processes the command, returning the result, (i.e. a pointer to the available memory), later on the IOBI bus (also in green). Figure 3–4 also shows an FPA DMA read access that (infrequently) may be required to replenish the pool of pointers contained in FPA. First, the FPA hardware places the L2/DRAM address onto the IOBI bus (shown in tan). The IOB bridges this address onto the coherent memory bus (not shown) and eventually returns the DMA read data containing the additional pointers on the IOBO bus (shown in tan).

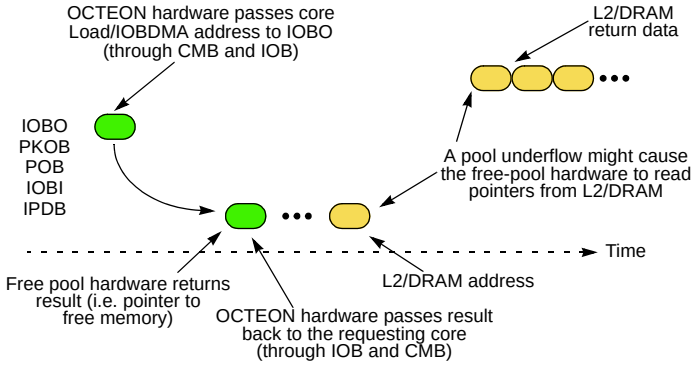


Figure 3–4 I/O Bus Flow: Memory Allocate

Figure 3–5 has a similar I/O bus flow example pertaining to the FPA, but this example is of a memory free rather than a memory allocate. The core freeing the memory executes a store to an address and the hardware reflects the store through the CMB (not shown). (This example also does not show the Don't-Write-Back (DWB) CMB commands that IOB hardware may create as a result of the memory free command.) The IOB bridges the address/data pair onto the IOBO bus (shown in green), the FPA hardware recognizes it, and buffers the pointer to the available memory in the pool within the FPA block. Figure 3–5 also shows an FPA DMA write access that (infrequently) may be required to free up space in the pool within the FPA block. The FPA hardware places the DMA address and data onto the IOBI bus (shown), which the IOB bridges onto the CMB (not shown).

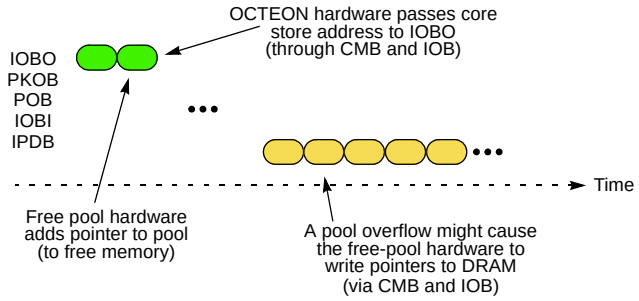


Figure 3–5 I/O Bus Flow: Memory Free

3.2 IOB Architecture

3.2.1 IOB Architecture Overview

Figure 3–6 shows a block diagram of the IOB internal architecture. The CMB is on the left side of the figure and the I/O buses are in the right, the top, and the bottom of the figure.

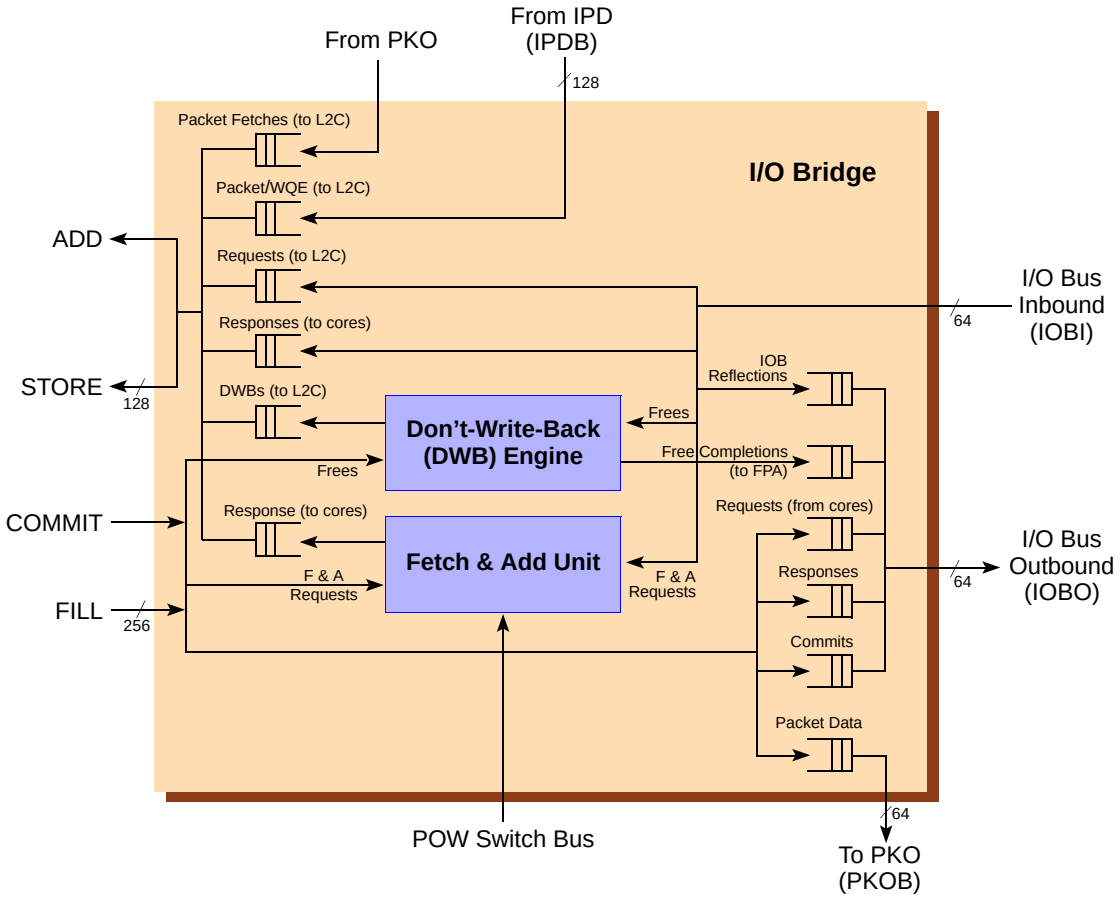


Figure 3–6 I/O Bridge Block Diagram

The primary function of the IOB is a bridge, so the largest component of the IOB architecture is queues to hold information transferred on the different buses. There are six queues arbitrating to transfer on the ADD/STORE buses of the CMB and five queues arbitrating to transfer on the IOBO bus. The multiple queues are required for a number of reasons: deadlock-avoidance, different data sources, and performance.

3.3 Don't-Write-Back Engine

Another important component within the IOB is the DWB Engine. This engine intercepts memory-free requests destined to the free-pool hardware (FPA). The memory-free requests include a hint indicating the number of DWB CMB transactions that the IOB hardware can issue. The memory has just been freed and it will not be used until it is reallocated, so it would be wasteful for the hardware to write the cache blocks from the level 2 cache back to the DRAM. The DWB commands will cause the dirty bit for the selected blocks to be cleared, thus avoiding these wasteful write-backs.

The IOB can intercept these memory-free commands arriving from either the cores (via a reflection onto the CMB COMMIT/FILL buses) or from other OCTEON hardware units (via the IOBI bus). The IOB can buffer a limited number of the memory free commands inside DWB. If buffering is available, the IOB intercepts the memory free request until it has finished issuing the CMB DWB commands for the request, and then forwards the request onto the FPA hardware (via the IOBO bus). If the DWB buffering is not available, the IOB DWB unit does not intercept the memory free command, and the command instead goes to the FPA hardware without the FPA issuing any DWB commands. This implementation is possible since it is optional for the hardware to issue the DWBs.

3.4 Fetch and Add Unit (FAU)

Figure 3–7 expands on the final important component of the IOB architecture - the fetch-and-add unit (FAU). The FAU is a 2KB register file supporting read, write, atomic fetch-and-add, and atomic update operations. The unit can be accessed from both the cores and the centralized packet-output (PKO) unit. The cores use the FAU for general synchronization purposes, including applications like assigning unique positions for inserting items into shared queues.

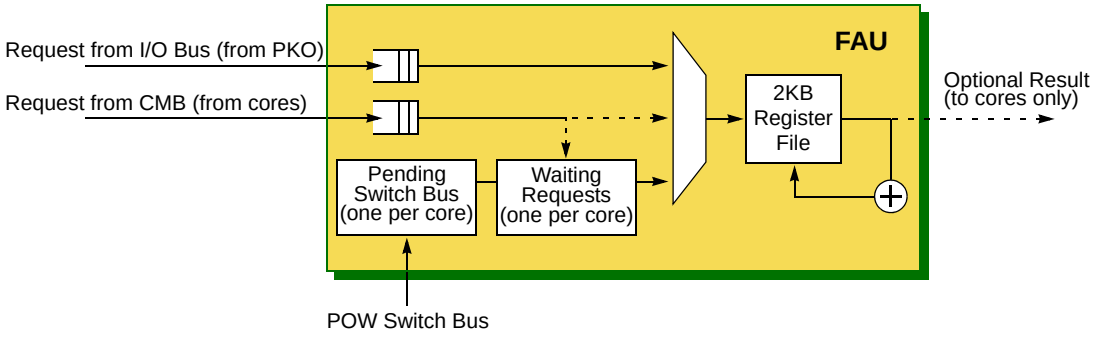


Figure 3–7 FAU Block Diagram

FAU registers are used particularly to maintain lengths of the output queues (up to 32) to transfer packets off-chip (via PKO). This is the reason that the PKO submits commands to the FAU. Core software increments counters in the FAU when it adds a packet to a queue. Under software command, the PKO decrements the same counter after it removes the packet from a queue and sends it off-chip. This way, the register continually maintains the size of the output queue (in packets and/or bytes). The core software will require the current queue size if it must implement Quality Of Service (QOS) algorithms like Random Early Discard (RED).

Another feature of the FAU is its tight integration with the tag switch synchronization provided by the packet order/work unit (POW). This tight integration can minimize critical section latencies when both tag switch and FAU synchronization is required.

The FAU tracks the pending tag switch state for each core. The FAU considers a tag switch to be pending from when the POW tag switch store (that uses the POW subDID of 0) issues until the switch completion indication returns via the POW switch bus. This FAU pending switch logic is, in general, a conservative estimator of the pending switch state held at both the cores and POW. For switches that are not descheduled, it tracks exactly.

Each core can specify the "tag-wait" feature for up to one outstanding FAU request at any time. (The number of outstanding FAU requests that do not specify the tag-wait feature is never restricted by the hardware.) When this option is enabled, the FAU attempts to delay the request until the prior tag switch is no longer pending. While the hardware delays the request, it buffers it in a store that contains one entry per core. Thus, the restriction on only one outstanding tag-wait request from each core.

The fetch-and-add unit processes requests without the tag-wait feature enabled immediately. It also processes requests with the preferred tag-wait requests immediately when (it thinks) there is not a pending switch. If the hardware buffers a tag-wait request for too long, it removes the request from the buffer and returns a failure indication to the requester without performing the operation specified in the request. The tag-wait time-out length is variable based on configuration, in multiples of 256 internal clock cycles.

Figure 3–7 shows all the components of the fetch-and-add hardware. The hardware arbitrates between the three different sources of requests; the cores, the PKO, and the buffered core tag-wait requests. The hardware completes the requests by reading and writing the 2KB register file. Some core-generated FAU requests require responses to be reflected back through the CMB.

Chapter 8 describes the software interface to the PKO.

3.5 Fetch-and-Add Operations

Core operations to the FAU issue through Load, Store, and IOBDMA instructions.

For Load and Store instructions, the FAU operation size matches the size specified by the instruction. For example, a core byte load instruction (i.e. LB or LBU) initiates a byte operation on the register file. This means that the register value can change, and the load result is returned, only for the particular byte in the register file referenced by the LB/LBU instruction. All the core load/store instruction sizes can be used to operate on the fetch-and-add register file:

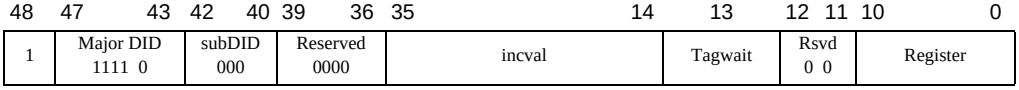
- 8-bit operation (LB/LBU/SB)
- 16-bit operation (LH/LHU/SH)
- 32-bit operation (LW/LWU/SW)
- 64-bit operation (LD/SD)

IOBDMA instructions destined for the FAU always return a 64-bit result to the core, but the actual operation performed by the hardware, and the effective result returned, may be either 8-bit, 16-bit, 32-bit, or 64-bit.

The configuration of the particular core that executes the load/store selects the endianness of the 8-bit, 16-bit, and 32-bit loads/stores. The FAU has a IOB_CTL_STATUS[FAU_END] bit that can specify either little-endian or big-endian addressing for IOBDMA instructions. This IOBDMA endian configuration value is common to both cores.

3.5.1 Load Operations

Load Physical Address for FAU Operations



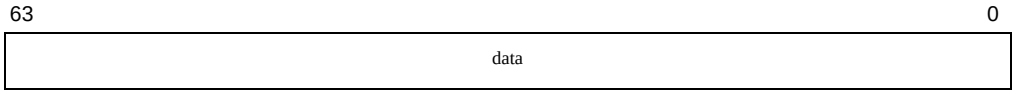
- **incval -** The value to add to the register after returning the load result. This is a 22-bit signed value, of which only the bottom 8-bits are used for 8-bit operations and the bottom 16-bits for 16-bit operations.
- **Tagwait -** If set, the hardware will attempt to delay servicing the request until after the prior tag switch completes.
- **Register -** Selects a particular FAU register.
  - **<10:3>** selects the register
  - **<2:0>** selects a byte offset into that register

NOTE: Register field must be naturally aligned to the size. This is required by the core.

NOTE: The value returned for a load or IOBDMA is the old value from the previous add operation.

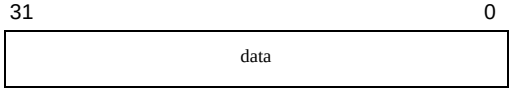
Load Operation Result In Cases Where Tagwait = 0:

64-bit operation result for a LD



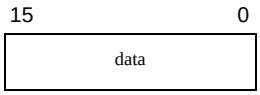
- **data -** data is current value of register location.

32-bit operation result for a LW/LWU



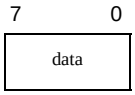
- **data -** data is current value of register location.

16-bit operation result for a LH/LHU



- **data -** data is current value of register location.

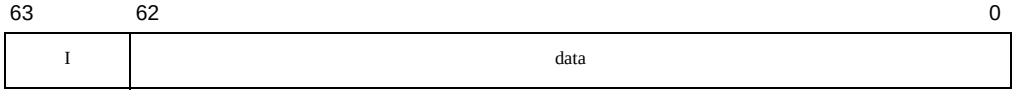
8-bit operation result for a LB/LBU



- **data -** data is current value of register location.

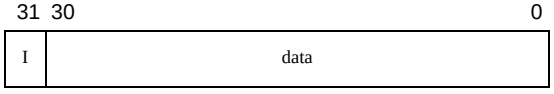
Load Operation Result In Cases Where Tagwait = 1:

64-bit operation result for a LD



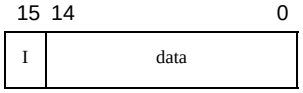
- **I -** Operation did not complete.
- **data -** Unpredictable if I is set. Otherwise, data is current value of the lower 63-bits of the register location.

32-bit operation result for a LW/LWU



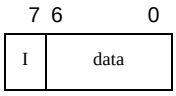
- **I -** Operation did not complete.
- **data -** unpredictable if I is set. Otherwise, data is current value of the lower 31-bits of the register location.

16-bit operation result for a LH/LHU



- **I -** Operation did not complete.
- **data -** unpredictable if I is set. Otherwise, data is current value of the lower 15-bits of the register location.

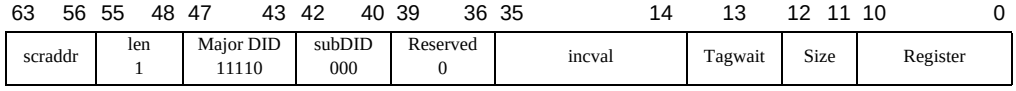
8-bit operation result for a LB/LBU



- **I -** Operation did not complete.
- **data -** unpredictable if I is set. Otherwise, data is current value of the lower 7 bits of the register location.

3.5.2 IOBDMA Operations

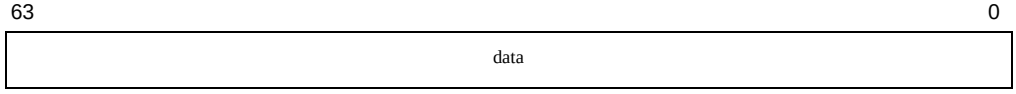
IOBDMA Store Data for FAU Operations



- **scraddr -** Defined in "cnMIPS™ Cores" on page 143.
- **len -** Must be 1. Defined in "cnMIPS™ Cores" on page 143.
- **incval -** The value to add to the register. This is a 22-bit signed value, of which only the bottom 8 bits are used for 8-bit operations and the bottom 16 bits for 16-bit operations.
- **Tagwait -** If set, the hardware will attempt to delay servicing the request until after the prior tag switch completes.
- **Size** - Indicates the size of the operation
  0 = 8-bit
  1 = 16-bit
  2 = 32-bit
  3 = 64-bit
- **Register -** Selects a particular FAU register.

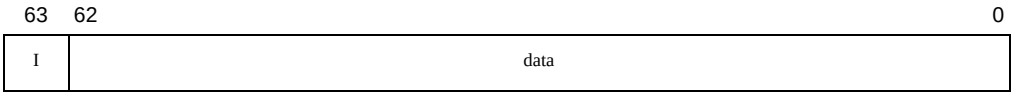
NOTE: Register field must be naturally aligned to the size.

64-bit operation (tagwait=0) result for an IOBDMA (in CVMSEG_LM)



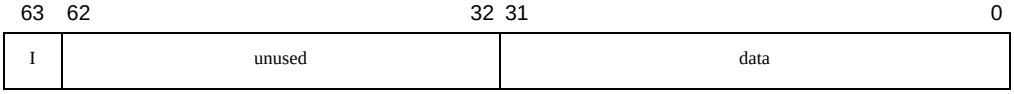
- **data -** data is current value of register location.

64-bit operation (tagwait=1) result for an IOBDMA (in CVMSEG_LM)



- **I -** Incomplete. When I = 1, the operation did not complete and the data is unpredictable.
- **data -** Unpredictable when I = 1. Otherwise, data is current value of the lower 63 bits of the register location.

32-bit operation result for an IOBDMA (in CVMSEG_LM)

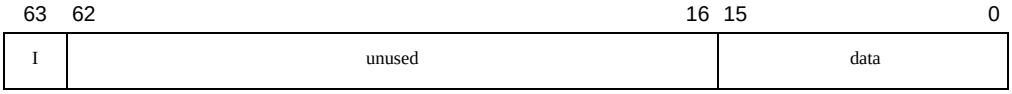


- **I -** Incomplete. The hardware always forces I = 0 when tagwait = 0. When tagwait = 1, then I may be 1 or 0.

  I = 1, the operation did not complete and the data is unpredictable.
  I = 0, the operation completed.

- **unused -** Unpredictable when I = 1; otherwise all 0s.
- **data -** Unpredictable when I = 1. Otherwise, data is the current value of the 32-bit register.

16-bit operation result for an IOBDMA (in CVMSEG_LM)

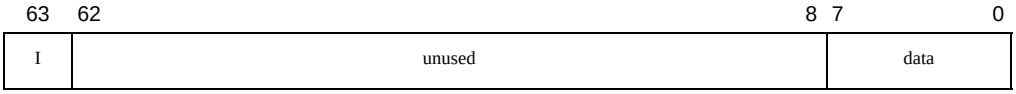


- **I -** Incomplete. The hardware always forces I = 0 when tagwait = 0. When tagwait = 1, then I may be 1 or 0.

  I = 1, the operation did not complete and the data is unpredictable.
  I = 0, the operation completed.

- **unused -** Unpredictable when I = 1; otherwise all 0s.
- **data -** Unpredictable when I = 1. Otherwise, data is the current value of the 16-bit register.

8-bit operation result for an IOBDMA (in CVMSEG_LM)



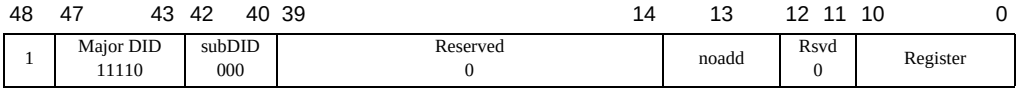
- **I -** Incomplete. The hardware always forces I = 0 when tagwait = 0. When tagwait = 1, then I may be 1 or 0.

  I = 1, the operation did not complete and the data is unpredictable.
  I = 0, the operation completed.

- **unused -** Unpredictable when I = 1; otherwise all 0s.
- **data -** Unpredictable when I = 1. Otherwise, data is the current value of the 8-bit register.

3.5.3 Store Operations

Store Physical Address for FAU Operations



- **noadd -** If clear, do an (atomic) update, else just store
- **Register -** Selects a particular FAU register.

NOTE: Register field must be naturally aligned to the size. This is required by the core.

The store address above can be used for any of the core 8-bit (SB), 16-bit (SH), 32-bit (SW), or 64-bit (SD) store instructions to produce FAU operations of the corresponding size. The no add bit selects whether to add the store data to the previous register value or to simply over-write the prior register value.

3.6 IOB Registers

The IOB registers are listed in Table 3–1.

Table 3–1 IOB Registers

Register	Address	CSR Type[1]	Detailed Description
IOB_FAU_TIMEOUT	0x00011800F0000000	RSL	See page 138
IOB_CTL_STATUS	0x00011800F0000050	RSL	See page 138
IOB_INT_SUM	0x00011800F0000058	RSL	See page 138
IOB_INT_ENB	0x00011800F0000060	RSL	See page 139
IOB_PKT_ERR	0x00011800F0000068	RSL	See page 139
IOB_INB_DATA_MATCH	0x00011800F0000070	RSL	See page 139
IOB_INB_CONTROL_MATCH	0x00011800F0000078	RSL	See page 140
IOB_INB_DATA_MATCH_ENB	0x00011800F0000080	RSL	See page 140
IOB_INB_CONTROL_MATCH_ENB	0x00011800F0000088	RSL	See page 140
IOB_OUTB_DATA_MATCH	0x00011800F0000090	RSL	See page 140
IOB_OUTB_CONTROL_MATCH	0x00011800F0000098	RSL	See page 141
IOB_OUTB_DATA_MATCH_ENB	0x00011800F00000A0	RSL	See page 141
IOB_OUTB_CONTROL_MATCH_ENB	0x00011800F00000A8	RSL	See page 141
IOB_BIST_STATUS	0x00011800F00007F8	RSL	See page 142

1. RSL-type registers are accessed indirectly across the I/O Bus.

Fetch and Add Unit Tag-Switch Time-Out Register IOB_FAU_TIMEOUT

See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:13>	—	RAZ	—	—	Reserved
<12>	TOUT_ENB	R/W	1	0	Enable FAU time-out: 1 = enable the time-out, 0 = disable.
<11:0>	TOUT_VAL	R/W	0x4	0x4	When a tag request arrives from the core a timer is started associated with that core. The timer, which increments every 256 core clock cycles, is compared to TOUT_VAL. When the two are equal, the IOB flags the tag request to complete as a time-out tag operation. The 256-count timer used to increment the core-associated timer is always running, so the first increment of the core-associated timer may occur anywhere within the first 256 core clock cycles.

I/O Bridge Control and Status Register IOB_CTL_STATUS

See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved.
<4>	OUTB_MAT	R/W1C	0	0	Outbound match. When this bit is set, it indicates there was a match on the outbound bus to the outbound pattern matchers.
<3>	INB_MAT	R/W1C	0	0	Inbound match. When this bit is set, it indicates there was a match on the inbound bus to the inbound pattern matchers.
<2>	PKO_ENB	R/W	0	0	PKO endian style. Toggles the endian style of the FAU for the PKO. 0 = big-endian, 1 = little-endian.
<1>	DWB_ENB	R/W	1	1	Enable DWB. Enables the don't-write-back function of the IOB.
<0>	FAU_END	R/W	0	0	FAU endian style. Toggles the endian style of the FAU. 0 = big-endian, 1 = little-endian.

I/O Bridge Interrupt Summary Register IOB_INT_SUM

Contains the interrupt summary bits of the IOB. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	—	—	Reserved
<3>	P_EOP	R/W1C	0	0	Set when a EOP is followed by an EOP for the same port for a passthrough packet. The first detected error associated with bits <3:0> of this register will only be set here. A new bit can be set when the previous reported bit is cleared.
<2>	P_SOP	R/W1C	0	0	Set when a SOP is followed by an SOP for the same port for a passthrough packet. The first detected error associated with bits <3:0> of this register will only be set here. A new bit can be set when the previous reported bit is cleared.
<1>	NP_EOP	R/W1C	0	0	Set when a EOP is followed by an EOP for the same port for a nonpassthrough packet. The first detected error associated with bits <3:0> of this register will only be set here. A new bit can be set when the previous reported bit is cleared.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<0>	NP_SOP	R/W1C	0	0	Set when a SOP is followed by an SOP for the same port for a nonpassthrough packet. The first detected error associated with bits <3:0> of this register will only be set here. A new bit can be set when the previous reported bit is cleared.

I/O Bridge Interrupt Enable Register
IOB_INT_ENB

Contains the interrupt summary bits of the IOB. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	—	—	Reserved
<3>	P_EOP	R/W1C	0	0	Passthrough packet EOP interrupt enable. When this bit is set and IOB_INT_SUM[P_EOP] is set, the IOB asserts an interrupt.
<2>	P_SOP	R/W1C	0	0	Passthrough packet SOP interrupt enable. When this bit is set and IOB_INT_SUM[P_SOP] is set, the IOB asserts an interrupt.
<1>	NP_EOP	R/W1C	0	0	Nonpassthrough packet EOP interrupt enable. When this bit is set and IOB_INT_SUM[NP_EOP] is set, the IOB asserts an interrupt.
<0>	NP_SOP	R/W1C	0	0	Nonpassthrough packet SOP interrupt enable. When this bit is set and IOB_INT_SUM[NP_SOP] is set, the IOB asserts an interrupt.

I/O Bridge Packet Error Register
IOB_PKT_ERR

Provides status about the failing packet receive error. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	—	—	Reserved
<5:0>	PORT	RO	0	—	When an IOB_INT_SUM[3:0] bit is set, this field latches the failing port associate with the IOB_INT_SUM[3:0] bit set.

I/O Bridge Inbound Data Match Register
IOB_INB_DATA_MATCH

This register provides the match pattern for inbound data that is used to set IOB_CTL_STATUS[INB_MAT]. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	DATA	R/W	0x0	—	Pattern to match on the inbound I/O bus.

I/O Bridge Inbound Control Match Register IOB_INB_CONTROL_MATCH

This register provides the match pattern for inbound control that is used to set IOB_CTL_STATUS[INB_MAT]. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:29>	—	RAZ	—	—	Reserved
<28:21>	MASK	R/W	0x0	—	Pattern to match on the inbound I/O bus.
<20:17>	OPC	R/W	0x0	—	Pattern to match on the inbound I/O bus.
<16:8>	DST	R/W	0x0	—	Pattern to match on the inbound I/O bus.
<7:0>	SRC	R/W	0x0	—	Pattern to match on the inbound I/O bus.

I/O Bridge Inbound Data Match Enable Register IOB_INB_DATA_MATCH_ENB

Enables the match of the corresponding bit in IOB_INB_DATA_MATCH. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	DATA	R/W	0x0	—	Bits that are set in this field enable the corresponding bits in IOB_INB_DATA_MATCH to be matched.

I/O Bridge Inbound Control Match Enable Register IOB_INB_CONTROL_MATCH_ENB

Enables the match of the corresponding bit in IOB_INB_CONTROL_MATCH. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:29>	—	RAZ	—	—	Reserved
<28:21>	MASK	R/W	0x0	—	Bits that are set in this field enable the corresponding bits in IOB_INB_CONTROL_MATCH to be matched.
<20:17>	OPC	R/W	0x0	—	Bits that are set in this field enable the corresponding bits in IOB_INB_CONTROL_MATCH to be matched.
<16:8>	DST	R/W	0x0	—	Bits that are set in this field enable the corresponding bits in IOB_INB_CONTROL_MATCH to be matched.
<7:0>	SRC	R/W	0x0	—	Bits that are set in this field enable the corresponding bits in IOB_INB_CONTROL_MATCH to be matched.

I/O Bridge Outbound Data Match Register IOB_OUTB_DATA_MATCH

This register provides the match pattern for outbound data that is used to set IOB_CTL_STATUS[OUTB_MAT]. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	DATA	R/W	0x0	—	Pattern to match on the outbound I/O bus.

I/O Bridge Outbound Control Match Register IOB_OUTB_CONTROL_MATCH

This register provides the match pattern for outbound control that is used to set IOB_CTL_STATUS[OUTB_MAT]. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:26>	—	RAZ	—	—	Reserved
<25:18>	MASK	R/W	0x0	—	Pattern to match on the outbound I/O bus.
<17>	EOT	R/W	0	—	Pattern to match on the outbound I/O bus.
<16:9>	DST	R/W	0x0	—	Pattern to match on the outbound I/O bus.
<8:0>	SRC	R/W	0x0	—	Pattern to match on the outbound I/O bus.

I/O Bridge Outbound Data Match Enable Register IOB_OUTB_DATA_MATCH_ENB

Enables the match of the corresponding bit in IOB_OUTB_DATA_MATCH. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	DATA	R/W	0x0	—	Bits that are set in this field enable the corresponding bits in IOB_OUTB_DATA_MATCH to be matched.

I/O Bridge Outbound Control Match Enable Register IOB_OUTB_CONTROL_MATCH_ENB

Enables the match of the corresponding bit in IOB_OUTB_CONTROL_MATCH. See Table 3–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:26>	—	RAZ	—	—	Reserved
<25:18>	MASK	R/W	0x0	—	Bits that are set in this field enable the corresponding bits in IOB_OUTB_CONTROL_MATCH to be matched.
<17>	EOT	R/W	0	—	Bits that are set in this field enable the corresponding bits in IOB_OUTB_CONTROL_MATCH to be matched.
<16:9>	DST	R/W	0x0	—	Bits that are set in this field enable the corresponding bits in IOB_OUTB_CONTROL_MATCH to be matched.
<8:0>	SRC	R/W	0x0	—	Bits that are set in this field enable the corresponding bits in IOB_OUTB_CONTROL_MATCH to be matched.

BIST Status of IOB Memories Register IOB_BIST_STATUS

This register shows the result of the BIST run on the IOB memories. See Table 3–1 for address. A 1 in any of the bits indicates a BIST error. A 0 indicates the BIST passed or never ran.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:14>	—	RAZ	—	—	Reserved
<13>	ICNR0	RO	0	0	icnr_reg_mem0_bist_status.
<12>	IBDR0	RO	0	0	ibdr_bist_req_fifo0_status.
<11>	IBDR1	RO	0	0	ibdr_bist_req_fifo1_status.
<10>	IBR0	RO	0	0	ibr_bist_rsp_fifo0_status.
<9>	IBR1	RO	0	0	ibr_bist_rsp_fifo1_status.
<8>	—	RAZ	—	—	Reserved
<7>	IBRQ0	RO	0	0	ibrq_bist_req_fifo0_status.
<6>	IBRQ1	RO	0	0	ibrq_bist_req_fifo1_status.
<5>	ICRN0	RO	0	0	icr_ncb_bist_mem0_status.
<4>	ICRN1	RO	0	0	icr_ncb_bist_mem1_status.
<3>	ICRP0	RO	0	0	icr_pko_bist_mem0_status.
<2>	ICRP1	RO	0	0	icr_pko_bist_mem1_status.
<1>	IBD	RO	0	0	ibd_bist_mem0_status.
<0>	ICD	RO	0	0	icd_ncb_fifo_bist_status.

Chapter 4

cnMIPS™ Cores

This chapter contains the following subjects:

Cavium Networks, Inc.'s OCTEON Plus CN50XX architecture is based on officially licensed MIPS Inc. Technology. For reference to MIPS® Technology found in CN50XX, you must have the following books:

Table 4–1 MIPS Publications

Publication	Document Number
EJTAG Specification, Revision 3.10, July 5, 2005	MD00047
MIPS64® Architecture For Programmers Volume I: Introduction to the MIPS64® Architecture, Revision 2.00, June 8, 2003	MD00083
MIPS64® Architecture For Programmers Volume II: The MIPS64® Instruction Set, Revision 2.00, June 9, 2003	MD00087
MIPS64® Architecture For Programmers Volume III: The MIPS64® Privileged Resource Architecture, Revision 2.00, June 9, 2003	MD00091

Overview

The OCTEON Plus CN50XX's cnMIPS™ cores, shown in Figure 4–1, support MIPS64® version 2 integer instruction set and privileged architecture. CN5010 has one cnMIPS™ processor core; CN5020 has two cnMIPS™ processor cores.

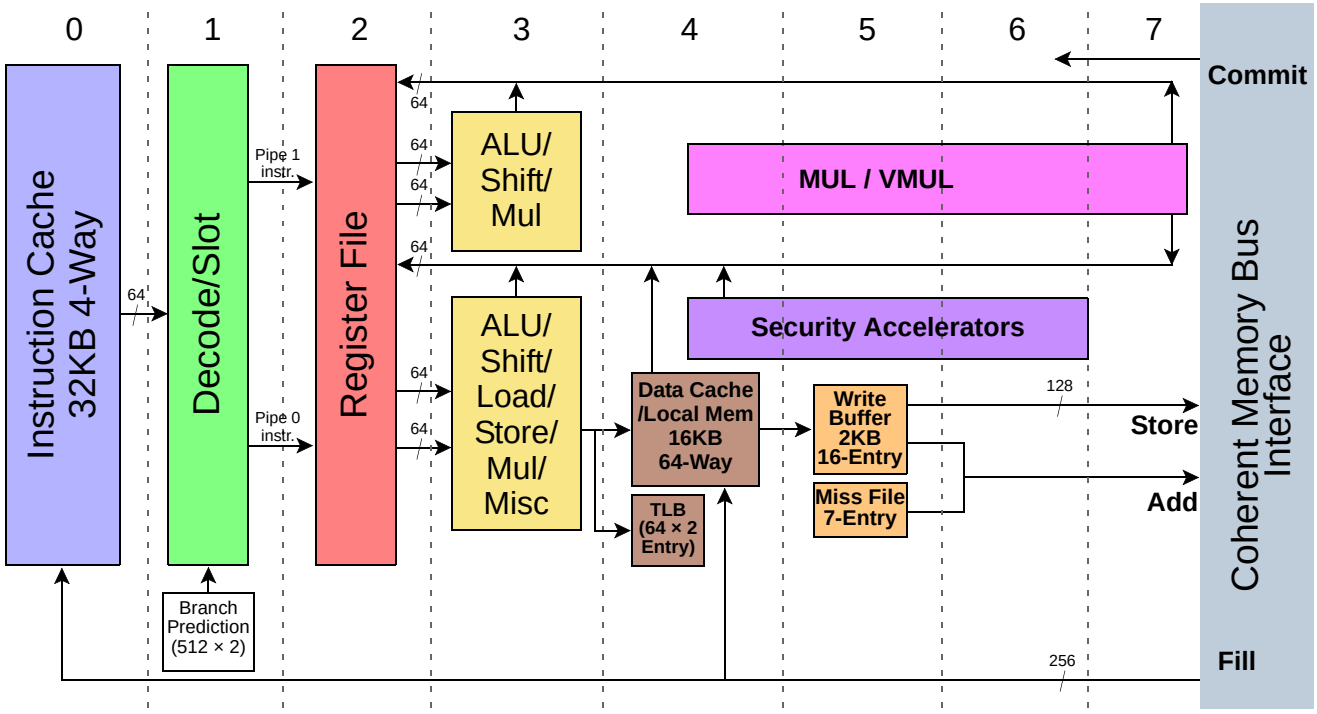


Figure 4–1 cnMIPS™ Core Block Diagram

Instruction fetch/decode operates at two instructions fetched per cycle. Up to two instructions can be issued per cycle, and one branch prediction per cycle. The cnMIPS cores have four-entry subroutine return call stacks and hardware instruction prefetching.

cnMIPS core pipelines are capable of a total of 2.8 GOPS peak (2 cores @ 700 MHz).

The cnMIPS core Load/Store units can do a total of 1.4 billion load/store operations/sec (two cores, 700 MHz), and have a high-performance coherent memory bus interface supporting up to 384 Gbps of data transfer.

4.1 Summary of cnMIPS Core Features

4.1.1 MIPS64 Version 2.0 Implementation

- Dual-issue, 5+ stage pipeline
- Up to 700 MHz in 90 nm process
- Floating-point not implemented
- Big and little endian support
- Aggressive conditional clocking for minimal power dissipation
- 32KB, 4-way virtual instruction cache (Icache)
  - Bitmask LRU or random replacement
  - Hardware single-bit-error correction
- Hardware instruction prefetching

- 512 entry × 2-bit branch prediction
- Four-entry subroutine-call return stack
- 16KB, 64-way data cache (Dcache)
  - Fully coherent with all multicore and IO DRAM references
  - Write-through, Invalidate protocol
  - Bitmask LRU replacement
  - Hardware single-bit-error correction
- High-performance coherent memory bus interface
- Up to 7 outstanding fills
- 2KB, 16 entry merging write-buffer
  - Aggressive merging for highest performance
  - Allows many outstanding stores
  - Marked differentiation for looser ordering via SYNCS or SYNCWS
  - Flushed by a SYNC or SYNCW
  - Partially flushed by a SYNCS or SYNCWS
  - High-performance write synchronization
  - Programmable overflow threshold
  - Programmable merge closure time-out

4.1.2 Cavium-Specific Architectural Additions

- Cavium-specific instruction additions
  - Bit-test branches
  - Bit-field signed extracts
  - Bit-field clear and insert
  - Unsigned byte add
  - Set equals, set not-equals
  - 32-bit and 64-bit population count
  - RDHWR additions
    - 64-bit cycle counter
    - fast POW tag switch access
  - True unaligned loads/stores
  - New SYNC instructions
  - PREF implementation
    - Many PREFs with different L1/L2 cache behavior
    - Don't write-back operation added
  - Cache implementation
- Cavium-specific in-core coprocessor 2 security accelerators
  - CRC (any polynomial 32-bit or less)
  - Hash (MD5/SHA-1/SHA-256/SHA-512)
  - 3DES
  - AES
  - KASUMI
  - Galois Field Multiplier

- Cavium-specific integer multiply unit
  - 64-bit register-direct integer multiply
  - 128-bit and 256-bit product mul/add
- CVMSEG
  - CVMSEG LM = Dcache partition can be a local memory / scratchpad
    - Variable and dynamic size from zero to 54 cache blocks (up to 6912B)
  - CVMSEG I/O = IOBDMA operations = I/O prefetches into CVMSEG LM
  - Separate kernel/debug, supervisor, user access
- XKPHYS access for supervisor/user applications
  - separate enables for memory and I/O spaces
  - separate enables for supervisor and user
- Read and Execute Inhibit TLB support

4.1.3 Full Privileged Architecture (i.e. Coprocessor 0) Support

- 64-entry TLB
  - Up to 128 pages
  - Page sizes from 4KB up to 256MB (1KB not supported)
- Cavium-specific Read Inhibit and Execute Inhibit extensions to PageGrain and EntryLo* COP0 registers
- SEGBITS = 49
- PABITS = 49
- Virtual Icache
- Soft-reset
  - Core-selected CIU_PP_RST[RST,RST0] (core must be forced idle)
  - Chip-wide CIU_SOFT_RST[SOFT_RST]
  - Core-local EJTAG TAP soft-reset (ECR[PrRst])
- Non-maskable interrupts
  - Core-selected CIU_NMI[NMI]
  - CIU watchdog timer expiration for the core
- Interrupts
  - Three interrupt input wires per core (Cause[IP4, IP3, IP2]) from CIU
    - CIU combines all interrupt sources separately for each
    - CIU has per-core mailbox registers for inter-core interrupts
  - No external interrupt controller (EIC)
  - Vectored interrupts not implemented
- Supervisor mode implemented
- Two performance counters
- Two watchpoints
  - One instruction-only, one load/store-only
- Hardware-automatic Icache and Dcache repair (by invalidation) on Icache and Dcache errors
  - Separate Icache and Dcache error registers for diagnostic purposes.

- HWREna support for Cavium-specific RDHWR additions
- No GPR shadow sets
- No user-mode endian reversal
- No reduced-power modes
- MDMX ASE not implemented
- SmartMIPS ASE not implemented
- MIPS16e not implemented
- Trace Logic not implemented

4.1.4 Full EJTAG Version 3.10 Support

- Single-step implemented
- EJTAG DSEG implemented
  - DRSEG
  - DMSEG accesses pass through EJTAG TAP controller
- Four Instruction Breakpoints
  - ASID compare
- Four Data Breakpoints
  - Both load and store value compare with byte lanes implemented
  - ASID compare
- Debug interrupts (DINT) implemented
  - Core-selected CIU_DINT[DINT]
  - Debug interrupts from the EJTAG TAP controller
- Multicore Debug Support
  - Immediate multicore stop on EJTAG breakpoint match
  - Dedicated multicore communication path for low-impact and very fast debugging
  - Global (CIU_GSTOP[GSTOP]) and per-core debug mode counter/ watchdog stop conditions
- EJTAG PC Sampling Implemented
- Full EJTAG TAP controller support
  - Special EJTAG chip pin interface
  - One EJTAG TAP controller per core - EJTAG TAPs from different cores link serially onto the EJTAG chip pin interface
  - DMSEG implemented
  - EJTAGBOOT implemented (for memory-less boot)
  - Core reset support (ECR[PrRst])
  - Peripheral reset not supported
- Imprecise debug data breakpoint on load value compare, otherwise all debug data breakpoints are precise.
- No imprecise bus errors
- No imprecise machine-checks
- No imprecise cache errors
- Internal bus clock never stops
- No soft-reset masking

4.2 cnMIPS Core Non-Privileged State

Table 4–2 details the non-privileged state of the cnMIPS cores.

NOTE: The cores do not implement floating-point. Otherwise, all instructions are implemented.

Table 4–2 CPU Visible State Resident in each cnMIPS Core

Register	Implemented/ Not Implemented	Comments
32 general-purpose registers	Implemented	
HI and LO registers	Implemented	
PC	Implemented	
FPU registers	Not implemented	Hardware floating point is not supported by the cnMIPS cores
MPL0, MPL1, MPL2	Cavium specific	64-bit multiplier registers used by Cavium Networks' multiplier additions. Refer to Section 4.15 on page 194 for more information. Not present when CvmCtl[NOMUL] = 1.
P0, P1, P2	Cavium specific	64-bit product registers used by Cavium Networks' multiplier additions. Refer to Section 4.15 on page 194 for more information. **NOTE:** P0, P1, and P2 are made unpredictable by any of the DDIV, DDIVU, DIV, DIVU, MUL, DMUL, DMULT, MADD, MADDU, MSUB, OR MSUBU instructions. You must save P0, P1, and P2 before any of these instructions are executed when a context switch is necessary. **NOTE:** Not present when CvmCtl[NOMUL] = 1.
CRCIV<31:0>	Cavium specific	IV for CRC32 engine
CRCPOLY<31:0>	Cavium specific	Polynomial for CRC32 engine
CRCLEN<3:0>	Cavium specific	LEN (number of bytes to add) for CRC32 engine
GFMMUL[1:0]<63:0>	Cavium specific	Galois Field Multiplier multiplier **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
GFMRESINP[1:0]<63:0>	Cavium specific	Galois Field Multiplier result/input **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
GFMPOLY<15:0>	Cavium specific	Galois Field Multiplier polynomial **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
HASHDAT[14:0]<63:0>	Cavium specific	First part of input data for 512/1024-bit hash **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
HASHIV[7:0]<63:0>	Cavium specific	Hash IV **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
3DESKEY[2:0]<63:0>	Cavium specific	3DES key. 3DESKEY[1:0] also is the KASUMI key when CvmCtl[[KASUMI] = 1. **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
3DESIV<63:0>	Cavium specific	3DES IV **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
3DESRESULT<63:0>	Cavium specific	3DES result. Also the KASUMI result when CvmCtl[[KASUMI] = 1. **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
AESKEY[3:0]<63:0>	Cavium specific	AES key **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.

Table 4–2 CPU Visible State Resident in each cnMIPS Core (Continued)

Register	Implemented/ Not Implemented	Comments
AESKEYLEN<1:0>	Cavium specific	AES key length indicator **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
AESIV[1:0]<63:0>	Cavium specific	AES IV **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
AESRESINP[1:0]<63:0>	Cavium specific	AES result/input **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
CVMSEG LM	Cavium specific	Local Scratchpad Memory

4.3 Cavium-Specific Instruction Summary

Table 4–3 lists all Cavium Networks-specific instructions in the cnMIPS core.

Table 4–3 Summary of Cavium Networks-specific Instructions

Instruction	Comments
BADDU	Unsigned byte add. Refer to Appendix A.
BBIT0, BBIT032, BBIT1, BBIT132	Bit-test branches. These instructions consume the MIPS LDC2, LWC2, SDC2, and SWC2 major opcodes. Refer to Appendix A.
CACHE	Cache manipulation instruction. Refer to Appendix A.
DMFC2, DMTC2 3DES: CVM_MF_3DES_IV CVM_MF_3DES_KEY CVM_MF_3DES_RESULT CVM_MT_3DES_DEC CVM_MT_3DES_DEC_CBC CVM_MT_3DES_ENC CVM_MT_3DES_ENC_CBC CVM_MT_3DES_IV CVM_MT_3DES_KEY CVM_MT_3DES_RESULT	Instructions to use the in-core 3DES Coprocessor. Refer to Appendix A. **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
DMFC2, DMTC2 AES: CVM_MF_AES_INP0 CVM_MF_AES_IV CVM_MF_AES_KEY CVM_MF_AES_KEYLENGTH CVM_MF_AES_RESINP CVM_MT_AES_DEC_CBC0 CVM_MT_AES_DEC_CBC1 CVM_MT_AES_DEC0 CVM_MT_AES_DEC1 CVM_MT_AES_ENC_CBC0 CVM_MT_AES_ENC_CBC1 CVM_MT_AES_ENC0 CVM_MT_AES_ENC1 CVM_MT_AES_IV CVM_MT_AES_KEY CVM_MT_AES_KEYLENGTH CVM_MT_AES_RESINP	Instructions to use the in-core AES Coprocessor. Refer to Appendix A. **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.

Table 4–3 Summary of Cavium Networks-specific Instructions (Continued)

Instruction	Comments
DMFC2, DMTC2 CRC: CVM_MF_CRC_IV CVM_MF_CRC_IV_REFLECT CVM_MF_CRC_LEN CVM_MF_CRC_POLYNOMIAL CVM_MT_CRC_BYTE CVM_MT_CRC_BYTE_REFLECT CVM_MT_CRC_DWORD CVM_MT_CRC_DWORD_REFLECT CVM_MT_CRC_HALF CVM_MT_CRC_HALF_REFLECT CVM_MT_CRC_IV CVM_MT_CRC_IV_REFLECT CVM_MT_CRC_LEN CVM_MT_CRC_POLYNOMIAL CVM_MT_CRC_POLYNOMIAL_REFLECT CVM_MT_CRC_VAR CVM_MT_CRC_VAR_REFLECT CVM_MT_CRC_WORD CVM_MT_CRC_WORD_REFLECT	Instructions to use the in-core CRC Coprocessor. Refer to Appendix A.
DMFC2, DMTC2 GFM CVM_MF_GFM_MUL CVM_MF_GFM_RESINP CVM_MF_GFM_POLY CVM_MT_GFM_MUL CVM_MT_GFM_RESINP CVM_MT_GFM_XOR0 CVM_MT_GFM_XORMUL1 CVM_MT_GFM_POLY	Instructions to use the Galois Field Multiplier. Refer to Appendix A. **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
DMFC2, DMTC2 HSH CVM_MF_HSH_DAT CVM_MF_HSH_DATW CVM_MF_HSH_IV CVM_MF_HSH_IVW CVM_MT_HSH_DAT CVM_MT_HSH_DATW CVM_MT_HSH_IV CVM_MT_HSH_IVW CVM_MT_HSH_STARTMD5 CVM_MT_HSH_STARTSHA CVM_MT_HSH_STARTSHA256 CVM_MT_HSH_STARTSHA512	Instructions to use the in-core HSH Coprocessor. (MD5/SHA-1). Refer to Appendix A. **NOTE:** Not present when CvmCtl[NOCRYPTO] = 1.
DMFC2, DMTC2 KASUMI: CVM_MF_KAS_KEY CVM_MF_KAS_RESULT CVM_MT_KAS_ENC_CBC CVM_MT_KAS_ENC CVM_MT_KAS_KEY CVM_MT_KAS_RESULT	Instructions to use the in-core KASUMI coprocessor. Refer to Appendix A. **NOTE:** Not present when CvmCtl[KASUMI] = 0.
DMUL	Register-direct 64-bit multiply. Refer to Appendix A.
EXTS, EXTS32, CINS, CINS32	Signed-bit field extract and clear/insert instructions. Refer to Appendix A.
MTM0, MTM1, MTM2, MTP0, MTP1, MTP2	Instructions to move data to/from Cavium-specific multiplier registers. Refer to Appendix A. **NOTE:** Not present when CvmCtl[NOMUL] = 1.

Table 4–3 Summary of Cavium Networks-specific Instructions (Continued)

Instruction	Comments
PREF 4, PREF 5 PREF 28 PREF 29 PREF 30	Prefetch into L1, do not put the block in L2 Prefetch into L2, do not put the block in L1 Don't-write-back (block locations are unpredictable until stored to). Prepare for store (block locations are unpredictable until stored to). Refer to Appendix A.
POP, DPOP	Count the number of ones in a 32-bit (POP) or a 64-bit (DPOP) variable. Refer to Appendix A.
RDHWR 31, RDHWR 30	64-bit cycle counter (31). Fast POW switch access (30). Refer to Appendix A.
SAA, SAAD	32-bit and 64-bit store atomic add instructions. Refer to Appendix A.
SEQ, SEQI, SNE, SNEI	SEQ and SNE have functionality similar to SLT. Refer to Appendix A.
SYNCIOBDMA, SYNCS, SYNCW, SYNCWS	Memory reference ordering instructions. Refer to Appendix A.
ULD, ULW, USD, USW	Unaligned Load / Store instructions. Included when CvmCtl[USEUN] is set. Refer to Appendix A.
V3MULU, VMULU, VMM0	Large multiply instructions. Refer to Appendix A. **NOTE:** Not present when CvmCtl[NOMUL] = 1.

4.4 cnMIPS Core Instruction Set Summary

Table 4–4 through Table 4–23 list the complete cnMIPS core instruction set.

Table 4–4 CPU Arithmetic Instructions

Instruction	Implemented/ Not Implemented	Comments
ADD, ADDI, DADD, DADDI, DSUB, SUB	Implemented	
ADDU, ADDIU, DADDU, DADDIU, DSUBU, SUBU	Implemented	
BADDU	Implemented (Cavium specific)	Unsigned byte add. Refer to Appendix A.
CLO, CLZ, DCLO, DCLZ	Implemented	
DDIV, DDIVU, DIV, DIVU	Implemented	These instructions make the Cavium large multiply registers P0, P1, and P2 unpredictable.
DMUL	Implemented (Cavium specific)	Register-direct 64-bit multiply. Refer to Appendix A. This instruction makes the Cavium large multiply registers P0, P1, and P2 unpredictable.
DMULT, DMULTU, MULT, MULTU	Implemented	These instructions make the Cavium large multiply registers P0, P1, and P2 unpredictable.
DPOP	Implemented (Cavium specific)	64-bit population count. Refer to Appendix A.
MUL, MADD, MADDU, MSUB, MSUBU	Implemented	These instructions make the Cavium large multiply registers P0, P1, and P2 unpredictable.
POP	Implemented (Cavium specific)	32-bit population count. Refer to Appendix A.
SEB, SEH	Implemented	
SEQ, SEQI, SNE, SNEI	Implemented (Cavium specific)	Equal/not equal comparison. Refer to Appendix A.
SLT, SLTI, SLTIU, SLTU	Implemented	

Table 4–4 CPU Arithmetic Instructions (Continued)

Instruction	Implemented/ Not Implemented	Comments
VMM0, V3MULU, VMULU	Implemented (Cavium specific)	Large multiplies. Refer to Appendix A.

Table 4–5 CPU Branch and Jump Instructions

Instruction	Implemented/ Not Implemented	Comments
BBIT0, BBIT032, BBIT1, BBIT132	Implemented (Cavium specific)	New bit test instructions not in the MIPS64 architecture. These instructions consume the MIPS LDC2, LWC2, SDC2, and SWC2 major opcodes. They are not, however, considered coprocessor instructions. Refer to Appendix A.
BEQ (B), BGEZ, BGTZ, BGEZAL (BAL), BLTZAL, BLEZ, BLTZ, BNE, J, JR, JAL, JALR.HB, JR.HB,	Implemented	
BEQL, BGEZALL, BGEZL, BGTZL, BLEZL, BLTZALL, BLTZL, BNEL	Implemented	Deprecated. Software is strongly encouraged to avoid use of the Branch likely instructions.

Table 4–6 CPU Instruction Control Instructions

Instruction	Implemented/ Not Implemented	Comments
NOP, SSNOP, EHB	Implemented	

Table 4–7 CPU Load, Store, and Memory Control Instructions

Instruction	Implemented/ Not Implemented	Comments
LB, LBU, SB	Implemented	
LD, LH, LHU, LW, LWU, SD, SH, SW	Implemented as specified in the MIPS specifications when CvmCtl[REPUN] is clear. Unaligned memory references are automatically completed by hardware when CvmCtl[REPUN] is set.	When CvmCtl[REPUN] is set, unaligned memory addresses are not trapped, they are instead completed by the hardware. The hardware cost is ~6 extra cycles when the different bytes required by the operation cross an aligned 64-bit boundary, so the ULD, ULW, USD, and USW instructions may be advantageous in this case. Refer to Appendix A for descriptions of the ULD, ULW, USD, and USW instructions.
LL, LLD, SC, SCD	Implemented	Refer to "cnMIPS Core Load-Linked / Store-Conditional" on page 200.
LDL, LWL, SDL, SWL LDR, LWR, SDR, SWR	Implemented when CvmCtl[USEUN] is clear.	The LDL, LWL, SDL, and SWL instruction opcodes are otherwise the ULD, ULW, USD, and USW instructions or NOPs when CvmCtl[USEUN] is set. Refer to Appendix A for descriptions of the ULD, ULW, USD, and USW instructions.
PREF	Implemented (Cavium specific)	Prefetch instructions. Refer to Appendix A.
SAA, SAAD	Implemented (Cavium specific)	32-bit and 64-bit store atomic add instructions. Refer to Appendix A.
SYNC	Implemented	
SYNCI	Implemented	Flushes the entire Icache. SYNCI instructions never match a watchpoint/breakpoint on CN50XX.

Table 4–7 CPU Load, Store, and Memory Control Instructions (Continued)

Instruction	Implemented/ Not Implemented	Comments
SYNCIOBDMA, SYNCS, SYNCW, SYNCWS	Implemented (Cavium specific)	SYNC instructions not included in the MIPS ISA. Refer to Appendix A.
ULD, ULW, USD, USW	Cavium-specific instructions that are available when CvmCtl[USEUN] is set	Unaligned Load and Store operations (ULW, USW = 32-bit, ULD, USD = 64-bit). Instructions included when CvmCtl[USEUN] is set. Consume the LDL, LWL, SDL, SWL, LDR, LWR, SDR, SWR opcodes whenCvmCtl[USEUN] is set. Refer to Appendix A.

Table 4–8 CPU Logical Instructions

Instruction	Implemented/ Not Implemented	Comments
AND, ANDI, LUI, NOR, OR, ORI, XOR, XORI	Implemented	

Table 4–9 CPU Insert/Extract Instructions

Instruction	Implemented/ Not Implemented	Comments
DEXT, DEXTM, DEXTU, DINS, DINSM, DINSU, DSBH, DSHD, EXT, INS, WSBH	Implemented	
EXTS, EXTS32, CINS, CINS32	Implemented (Cavium specific)	Signed bit field extract and clear/insert instructions not included in the MIPS64 architecture. Refer to Appendix A.

Table 4–10 CPU Move Instructions

Instruction	Implemented/ Not Implemented	Comments
MFHI, MFLO, MTHI, MTLO, MOVN, MOVZ	Implemented	
MOVF, MOVT	Not implemented	cnMIPS cores do not implement floating point.
MTM0, MTM1, MTM2, MTP0, MTP1, MTP2,	Implemented (Cavium specific)	Instructions to move data to/from Cavium-specific multiplier registers. Refer to Appendix A.
RDHWR	Implemented (Cavium specific)	RDHWR 30, 31 added beyond the RDHWR 0, 1, 2, 3 already in MIPS Specifications. Refer to Appendix A.

Table 4–11 CPU Shift Instructions

Instruction	Implemented/ Not Implemented	Comments
DROTR, DROTR32, DROTRV, ROTR, ROTRV DSLL, DSLL32, DSLLV, DSRA, DSRA32, DSRAV, DSRL, DSRL32, DSRLV, SLL, SLLV, SRA, SRAV, SRL, SRLV	Implemented	

Table 4–12 CPU Trap Instructions

Instruction	Implemented/ Not Implemented	Comments
BREAK, SYSCALL, TEQ, TEQI, TGE, TGEI, TGEIU, TGEU, TLT, TLTI, TLTIU, TLTU, TNE, TNEI	Implemented	

Table 4–13 FPU Arithmetic Instructions

Instruction	Implemented/ Not Implemented	Comments
All	Not Implemented	cnMIPS cores do not implement floating point.

Table 4–14 FPU Branch Instructions

Instruction	Implemented/ Not Implemented	Comments
All	Not Implemented	cnMIPS cores do not implement floating point.

Table 4–15 FPU Compare Instructions

Instruction	Implemented/ Not Implemented	Comments
All	Not Implemented	cnMIPS cores do not implement floating point.

Table 4–16 FPU Convert Instructions

Instruction	Implemented/ Not Implemented	Comments
All	Not Implemented	cnMIPS cores do not implement floating point.

Table 4–17 FPU Load, Store, and Memory Control Instructions

Instruction	Implemented/ Not Implemented	Comments
All	Not Implemented	cnMIPS cores do not implement floating point.

Table 4–18 FPU Move Instructions

Instruction	Implemented/ Not Implemented	Comments
All	Not Implemented	cnMIPS cores do not implement floating point.

Table 4–19 Coprocessor Branch Instructions

Instruction	Implemented/ Not Implemented	Comments
BC2F, BC2T, BC2FL, BC2TL	Not implemented	

Table 4–20 Coprocessor Execute Instructions

Instruction	Implemented/ Not Implemented	Comments
COP2	Not Implemented	

Table 4–21 Coprocessor Load and Store Instructions

Instruction	Implemented/ Not Implemented	Comments
LDC2, LWC2, SDC2, SWC2	Not Implemented	These opcodes are consumed by the Cavium-specific BBIT0, BBIT032, BBIT1, BBIT132 instructions. Refer to Appendix A.

Table 4–22 Coprocessor Move Instructions

Instruction	Implemented/ Not Implemented	Comments
CFC2, CTC2, MFC2, MTC2, MFHC2, MTHC2	Not Implemented	
DMFC2, DMTC2	Implemented (Cavium specific)	Refer to Appendix A.

Table 4–23 Privileged Instructions

Instruction	Implemented/ Not Implemented	Comments
CACHE	Implemented (Cavium specific)	Cache manipulation instruction, addresses never-match watchpoints/ breakpoints on CN50XX. Refer to Appendix A.
ERET, DI, EI, DMFC0, DMTC0, MFC0, MTC0	Implemented	
RDPGPR, WRPGPR	Implemented	Implemented as a register-to-register copy. The cnMIPS cores have no shadow register set, so both rd and rt refer to the main register file.
TLBP, TLBR, TLBWI, TLBWR	Implemented	A machine-check may occur during a TLB write. This is what will happen to the TLB contents when a machine-check is taken: The even entry will be written, the VPN side and odd entry will be unchanged.
WAIT	Implemented	CvmCtl[DISWAIT] changes WAIT into a NOP.

Table 4–24 EJTAG Instructions

Instruction	Implemented/ Not Implemented	Comments
DERET, SDBBP	Implemented	See “cnMIPS Core Hardware Debug Features” on page 197.

4.5 cnMIPS Core Virtual Addresses and CVMSEG

SEGBITS = 49, as per the MIPS specifications.

When CvmMemCtl Register[CVMK/S/U] is set for the appropriate mode (Kernel, Supervisor, or User), virtual addresses for loads/stores in the KSEG3 range `0xFFFF FFFF FFFF 8000` to `0xFFFF FFFF FFFF BFFF` are called CVMSEG references and are treated specially by the cnMIPS cores. Instruction references to these addresses are always treated as normal KSEG3 references in the cores. When CvmMemCtl[CVM*] is clear for the appropriate mode, these CVMSEG addresses act as normal MIPS-defined KSEG3 addresses.

CVMSEG has two parts:

```
CVMSEG LM = 0xFFFF FFFF FFFF 8000 to 0xFFFF FFFF FFFF 9FFF
CVMSEG IO = 0xFFFF FFFF FFFF A000 to 0xFFFF FFFF FFFF BFFF
```

CVMSEG LM is a segment that accesses portions of the Dcache as a local memory; the larger CVMSEG is, the smaller the cache is. CvmMemCtl[LMEMSZ] selects the size of CVMSEG LM, which is in cache blocks. CvmMemCtl[LMEMSZ] can legally range from zero to 54 cache blocks (i.e. CVMSEG LM is between zero and 6192 bytes). The legal CVMSEG LM addresses (when CvmMemCtl[LMEMSZ] is larger than zero) start at virtual address `0xFFFF FFFF FFFF 8000` and may increase up to the maximum possible legal CVMSEG LM virtual address `0xFFFF FFFF FFFF 9AFF` (which is only legal when CvmMemCtl[LMEMSZ] is 54). CVMSEG LM references above the range allocated by CvmMemCtl[LMEMSZ] (but at `0xFFFF FFFF FFFF 9FFF` or below) cause an address error, but stores to these illegal addresses may not be stopped by the hardware, so can cause cache corruption.

CVMSEG I/O is a segment that has only one legal address that does not give address errors:

```
0xFFFF FFFF FFFF A200
```

This address can only be referenced by SD instructions. SD instructions to this CVMSEG I/O address cause the core hardware to issue IOBDMA commands. IOBDMA commands return data from I/O bus devices into selected CVMSEG LM locations.

Store instructions to CVMSEG LM execute so early in the processor pipeline that the core implementation cannot stop them in certain unusual circumstances, causing anomalies. One such anomaly is that cache corruption can occur when an address error occurs solely due to exceeding the CvmMemCtl[LMEMSZ] limit on CVMSEG LM size. As “address error on data access” in Table 4–36 indicates, these erroneous CVMSEG LM stores can corrupt other areas of the Dcache. Either these erroneous stores must not be present in proper applications or, if necessary, software can invalidate the Dcache on these CVMSEG LM store range address errors to reduce the likelihood of problems. Another anomaly is the early execution of CVMSEG LM stores due to implementation-specific traps related to the presence of virtual-to-physical aliases in the cache. When these aliases are present near when an interrupt or other asynchronous exception occurs, in rare circumstances it is possible that the core will have executed the CVMSEG LM store, though the exception program counters indicate that the store has not yet executed. This early CVMSEG LM store execution anomaly is not generally a problem, as there will never be another load or store between the exception PC and the store to CVMSEG LM when the CVMSEG LM store executes early. When the software returns from the exception in this case, the store will re-write CVMSEG LM and the software will likely not notice the anomaly. Nevertheless, it is possible to observe the early CVMSEG LM stores when debugging or in other circumstances.

4.6 Physical Addresses

The physical address is described in Figure 4–2. The number of physical address bits (PABITS) is 49, as per the MIPS specifications.

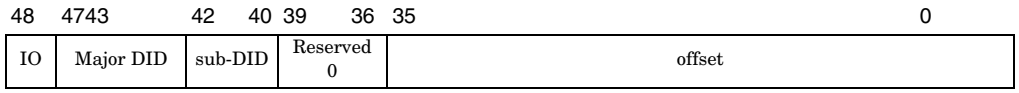


Figure 4–2 49-bit Physical Address Format

- **IO** - I/O bit
- **Major did** - Directs the request to the correct hardware block.
- **subdid** - Directs the request within the hardware block.
- **offset** - Either L2/DRAM addresses or I/O Bus device addresses.

NOTE: CN50XX does not use the MIPS-defined cache-coherence attributes to determine cacheability. Only the physical addresses determine cacheability. Except for physical addresses in the range 0x1000 0000 to 0x1FFF FFFF, CN50XX caches all addresses that have I/O bit clear. CN50XX does not cache an address that either has the I/O bit set or is in the range
0x1000 0000 to 0x1FFF FFFF. These addresses are called I/O space addresses.

NOTE: The physical addressing implementation does not support uncached DRAM addressing or cached I/O space accesses.

Physical Address Detail:

Table 4–25 cnMIPS Core Physical Addresses

Physical Addresses	Mem / Bus	Comment
0x0 0000 0000 0000 to 0x0 0000 0FFF FFFF	DR0 DRAM	Cached (first 256 MB of DRAM)
0x0 0000 1000 0000 to 0x0 0000 1FFF FFFF	Boot Bus	Uncached I/O space. converted to: 0x1 0000 1000 0000 to 0x1 0000 1FFF FFFF
0x0 0000 2000 0000 to 0x0 0003 FFFF FFFF	DR2 DRAM	Cached (all DRAM memory above the first 512 MB)
0x0 0004 1000 0000 to 0x0 0004 1FFF FFFF	DR1 DRAM	Cached (second 256 MB of DRAM)
0x1 0000 0000 0000 to 0x1 0000 FFFF FFFF	Boot bus	Uncached (I/O space)
0x1 0700 0000 0000 to 0x1 0700 0000 08FF	CIU and GPIO NCB type CSRs	Uncached (I/O space)
0x1 1800 0000 0000 to 0x1 1800 0000 1FFF	MIO BOOT, LED, FUS, TWSI, UART0, UART1, SMI RML type CSRs	Uncached (I/O space)
0x1 1800 0800 0000 to 0x1 1800 0800 1FFF	GMX0 RML type CSRs	Uncached (I/O space)
0x1 1800 2800 0000 to 0x1 1800 2800 01FF	FPA RML type CSRs	Uncached (I/O space)
0x1 1800 4000 0000 to 0x1 1800 4000 000F	RNM RML type CSRs	Uncached (I/O space)

Table 4–25 cnMIPS Core Physical Addresses (Continued)

Physical Addresses	Mem / Bus	Comment
0x1 1800 5000 0000 to 0x1 1800 5000 1FFF	PKO RML type CSRs	Uncached (I/O space)
0x1 1800 5800 0000 to 0x1 1800 5800 1FFF	TIM RML type CSRs	Uncached (I/O space)
0x1 1800 6800 0000 to 0x1 1800 6800 1FFF	USBC RML CSRs	Uncached (I/O space)
0x1 1800 8000 0000 to 0x1 1800 8000 07FF	L2C RML type CSRs	Uncached (I/O space)
0x1 1800 8800 0000 to 0x1 1800 8800 007F	LMC RML type CSRs	Uncached (I/O space)
0x1 1800 A000 0000 to 0x1 1800 A000 1FFF	PIP RML type CSRs	Uncached (I/O space)
0x1 1800 B000 0000 to 0x1 1800 B000 03FF	ASX0 RML type CSRs	Uncached (I/O space)
0x1 1800 F000 0000 to 0x1 1800 F000 07FF	IOB RML type CSRs	Uncached (I/O space)
0x1 1900 0000 0000 to 0x1 190F FFFF FFFF	PCI Bus Config/IACK/Special space	Uncached (I/O space)
0x1 1A00 0000 0000 to 0x1 1A0F FFFF FFFF	PCI Bus I/O space	Uncached (I/O space)
0x1 1B00 0000 0000 to 0x1 1B0F FFFF FFFF	PCI Bus Memory space (subdid 3)	Uncached (I/O space)
0x1 1C00 0000 0000 to 0x1 1C0F FFFF FFFF	PCI Bus Memory space (subdid 4)	Uncached (I/O space)
0x1 1D00 0000 0000 to 0x1 1D0F FFFF FFFF	PCI Bus Memory space (subdid 5)	Uncached (I/O space)
0x1 1E00 0000 0000 to 0x1 1E0F FFFF FFFF	PCI Bus Memory space (subdid 6)	Uncached (I/O space)
0x1 1F00 0000 0000 to 0x1 1F0F FFFF FFFF	NPI NCB type CSRs, doorbells	Uncached (I/O space)
0x1 2800 0000 0000 to 0x1 280F FFFF FFFF	FPA Pool 0 Allocate/Free operations	Uncached (I/O space)
0x1 2900 0000 0000 to 0x1 290F FFFF FFFF	FPA Pool 1 Allocate/Free operations	Uncached (I/O space)
0x1 2A00 0000 0000 to 0x1 2A0F FFFF FFFF	FPA Pool 2 Allocate/Free operations	Uncached (I/O space)
0x1 2B00 0000 0000 to 0x1 2B0F FFFF FFFF	FPA Pool 3 Allocate/Free operations	Uncached (I/O space)
0x1 2C00 0000 0000 to 0x1 2C0F FFFF FFFF	FPA Pool 4 Allocate/Free operations	Uncached (I/O space)
0x1 2D00 0000 0000 to 0x1 2D0F FFFF FFFF	FPA Pool 5 Allocate/Free operations	Uncached (I/O space)
0x1 2E00 0000 0000 to 0x1 2E0F FFFF FFFF	FPA Pool 6 Allocate/Free operations	Uncached (I/O space)
0x1 2F00 0000 0000 to 0x1 2F0F FFFF FFFF	FPA Pool 7 Allocate/Free operations	Uncached (I/O space)
0x1 4000 0000 0000 to 0x1 4000 0000 07FF	RNG Load/IOBDMA operations	Uncached (I/O space)
0x1 4F00 0000 0000 to 0x1 4F00 0000 07FF	IPD NCB type CSRs	Uncached (I/O space)
0x1 5200 0000 0000 to 0x1 5200 0003 FFFF	PKO doorbell store operations	Uncached (I/O space)

Table 4–25 cnMIPS Core Physical Addresses (Continued)

Physical Addresses	Mem / Bus	Comment
0x1 6000 0000 0000 to 0x1 600F FFFF FFFF	POW getwork load/iobdma operations, store work operations	Uncached (I/O space)
0x1 6100 0000 0000 to 0x1 610F FFFF FFFF	POW status load operations, store work operations	Uncached (I/O space)
0x1 6200 0000 0000 to 0x1 6200 0000 FFFF	POW memory load operations	Uncached (I/O space)
0x1 6300 0000 0000 to 0x1 630F FFFF FFFF	POW index loads, store operations	Uncached (I/O space)
0x1 6300 0000 0000 to 0x1 6300 0000 0007	POW NullRd load operations	Uncached (I/O space)
0x1 6700 0000 0000 to 0x1 6700 0000 03FF	POW NCB type CSRs	Uncached (I/O space)
0x1 6F00 0000 0000 to 0x1 6F00 0000 0FFF	USBN NCB type CSRs	Uncached (I/O space)
0x1 F000 0000 0000 to 0x1 F00F FFFF FFFF	FAU operations	Uncached (I/O space)

4.7 IOBDMA Operations

The cnMIPS core issues an IOBDMA request when the store that references the CVMSEG I/O address issues, and does not wait for the return data to be deposited into the selected CVMSEG LM addresses before executing subsequent instructions. SYNCIOBDMA or SYNC instructions stall subsequent instructions until the CVMSEG LM addresses are updated for all prior IOBDMA operations, however. These IOBDMA operations are “I/O prefetches” since IOBDMA operations allow multiple outstanding “loads” to I/O Bus devices and also can reference more than 64-bits.

See Figure 4–3. The 64-bit store data format for a CVMSEG I/O store access is:

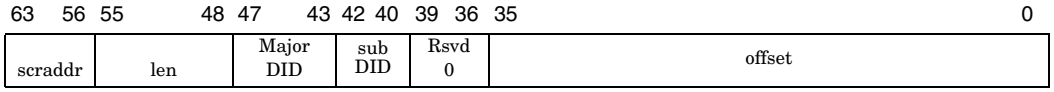


Figure 4–3 64-bit IOBDMA Store Data Format

- **scraddr** - A (64-bit word) CVMSEG starting location where the core puts the result. The effective CVMSEG LM is the combination of the 8-bit scradr field in the store data of the IOBDMA together with the CvmMemCtl[IOBDMASCRMSB] extension bits.

  effective_scraddr = scraddr + (CvmMemCtl[IOBDMASCRMSB] << 8)

  The corresponding CVMSEG LM is the combination starting address is:

  0xFFFF FFFF FFFF 8000 + (8 × effective_scraddr).

NOTE: IOBDMA operations can start only in the first 2KB of CVMSEG LM, unless CvmMemCtl[IOBDMASCRMSB] is non-zero.

- **len** - The number of 64-bit words in the result. The results are placed sequentially in CVMSEG LM from the starting address indicated by scraddr.
- **Major did** - Directs the request to the correct hardware block, as with a normal physical address.
- **subdid** - Directs the request within the hardware block.
- **offset** - Interpreted by the hardware on the I/O bus as with offset for a normal physical address.

A CVMSEG I/O store causes an I/O bus transaction using the did and offset. This I/O bus transaction is very similar to the transaction created to service an I/O load. The difference is that IOBDMA operations do not stall the core and can return a vector of 64-bit words.

NOTE: IOBDMA operations cannot reference L2/DRAM addresses.

IOBDMA constraints:

- effective_scraddr + len ≤ (CvmMemCtl[LMEMSZ]×16)
- len > 0
- if len is 2 or 3, then (scraddr<1:0>) must not be 3
- if len is 4 or greater, then (scraddr<0>) must be zero

4.8 cnMIPS Core-Memory Reference Ordering

L2/DRAM stores may be buffered by cnMIPS cores until a subsequent SYNC, SYNCS, SYNCW, or SYNCWS instruction issues. The cnMIPS cores may choose to issue these buffered stores into the system memory in any order when there are no SYNC* instructions issued by the cores.

The cnMIPS core's store-buffering architecture is important for correct functionality and for best performance. Proper use of the cnMIPS core's synchronization instructions is essential with the cnMIPS core's store-buffering architecture, and code optimized for cnMIPS cores may be different from code optimized for other architectures.

For an example of the architectural importance of the cnMIPS core's store-buffering behavior, consider lock releases. Lock releases are typically ordinary stores to a lock variable indicating that the lock is no longer held. For best performance with multiple cores, this releasing store should typically be made visible to other cores quickly so that the other cores have the opportunity to obtain the lock soon after it is released.

Without a synchronization instruction (typically, a SYNCW or SYNCWS), the cnMIPS core does not by default make a store visible quickly since it aggressively buffers stores. Thus, for best lock performance on cnMIPS core, you should typically follow the releasing-store instruction with a synchronization instruction. This synchronization instruction is not be needed with many other memory architectures.

The description of the SYNCW instruction in Appendix A details a similar producer/consumer example. Section 4.10 discusses more about the cnMIPS core's store-buffering implementation.

The cnMIPS core differentiates L2/DRAM references based the MIPS cache-coherence attribute attached to the references as follows:

Cache Coherence Attribute	Relevance to L2/DRAM References
0–6	Unmarked
7	Marked

This differentiation is only used for ordering purposes. System performance can be higher when load/store operations are unordered. All L2/DRAM loads/store operations are ordered by SYNC and SYNCW instructions. Marked L2/DRAM loads/stores are not ordered by SYNCS and SYNCWS instructions, so SYNCS and SYNCWS can be higher performance.

- A SYNC guarantees that all prior load and IOBDMA operations complete and all prior (possibly buffered) L2/DRAM store operations are visible to all CN50XX devices before any subsequent load, IOBDMA, or L2/DRAM store operations from the core can issue. A SYNCS is identical to a SYNC, except that it does not order marked L2/DRAM load and store operations.
- A SYNCW guarantees that all prior (possibly buffered) L2/DRAM store operations are visible to all CN50XX devices before any subsequent store operations can issue to the system. SYNCWs are considerably faster than SYNCs since they do not cause the pipeline to stall (subsequent store operations are buffered until the prior store operations are visible to all devices). A SYNCWS is identical to a SYNCW, except that it does not order marked L2/DRAM store operations.

Table 4–26 summarizes what is ordered by the different synchronization instructions available on CN50XX.

Table 4–26 Synchronization Instruction Ordering

Synchronization Instruction	Unmarked L2/DRAM		Marked L2/DRAM		I/O		IOBDMA
	Load	Store	Load	Store	Load	Store	
SYNC	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SYNCS	Yes	Yes	No	No	Yes	Yes	Yes
SYNCW	No	Yes	No	Yes	No	Yes	No
SYNCWS	No	Yes	No	No	No	Yes	No
SYNCIOBDMA	No	No	No	No	No	No	Yes

Note, however, that I/O space store operations are posted, which means that the core considers I/O store operations to be complete when they reach the coherent memory bus (CMB). There is no direct synchronization support available to guarantee that a prior I/O store reached a given device. When software must guarantee that an I/O store completes before proceeding with references, it must follow the store operation with a load operation or IOBDMA/SYNCIOBDMA.

I/O space load/store/IOBDMA requests from a single core to a single device arrive on the CMB and I/O bus in the same order that the instructions issue, so this synchronization often is not needed.

SYNCW/SYNCWS instructions guarantee that all prior I/O-space store/IOBDMA operations are on the CMB before subsequently ordered store/IOBDMA operations become visible to the rest of the system.

SYNC/SYNCS instructions enforce all constraints of a SYNCW/SYNCWS instruction, and also enforces ordering for I/O-space loads. CN50XX guarantees the ordering of all I/O space-load/store/IOBDMA requests from the CMB to the IOB, so SYNCW/SYNCWS and SYNC/SYNCS instructions can be used by software to constrain ordering between I/O bus requests from different cores. cnMIPS core SYNCW/SYNCWS instructions are considerably faster for these I/O space references for the same reason that they are for L2/DRAM references.

SYNCIOBDMA instructions guarantee that prior IOBDMA requests have arrived at the device (and the response has been returned) before any subsequent load/store/IOBDMA can issue. SYNC/SYNCS instructions enforces the same ordering constraints as SYNCIOBDMA, and also orders ordinary loads/stores.

4.9 cnMIPS Core CSR Ordering

Note that the Section 4.8 I/O load/store ordering description refers only to the ordering of the load and store operations on the CMB and I/O bus. In the cnMIPS core design, when the device has CSRs of type RSL, the order of the I/O load/store/IOBDMA operations on the I/O bus may not match the order in which the I/O load/store/IOBDMA operations are received and executed by the device on the I/O bus.

This complicating factor occurs because CN50XX services RSL-type CSR references indirectly off the I/O bus, and this indirect mechanism is much slower than direct accesses via the I/O bus. Later I/O-bus direct references to the device may bypass earlier RSL references. For example, the FPA (described in Chapter 6) has RSL-type CSRs as well as I/O-bus direct FPA operations to free/allocate buffers. Without synchronization, a prior CSR write that configures FPA may be delayed until after a subsequent FPA free operation, causing FPA to malfunction. Software must insert

synchronization to force prior RSL-type writes to complete before issuing the subsequent FPA operations to avoid these malfunctions caused by the hardware re-ordering of RSL CSR references.

Whenever it is necessary for cnMIPS core software to guarantee completion of prior store operations to RSL-type CSRs before subsequent operations, as in the FPA example just described, the core can perform either of the following operations

- Issue a load request to any RSL-type CSR
- Issue an IOBDMA request to any RSL-type CSR followed by a SYNCIOBDMA request

As CN50XX services all load/store/IOBDMA operations to any RSL-type CSR in the same order that they appear on the I/O bus, these load/IOBDMA operations flush all prior store/ load/IOBDMA operations. In general, write operations to RSL-type CSRs should be infrequent in CN50XX, and liberal doses of this RSL synchronization should not cause performance problems.

4.10 cnMIPS Core Write Buffer

The behavior of the write buffers in the cnMIPS cores is important for correct functionality and for best performance. There are 16 write-buffer entries in each core, and each can merge multiple individual stores up to a full 128-byte cache line. A write-buffer entry is marked if its merged stores are all marked.

The write buffer can only stall the instruction pipeline when SYNC/SYNCS instructions execute or when there are no write-buffer entries available. SYNCW/ SYNCWS instructions never stall the instruction pipeline, and are queued and complete in parallel with instruction issue.

A SYNC/SYNCW forces all write-buffer entries to be sent to the CMB. A SYNCS/ SYNCWS does the same, except that marked L2/DRAM write-buffer entries are unmodified. The core does not merge any subsequent store into any write buffer entries that are sent to the CMB, but continues merging into marked L2/DRAM write buffer entries after a SYNCS/SYNCWS.

All sent write-buffer entries commit before any instruction following a SYNC/SYNCS issues.

Following a SYNCW/SYNCWS instruction, the core does not send any subsequently created write-buffer entries to the CMB until after all previously sent write-buffer entries commit. (Note that commit order is, effectively, the store order that is visible to the other cores.)

The write buffer aggressively merges L2/DRAM stores. If an L2/DRAM store is in the same 128-byte (naturally aligned) cache block, the store merges into the existing entry. Otherwise, stores allocate new write buffer entries. L2/DRAM stores normally remain in the write buffer until one of the following occurs:

- A SYNC or SYNCW instruction executes.
- A SYNCS or SYNCWS instruction executes and the write-buffer entry merged at least one unmarked store.

- The write buffer entry times out. CvmMemCtl[WBFLTIME] controls the expiration interval of unmarked L2/DRAM write buffer entries. The expiration interval starts when the write buffer entry is last referenced (unless CvmMemCtl[DISMRGCLRWBTO] is set, when it starts when the write buffer entry is allocated). The timeout interval can be as small as 2K cycles, and as large 500K cycles. (Larger CvmMemCtl[WBFLTIME] values result in larger timeouts.) Marked L2/DRAM write buffer entries always use the maximum timeout interval (unless CvmMemCtl[DISMARKWBLONGTO] is set, when the timeout of marked entries is the same as other entries).
- A load/PREF instruction to the same cache block misses in the (L1) Dcache.
- An SC/SCD instruction executes with the following properties:
  - hits in the data cache
  - to the same cache block
- The write buffer entry is ejected to make space for other write-buffer entries. The core ejects blocks to make space when there are more than CvmMemCtl[WBTHRESH] active write-buffer entries. (Smaller CvmMemCtl[WBTHRESH] values eject blocks sooner, reducing merging, but leave more write-buffer entries available for later use. Larger CvmMemCtl[WBTHRESH] values eject blocks later, increasing merging, but may result in more pipeline stalls because write-buffer entries are not available.)

SC/SCD instructions are treated specially in the write buffer. SC/SCD instructions that miss in the data cache do nothing. SC/SCD instructions that hit evict any matching write-buffer entry that was previously in the write buffer. In addition, all SC/SCD instructions that hit in the data cache create a write-buffer entry that is immediately pushed to the CMB. For many effects, it appears as if the SC/SCD instructions bypass the write buffer.

I/O write operations and IOBDMA operations create write-buffer entries. But they never merge with any other accesses, and are always immediately pushed to the CMB, through the I/O bridge, and onto the I/O buses. For many effects, it appears as if the I/O write operations and IOBDMA operations bypass the write buffer.

Don't-Write-Backs (i.e. PREF 29s, see Appendix A) invalidate earlier write buffer entries that are to the same cache block as the Don't-Write-Back. A Don't-Write-Back also creates a write buffer entry that is immediately pushed to the CMB.

Prepare For Stores (i.e. PREF 30s, see Appendix A) create write buffer entries like ordinary stores, but also make the (either merged into or newly created) write-buffer entry a full cache block write. Note that Prepare For Stores do not update the L1 Dcache, so it is possible for the Dcache copy of the block to become inconsistent with the value that is eventually written into L2/DRAM by the write buffer created for the Prepare For Store. The specification of Prepare For Store defines the value of the bytes in the cache block to be unpredictable after the Prepare For Store, so this is acceptable. If software needs these bytes set to a predictable value (in both the Dcache and the L2/DRAM), it must eventually follow the Prepare For Store with stores to the bytes of interest in the cache block (or with stores to the cache block by another core or device, which invalidates the Dcache).

CvmMemCtl[NOMERGE,ALLSYNCW] are additional debug bits. When CvmMemCtl[NOMERGE] is set, writes do not merge and are forced immediately to the L2. When both CvmMemCtl[NOMERGE,ALLSYNCW] are set, these stores are additionally forced to commit in order.

4.11 cnMIPS Core Coprocessor 0 Privileged Registers

MIPS COP0 is implemented. 64-entry TLB. SEGBITS = PABITS = 49. EJTAG implemented. CN50XX maintains a virtual Icache.

Table 4–27 Coprocessor 0 Register Summary

Register	Implemented/Not implemented	Comments
Index	Implemented	See page 167
Random	Implemented	See page 167
EntryLo0	Implemented	See page 167
EntryLo1	Implemented	See page 167
Context	Implemented	See page 168
ContextConfig	Not Implemented	No SmartMIPS ASE.
PageMask	Implemented	See page 168
PageGrain	Implemented	See page 168
Wired	Implemented	See page 169
HWREna	Implemented	See page 169
BadVAddr	Implemented	See page 169
Count	Implemented	See page 169
EntryHi	Implemented	See page 169
Compare	Implemented	See page 170
Status	Implemented	See page 170
IntCtl	Implemented	See page 171
SRSCtl	Implemented	See page 171
SRSMap	Not Implemented	No shadow register set.
Cause	Implemented	See page 171
EPC	Implemented	See page 172
PRId	Implemented	See page 172
Ebase	Implemented	See page 172
Config	Implemented	See page 172
Config1	Implemented	See page 173
Config2	Implemented	See page 173
Config3	Implemented	See page 174
LLAddr	Not Implemented	
WatchLo	Two Implemented	One instruction-only (reg = 18, sel = 0), one data-only (reg = 18, sel = 1). See page 174
WatchHi	Two Implemented	One instruction-only (reg = 19, sel = 0), one data-only (reg = 19, sel = 1). See page 174
XContext	Implemented	See page 175
Debug	Implemented	See page 176
TraceControl	Not implemented	PDtrace not implemented.
TraceControl2	Not implemented	PDtrace not implemented.
UserTraceData	Not implemented	PDtrace not implemented.
TraceBPC	Not Implemented	PDtrace not implemented.
DEPC	Implemented	See page 176

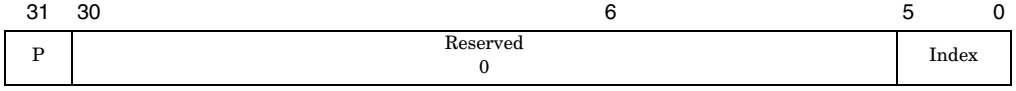
Table 4–27 Coprocessor 0 Register Summary (Continued)

Register	Implemented/Not implemented	Comments
PerfCnt	Two Implemented	Four total registers at reg = 25, one set at sel=0, 1, another set at sel=2, 3. See page 176
ErrCtl	Not Implemented	
CacheErr	Implemented (Cavium-specific)	reg = 27, Sel = 0 implemented to latch Icache error information. See "CacheErr (Icache)" on page 179.
		reg = 27, Sel = 1 implemented to latch Dcache error information. See "CacheErr (Dcache)" on page 179.
TagLo	Implemented (Cavium-specific)	reg = 28, Sel = 0 (Icache) Implemented. See "TagLo Register (Icache)" on page 180.
		reg = 28, Sel = 2 (Dcache) Implemented. See "TagLo Register (Dcache)" on page 180.
DataLo	Implemented (Cavium-specific)	reg = 28, Sel = 1 (Icache) Implemented. See "DataLo Register (Icache)" on page 180.
		reg = 28, Sel = 3 (Dcache) Implemented. See "DataLo Register (Dcache)" on page 181.
TagHi	Implemented (Cavium-specific)	reg = 29, sel = 2 (Dcache) implemented. See "TagHi Register" on page 181.
DataHi	Implemented (Cavium-specific)	reg = 29, sel = 1 (Icache) implemented. See "DataHi Register (Icache)" on page 181
		reg = 29, sel = 3 (Dcache) implemented. See "DataHi Register (Dcache)" on page 182
ErrorEPC	Implemented	See page 178
DESAVE	Implemented	See page 178
CvmCtl	Implemented (Cavium-specific)	Implemented at reg = 9, sel = 7. Controls the Icache and instruction issue. See "CvmCtl Register" on page 182.
CvmMemCtl	Implemented (Cavium-specific)	Implemented at reg = 11, sel = 7. Controls the cores' memory system. See "CvmMemCtl Register" on page 184.
CvmCount	Implemented (Cavium-specific)	Implemented at reg = 9, sel = 6. A 64-bit cycle counter. See page 185 Read by RDHWR 31
MultiCore Debug	Implemented (Cavium-specific)	Implemented at reg = 22, sel = 0. See page 186 Used for multi-core debugging. See "cnMIPS Core Hardware Debug Features" on page 197 for more details.

The remainder of this section details the Coprocessor 0 registers in the cnMIPS™ cores.

Index Register

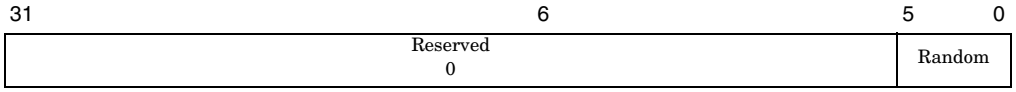
CP0 Register 0, Select 0)



- **P** - Probe Failure. Hardware writes this bit during execution of the TLBP instruction to indicate whether a TLB match occurred.
- **Index** - TLB index. Software writes this field to provide the index to the TLB entry referenced by the TLBR and TLBWI instructions.

Random Register

CP0 Register 1, Select 0

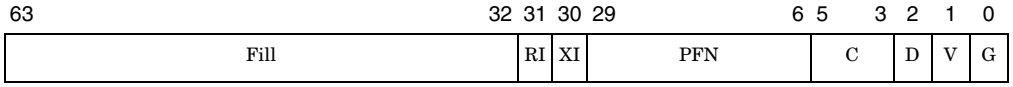


- **Random** - TLB Random Index

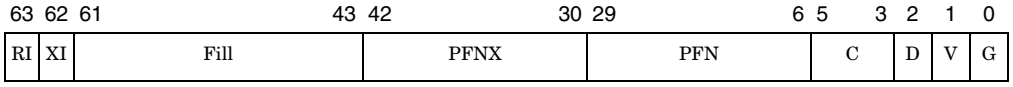
EntryLo0, EntryLo1 Registers

CP0 Registers 2 and 3, Select 0

When PageGrain[ELPA] = 0:



When PageGrain[ELPA] = 1:

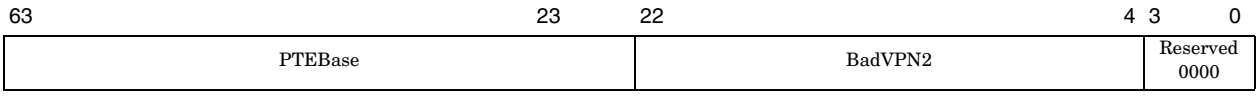


- **RI** - Read inhibit. If this bit is set to 1 in a TLB entry, any load instruction that reads data on the virtual page causes a TLB invalid exception, even if the V (valid) bit is set. This bit is writable only if PageGrain[RIE] = 1. If PageGrain[RIE] = 0, this bit is set to 0 on any write to the EntryLo0/1 register, regardless of the value written.
- **XI** - Execute inhibit. If this bit is set to 1 in a TLB entry, any attempt to fetch an instruction causes a TLB invalid exception, even if the V (valid) bit is set. This bit is writable only if PageGrain[XIE] = 1. If PageGrain[XIE] = 0, this bit is set to 0 on any write to the EntryLo0/1 register, regardless of the value written.
- **Fill** - Ignored on write, return 0x0 on read.
- **PFNX** - When PageGrain[ELPA] = 1, the hardware uses the full address width, with PFNX expanding the PFN field to the full physical address.
  When PageGrain[ELPA] = 0, the PFNX is always written to 0x0 on a mtc0/dmtc0 operation to EntryLo0/1. Otherwise, the hardware uses the full address width in all cases. See “PageGrain Register” on page 168.
  - cnMIPS™ cores implement a 49-bit physical address requiring a 13-bit PFNX field.
  - cnMIPS™ cores do not support 1KB pages.

- **PFN** - Page Frame Number
- **C** - Coherency attribute of page. Refer to Section 4.8.
- **D** - Dirty bit (indicates page is writable)
- **V** - Valid bit
- **G** - Global bit

Context Register

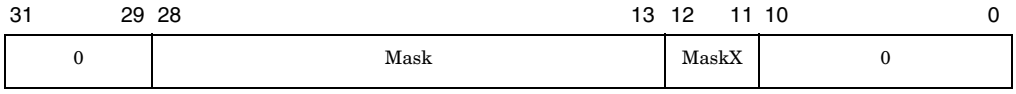
CP0 Register 4, Select 0



- **PTEBase** - This field is for use by the operating system and is normally written with a value that allows the operating system to use the Context Register as a pointer into the current PTE array in memory.
- **BadVPN2** - Bad Virtual Page Number/2 field is written by hardware on a miss.

PageMask Register

CP0 Register 5, Select 0

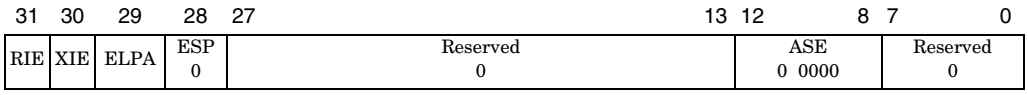


The PageMask register is a read/write register used for reading from and writing to the TLB.

- **Mask** - The Mask field is a bit mask in which a 1 bit indicates that the corresponding bit of the virtual address should not participate in the TLB match. CN50XX supports the full mask range, and supports all page sizes 4KB, 8KB, 16KB, ... 64MB, 128MB, and 256MB.
- **MaskX** - CN50XX does not support 1KB pages, so the MaskX field is not used (writes are ignored, reads as 0).

PageGrain Register

CP0 Register 5, Select 1

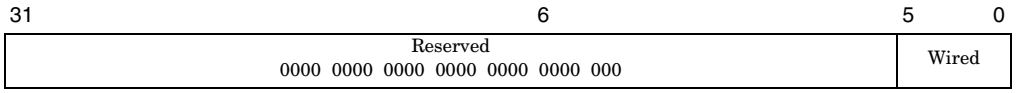


See "EntryLo0, EntryLo1 Registers" on page 167 for description for the only effect of PageGrain[ELPA].

- **RIE -** Read inhibit enable. When set to 1, EntryLo0/1[RI] is enabled. When cleared to 0, EntryLo0/1[RI] is disabled and is not writable by software.
- **XIE -** Execute inhibit enable. When set to 1, EntryLo0/1[XI] is enabled. When cleared to 0, EntryLo0/1[XI] is disabled and is not writable by software.
- **ASE = 0x0** - (ASE not implemented, must be written as 0).
- **ELPA** - Enables support for large physical addresses. As PABITS = 49 for CN50XX, ELPA should normally be enabled on CN50XX.
- **ESP = 0** - (1KB pages not implemented).

Wired Register

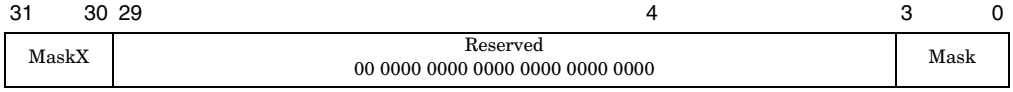
CP0 Register 6, Select 0



- **Wired** - TLB wired boundary

HWREna Register

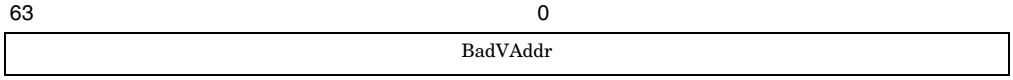
CP0 Register 7, Select 0



- **MaskX** - cnMIPS™ cores include Cavium-specific instruction RDHWR 31, 30, so a MaskX field exists at <31:30> to enable them. Refer to Appendix A for the RDHWR description.
- **Mask** - Each bit in this field enables access by the RDHWR instruction to a particular hardware register

BadVAddr Register

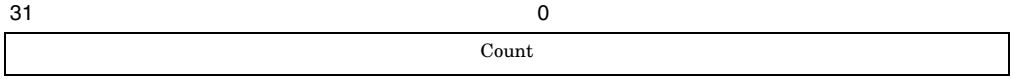
CP0 Register 8, Select 0



- **BadVAddr** - Bad virtual address

Count Register

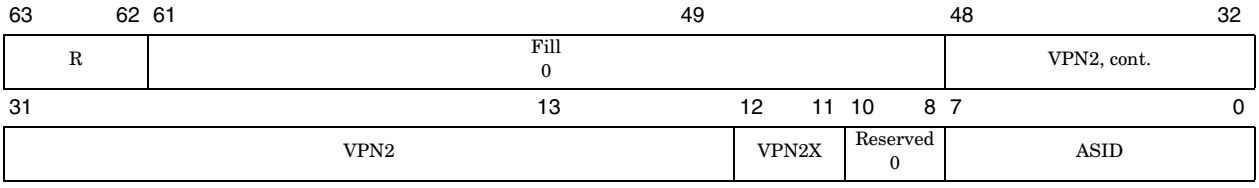
CP0 Register 9, Select 0



The Count register is zeroed during reset and increments by one every cycle.

EntryHi Register

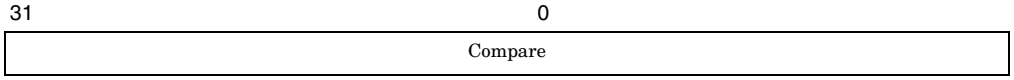
CP0 Register 10, Select 0



- **R** - Virtual memory region.
- **Fill** - Fill bits reserved for expansion of the virtual address space.
- **VPN2** - This field is written by hardware on a TLB exception or on a TLB read.
- **VPN2X** - Not used. (1KB pages not implemented). Must be written with 0, reads return 0.
- **ASID** - Address space identifier.

Compare Register

CP0 Register 11, Select 0



- **Compare** - Interval count compare value

Status Register

CP0 Register 12, Select 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17 16	15 10	9 8	7	6	5	4 3	2	1	0
CU3 0	CU2	CU1 0	CU0	RP	FR	RE	MX	PX	BEV	TS	SR	NMI	0	Impl 00	IM7...IM2	IM1, IM0	KX	SX	UX	KSU	ERL	EXL	IE

NOTE: The Dcache (virtual tags only, not physical tags) is invalidated whenever the Status[ERL] value changes.

- **CU3 = 0**
- **CU1 = 0** (Floating point not implemented)
- **CU2, CU0** - Control access to coprocessors 2 and 0, respectively:
- **RP = 0** (Reduced power mode is not implemented.)
- **FR = 0** (Floating point unit is not implemented.)
- **RE = 0** (Reverse-endian mode is not implemented.)
- **MX = 0** (MDMX™ is not implemented.)
- **PX** - Enables access to 64-bit operations in User mode, without enabling 64-bit addressing.
- **BEV** - Controls the location of exception vectors.
- **TS** - Indicates that the TLB has detected a match on multiple entries (i.e machine-check).
- **SR** - Indicates that the entry through the reset exception vector was due to a Soft Reset.
- **NMI** - Indicates that the entry through the reset exception vector was due to an NMI exception.
- **Impl = 0**
- **IM7...IM2** - Interrupt Masks. When set, these mask bits enable the corresponding Cause[IP7..IP2] interrupts.
- **IM1, IM0** - Interrupt Masks. These are used to control SW interrupts.
- **KX** - Enables access to 64-bit Kernel Segments and use of the XTLB Refill Vector for references to Kernel Segments.
- **SX** - Enables access to 64-bit Supervisor Segments and use of the XTLB Refill Vector for references to Supervisor Segments.
- **UX** - Enables access to 64-bit User Segments, Use of the XTLB Refill Vector for references to User Segments and Execution of instructions which perform 64-bit operations while the processor is operating in User Mode.
- **KSU** - The encoding of this field denotes the base operating mode of the processor.
- **ERL** - Error Level
- **EXL** - Exception Level
- **IE** - Interrupt Enable

IntCtl Register

CP0 Register 12, Select 1

31 29	28 26	25 10	9 5	4 0
IPTI	IPPCI	Reserved 0	VS 0 0000	Reserved 0 0000

- **IPTI** = This field is a read-only copy of CvmCtl[IPTI].
- **IPPCI** = This field is a read-only copy of CvmCtl[IPPCI].
- **VS = 0** (vectored interrupt not Implemented)

SRSCtl Register

CP0 Register 12, Select 2

31 30	29 26	25 22	21 18	17 16	15 12	11 10	9 6	5 4	3 0
Reserved 00	HSS 0000	Reserved 0000	EICSS 0000	Reserved 00	ESS 0000	Reserved 00	EXL 0000	Reserved 00	CSS 0000

NOTE: SRS map is not implemented.

cnMIPS™ cores have no GPR shadow sets so the entire SRSCtl register is read as zero.

Cause Register

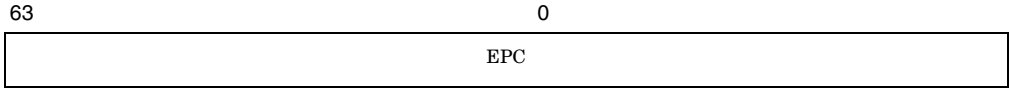
CP0 Register 13, Select 0

31	30	29 28	27	26	25 24	23	22	21 16	15 13	12 10	9 8	7	6 2	1 0
BD	TI	CE	DC	PCI	Reserved 00	IV	WP	Reserved 00 0000	IP7...IP5	IP4...IP2	IP1, IP0	Reserved 0	ExcCode	Reserved 00

- **BD** - Indicates whether the last exception taken occurred in a branch-delay slot.
- **TI** - Timer Interrupt.
- **CE** - Coprocessor unit number referenced when a coprocessor unusable exception is taken.
- **DC** - Disable count register.
- **PCI** - Implemented (two performance counters implemented).
- **IV** - Indicates whether an interrupt exception uses the general exception vector or a special interrupt vector.
- **WP** - Implemented (two watch registers implemented).
- **IP7...IP5** - Indicate an interrupt is pending. May be set for timer interrupts (Cause[TI] is set in this case) and/or performance counter interrupts (Cause[PCI] is set in this case). CvmCtl[IPTI,IPPCI] select the IP bits used for timer and performance counter interrupts, respectively.
- **IP4...IP2** - Indicate an interrupt is pending. may assert due to timer and performance counter interrupts, and may additionally assert due to the CIU interrupt distribution. Refer to Chapter 11, Central Interrupt Unit (CIU).
- **IP1, IP0** - Indicate an interrupt is pending. These are used to control the request for SW interrupts.
- **ExcCode** - Exception code. This field is compliant with MIPS® documentation. Please refer to the MIPS® documentation for cause code value assignment. (See Table 2.)

Exception Program Counter

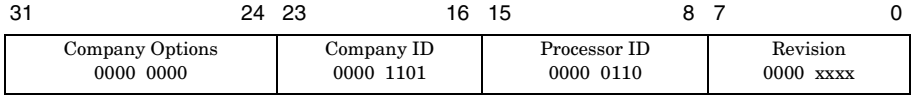
CP0 Register 14, Select 0



- **EPC** - Exception Program Counter

PRId Register

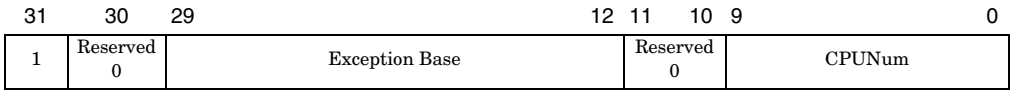
CP0 Register 15, Select 0



- **Company Options** = 0.
- **CompanyID** = 13 (1101_2).
- **Processor ID** = 6.
  0 = CN38XX/CN36XX **2** = CN3005/CN3010 **4** = CN54/5/6/7XX **7** = CN52XX
  1 = CN31XX/CN3020 **3** = CN58XX **6** = CN50XX
- **Revision ID** = 0/1.
  0 = pass 1.0 **1** = pass 1.1

EBase Register

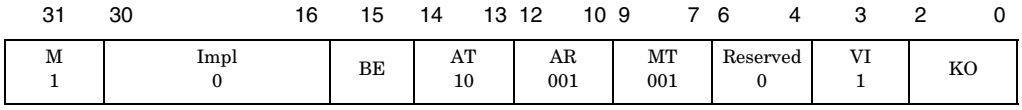
CP0 Register 15, Select 1



- **Exception Base** - In conjunction with bits <31:30>, this field specifies the base address of the exception vectors when StatusBEV = 0.
- **CPUNum** - specifies the cnMIPS™ core ID.

Config Register

CP0 Register 16, Select 0



- **M = 1** (Indicates that the Config1 register is present)
- **IMPL** = Read as 0
- **BE** = $\overline{\text{CvmCtl[LE]}}$
- **AT = 2** (MIPS 64 with access to all address segments)
- **AR = 1** (Release 2)
- **MT = 1** (Standard TLB)
- **VI = 1** (Icache is virtual)
- **KO** - Field is implemented, and the only effect is on the ordering of kseg0 stores. Section 4.8 describes the CN50XX use of the MIPS cache-coherency attributes.

Config1 Register

CP0 Register 16, Select 1

31	30 25	24 22	21 19	18 16	15 13	12 10	9 7	6	5	4	3	2	1	0
M	MMU Size–1	IS	IL	IA	DS	DL	DA	C2	MD	PC	WR	CA	EP	FP
1	11 1111	000	110	011	000	000	111	1	0	1	1	0	1	0

- **M = 1** (Indicates that a Config2 register is present)
- **MMU Size–1 = 63 (**64 entries)
- **IS = 0** (Denotes 64 Icache sets per way)
- **IL = 6** (128 byte line size)
- **IA = 3** (Denotes 4-way Icache)
- **DS = 0** (Denotes 64 Dcache sets per way, actual is 1)
- **DL = 0** (Denotes no Dcache, though cache is present and line size is 128 bytes)
- **DA = 7** (Denotes 8-way Dcache, actual is 64)
- **C2 = 1** (Coprocessor 2 implemented)
- **MD = 0** (MDMX ASE not implemented)
- **PC = 1** (Two Performance Counter registers implemented)
- **WR = 1** (Two Watch registers implemented)
- **CA = 0** (MIPS16e not implemented)
- **EP = 1** (EJTAG implemented)
- **FP = 0** (No FPU implemented)

Config2 Register

CP0 Register 16, Select 2

31	30 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
M	TU	TS	TL	TA	SU	SS	SL	SA
1	000	0000	0000	0000	0000	0100	0110	0111

- **M = 1** (Config 3 is present)
- **TU** - Not implemented (Read as zero)
- **TS = 0** (No tertiary cache)
- **TL = 0** (No tertiary cache)
- **TA = 0** (No tertiary cache)
- **SU** - Not implemented (Read as zero)
- **SS** - 4 (1024 sets per way in secondary code)
- **SL** - 6 (128-byte cache line in secondary cache)
- **SA** - 7 (8-way set-associative secondary cache)

Config3 Register

CP0 Register 16, Select 3

31	30 — 13	12	11 — 8	7	6	5	4	3 — 2	1	0
M	Reserved	RXI	Reserved	LPA	VEIC	VInt	SP	Reserved	SM	TL
0	0	1	0	1	0	0	0	00	0	0

The Config3 register encodes additional capabilities. All fields in the Config3 register are read-only.

- **M = 0** (Config4 is not present)
- **RXI = 1** (read/execute inhibit function is implemented)
- **LPA = 1** (Large physical address support is implemented, and the PageGrain register exists)
- **VEIC = 0** (no external interrupt controller)
- **VInt = 0** (Vectored interrupts not implemented)
- **SP = 0** (1KB page size not implemented)
- **SM = 0** (SmartMIPS™ not implemented)
- **TL = 0** (Trace Logic not implemented)

WatchLo Register

CP0 Register 18, select 0, 1

63 — 3	2	1	0
VAddr	**I**	**R**	**W**

- **VAddr** - specifies the virtual address to match.
- **I** - Implemented at select = 0 (only first watchpoint is instruction)
- **R** - Implemented at select = 1 (only second watchpoint is data)
- **W** - Implemented at select = 1 (only second watchpoint is data)

Two watchpoints are implemented; the first instruction-only and the second data-only.

WatchHi Register

CP0 Register 19, select 0, 1

31	30	29 — 24	23 — 16	15 — 12	11 — 3	2	1	0
M	G	Reserved	ASID	Reserved	Mask	I	R	W
		0		0				

- **M =** Set for select = 0, clear for select = 1 (2 watchpoints implemented)
- **G** - If this bit is one, any address that matches that specified in the WatchLo register will cause a watch exception. If this bit is zero, the ASID field of the WatchHi register must match the ASID field of the EntryHi register to cause a watch exception.
- **ASID** - ASID value which is required to match that in the EntryHi register if the G bit is zero in the WatchHi register.
- **Mask** - Implemented in the maximum width
- **I** - Set by hardware when an instruction fetch condition matches the values in this watch register pair.
- **R** - Set by hardware when a load condition matches the values in this watch register pair.
- **W** - Set by hardware when a store condition matches the values in this watch register pair.

XContext Register

CP0 Register 20, Select 0

63 — 42	41 — 40	39 — 4	3 — 0
PTEBase	R	BadVPN2	Reserved 0

- **PTEBase** - This field is for use by the operating system and is normally written with a value that allows the operating system to use the Context Register as a pointer into the current PTE array in memory.
- **R** - Region field. Contains bits 63...62 of the virtual address.
- **BadVPN2** - Bad Virtual Page Number/2 field is written by hardware on a miss.

Debug Register

CP0 Register 23, Select 0

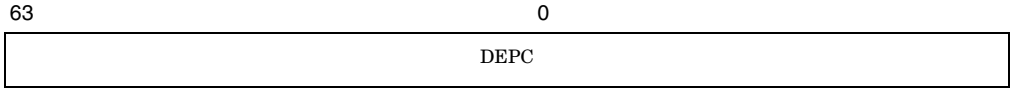
31	30	29	28	27	26	25	24	23	22	21	20	19
DBD	DM	No DCR 0	LSNM	Doze 0	Halt 0	CountDM	IBusEP 0	MCheckP 0	CacheEP 0	DBusEP	IEXI	DDBSImpr 0

18	17 — 15	14 — 10	9	8	7 — 6	5	4	3	2	1	0
DDBLImpr	EJTAG 011	DExcCode	NoSSt 0	SSt	Reserved 0	DINT	DIB	DDBS	DDBL	DBp	DSS

- **DBD** - Indicates whether the last debug exception or exception in Debug Mode occurred in a branch or jump delay slot.
- **DM** - Indicates that the processor is operating in Debug Mode.
- **NoDCR = 0** (dseg present)
- **LSNM** - Implemented
- **Doze = 0** (No low power mode)
- **Halt = 0** (Internal bus clock never stopped)
- **CountDM** - Implemented
- **IBusEP = 0** (No imprecise bus errors)
- **MCheckP = 0** (No imprecise machine checks)
- **CacheEP = 0** (No imprecise cache errors)
- **DBusEP** - Implemented
- **IEXI** - Implemented
- **DDBSImpr = 0** (No imprecise Debug Data breakpoint on store)
- **DDBLImpr** - Implemented for load value breakpoints
- **EJTAGver = 3** (EJTAG version 3.10)
- **DExcCode** - Implemented
- **NoSSt = 0** - (Single step is implemented)
- **SSt** - Implemented
- **DINT** - Implemented
- **DIB** - Implemented
- **DDBS** - Implemented
- **DDBL** - Implemented
- **DBP** - Implemented
- **DSS** - Implemented

Debug Exception Program Counter Register

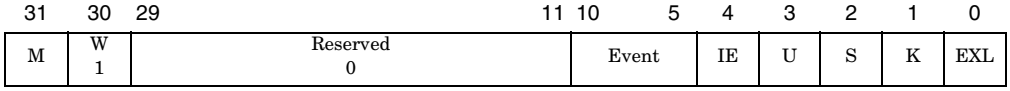
CP0 Register 24, Select 0



For imprecise debug exceptions and imprecise exceptions in debug mode, the DEPC register contains the address at which execution is resumed when returning to non-debug mode.

Performance Counter Control Register

CP0 Register 25, Select 0, 2



- **M** - 1 for sel = 0, 0 for sel = 2 (Two performance counters implemented)
- **W = 1** - (64-bit counter)
- **Event** - Selects the event to be counted by the corresponding Counter Register. See Table 4–28

Table 4–28 Performance Counter Control Register Events

Value	Event	Meaning
0x0	—	Reserved
0x1	PERF_CNT_CLK	Conditionally clocked cycles (as opposed to count/cvm_count which count even with no clocks)
0x2	PERF_CNT_ISSUE	Instructions issued but not retired
0x3	PERF_CNT_RET	Instructions retired
0x4	PERF_CNT_NISSUE	Cycles no issue
0x5	PERF_CNT_SISSUE	Cycles single issue
0x6	PERF_CNT_DISSUE	Cycles dual issue
0x7	PERF_CNT_IFI	Cycle ifetch issued (but not necessarily commit to pp_mem)
0x8	PERF_CNT_BR	Branches retired
0x9	PERF_CNT_BRMIS	Branch mispredicts
0xA	PERF_CNT_J	Jumps retired
0xB	PERF_CNT_JMIS	Jumps mispredicted
0xC	PERF_CNT_REPLAY	Mem Replays
0xD	PERF_CNT_IUNA	Cycles idle due to unaligned_replays
0xE	PERF_CNT_TRAP	trap_6a signal
0xF	—	Reserved
0x10	PERF_CNT_UULOAD	Unexpected unaligned loads (REPUN=1)
0x11	PERF_CNT_UUSTORE	Unexpected unaligned store (REPUN=1)
0x12	PERF_CNT_ULOAD	Unaligned loads (REPUN=1 or USEUN=1)
0x13	PERF_CNT_USTORE	Unaligned store (REPUN=1 or USEUN=1)
0x14	PERF_CNT_EC	Exec clocks (must set CvmCtl[DISCE] for accurate timing)
0x15	PERF_CNT_MC	Mul clocks (must set CvmCtl[DISCE] for accurate timing)
0x16	PERF_CNT_CC	Crypto clocks (must set CvmCtl[DISCE] for accurate timing)
0x17	PERF_CNT_CSRC	Issue_csr clocks (must set CvmCtl[DISCE] for accurate timing)
0x18	PERF_CNT_CFETCH	Icache committed fetches (demand+prefetch)
0x19	PERF_CNT_CPREF	Icache committed prefetches

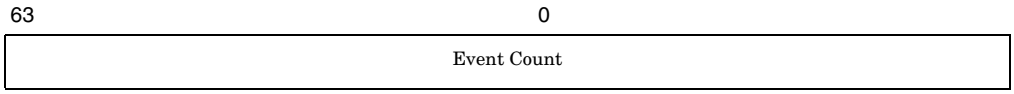
Table 4–28 Performance Counter Control Register Events (Continued)

Value	Event	Meaning
0x1A	PERF_CNT_ICA	Icache aliases
0x1B	PERF_CNT_II	Icache invalidates
0x1C	PERF_CNT_IP	Icache parity error
0x1D	PERF_CNT_CIMISS	Cycles idle due to imiss (must set CvmCtl[DISCE] for accurate timing)
0x1E	—	Reserved
0x1F	—	Reserved
0x20	PERF_CNT_WBUF	Number of write buffer entries created
0x21	PERF_CNT_WDAT	Number of write buffer data cycles used (may need to set CvmCtl[DISCE] for accurate counts)
0x22	PERF_CNT_WBUFLD	Number of write buffer entries forced out by loads
0x23	PERF_CNT_WBUFFL	Number of cycles that there was no available write buffer entry (may need to set CvmCtl[DISCE] and CvmMemCtl[MCLKALWYS] for accurate counts)
0x24	PERF_CNT_WBUFTR	Number of stores that found no available write buffer entries
0x25	PERF_CNT_BADD	Number of address bus cycles used (may need to set CvmCtl[DISCE] for accurate counts)
0x26	PERF_CNT_BADDL2	Number of address bus cycles not reflected (i.e. destined for L2) (may need to set CvmCtl[DISCE] for accurate counts)
0x27	PERF_CNT_BFILL	Number of fill bus cycles used (may need to set CvmCtl[DISCE] for accurate counts)
0x28	PERF_CNT_DDIDS	Number of Dstream DIDs created
0x29	PERF_CNT_IDIDS	Number of Istream DIDs created
0x2A	PERF_CNT_DIDNA	Number of cycles that no DIDs were available (may need to set CvmCtl[DISCE] and CvmMemCtl[MCLKALWYS] for accurate counts)
0x2B	PERF_CNT_LDS	Number of load issues
0x2C	PERF_CNT_LMLDS	Number of local memory load issues
0x2D	PERF_CNT_IOLDS	Number of I/O load issues
0x2E	PERF_CNT_DMLDS	Number of loads that were not prefetches and missed in the cache
0x2F	—	Reserved
0x30	PERF_CNT_STS	Number of store issues
0x31	PERF_CNT_LMSTS	Number of local memory store issues
0x32	PERF_CNT_IOSTS	Number of I/O store issues
0x33	PERF_CNT_IOBDMA	Number of IOBDMA operations
0x34	—	Reserved
0x35	PERF_CNT_DTLB	Number of dstream TLB refill, invalid, or modified exceptions
0x36	PERF_CNT_DTLBAD	Number of dstream TLB address errors
0x37	PERF_CNT_ITLB	Number of istream TLB refill, invalid, or address error exceptions
0x38	PERF_CNT_SYNC	Number of SYNC stall cycles (may need to set CvmCtl[DISCE] for accurate counts)
0x39	PERF_CNT_SYNCIOB	Number of SYNCIOBDMA stall cycles (may need to set CvmCtl[DISCE] for accurate counts)
0x3A	PERF_CNT_SYNCW	Number of SYNCWs or SYNCWSs
0x3B -0x3F	—	Reserved

- **IE** - Interrupt Enable.
- **U** - Enables event counting in User Mode.
- **S** - Enables event counting in Supervisor Mode
- **K** - Enables event counting in Kernel Mode.
- **EXL** - Enables event counting when the EXL bit in the Status register is one and the ERL bit in the Status register is zero.

Performance Counter Counter Register

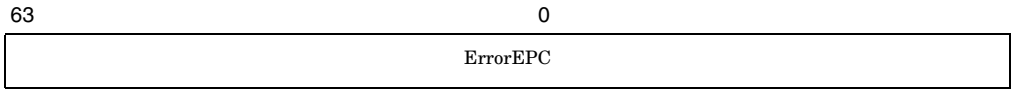
CP0 Register 25, select 1, 3



Increments once for each event that is enabled by the corresponding Control Register. When the MSB is one, a pending interrupt request is ORed with those from other performance counters and indicated by the PCI bit in the Cause register.

ErrorEPC

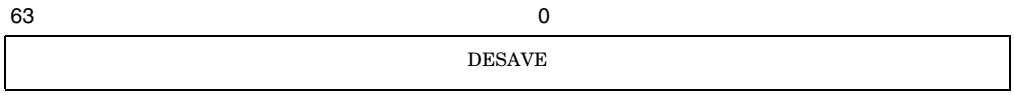
CP0 Register 30, Select 0



The ErrorEPC register contains the virtual address at which instruction processing can resume after servicing an error.

DESAVE Register

CP0 Register 31, Select 0)



The Debug Exception Save (DESAVE) register is a read/write register that functions as a simple scratchpad register.

4.11.1 Cavium Networks-Specific Coprocessor 0 Registers

CacheErr (Icache)

Cache Error Icache Register, Implemented at reg# = 27, select = 0

Implemented to latch Icache error information.

63 55	54 48	47	46 40	39 38	37	36	35	34	33	32	31 15	14 13	12 11	10 5	4 3	2 1	0
Reserved 0	badcolf	Reserved 0	badcol	Reserved 0000	regfail	lrufail	aesfail	hshfail	BHTbroke	icbroke	Reserved 0	qw	row	set	way	Reserved 00	dperr

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:55>	—	RO	0	0	Reserved
<54:48>	badcolf	RO	0	0	Bad column stored in fuses. 0x7F for no fuse
<47>	—	RO	0	0	Reserved
<46:40>	badcol	RO	0	0	Bad column in Icache - 0x7F for perfect
<39:38>	—	RO	0	0	Reserved
<37>	regfail	RO	0	0	Register file BIST failed
<36>	lrufail	RO	0	0	LRU BIST failed
<35>	hshfail	RO	0	0	HSH/GFM BIST failed
<34>	aesfail	RO	0	0	AES BIST failed
<33>	BHTBK	RO	0	0	BHT is unrepairable
<32>	icbk	RO	0	0	Icache is unrepairable
<31:15>	—	RO	0	0	Reserved
<14:13>	qw	R/W	0	0	Address<4:3>
<12:11>	row	R/W	0	0	Address<6:5>
<10:5>	set	R/W	0	0	Address<12:7>
<4:3>	way	R/W	0	0	Selects the cache way.
<2:1>	—	RO	0	0	Reserved
<0>	dperr	R/W	0	0	Data parity error

CacheErr (Dcache)

Cache Error Dcache Register, Implemented at reg# = 27, select = 1

This register is updated by the hardware on any load to cacheable memory (i.e. L2/DRAM), not just when there is a parity error. For this register to contain useful information following a Cache Error on Data Access Exception (refer to Table 4–36 and Section 4.19), there must be no L2/DRAM loads prior to the mfc0 27,1 instruction that captures the data cache error information during the exception handler.

Implemented to latch Dcache error information. .

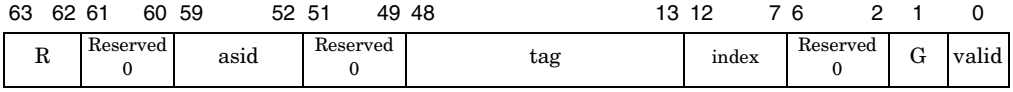
63 14	13 8	7 3	2 1	0
Reserved 0	way	va73	Reserved 0	perr

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:13>	—	RO	0	0	Reserved
<13:8>	way	RO	0	0	Way in which the parity error happened
<7:3>	va73	RO	0	0	VA<7:3> of the address that had the error
<2:1>	—	RO	0	0	Reserved
<0>	perr	R/W	0	0	Set on Dcache parity error

TagLo Register (Icache)

Implemented at reg# = 28, select = 0 (Icache)

Icache implemented.

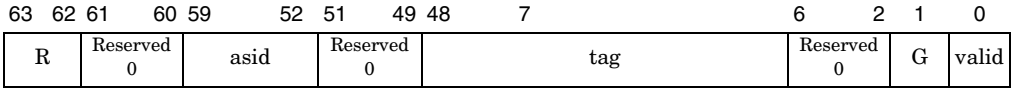


Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:62>	R	R/W	0	0	<63:62> of virtual address
<61:60>	—	RO	0	0	Reserved
<59:52>	asid	R/W	0	0	ASID
<51:49>	—	RO	0	0	Reserved
<48:13>	tag	R/W	0	0	<48:13> of the virtual address
<12:7>	index	R/W	0	0	<12:7> from the address in the cache instruction
<6:2>	—	RO	0	0	Reserved
<1>	G	R/W	0	0	G bit
<0>	valid	R/W	0	0	valid bit

TagLo Register (Dcache)

Implemented at reg# = 28, select = 2 (Dcache)

Dcache implemented.

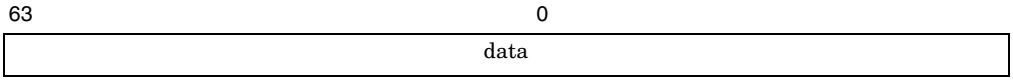


Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:62>	R	R/W	0	0	<63:62> of virtual address
<61:60>	—	RO	0	0	Reserved
<59:52>	asid	R/W	0	0	ASID
<51:49>	—	RO	0	0	Reserved
<48:7>	tag	R/W	0	0	<48:7> of the virtual address
<6:2>	—	RO	0	0	Reserved
<1>	G	R/W	0	0	G bit
<0>	valid	R/W	0	0	valid bit

DataLo Register (Icache)

Implemented at reg# = 28, select = 1 (Icache))

The repair solution (i.e. Icache CacheErr[badcol]) must be known to interpret the raw Icache data. For a perfect Icache, raw Icache data[63:0] is the data in the instruction cache (big-endian format).

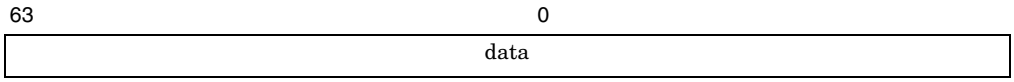


Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	data	R/W	0	0	Raw Icache data <63:0>.

DataLo Register (Dcache)

Implemented at reg# = 28, select = 3 (Dcache)

The 64 bits read from the CACHE instruction. Dcache implemented. DataLo is unpredictable after a load to a noncacheable (i.e. I/O) address. To successfully read out the Dcache, there must not be a load to a noncacheable address between the CACHE instruction and the read of DataLo.

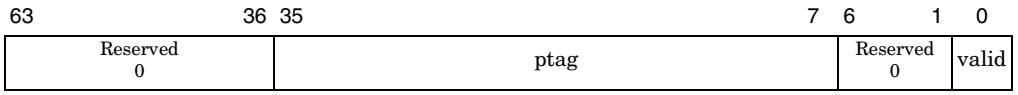


Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	data	R/W	0	0	Data from the Dcache.

TagHi Register

Implemented at reg# = 29, select = 2 (Dcache))

Dcache implemented.

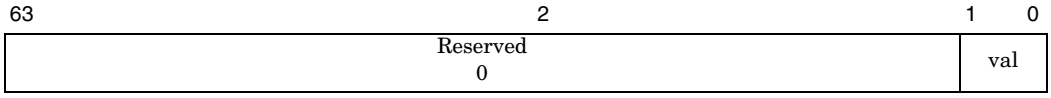


Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RO	0	0	Reserved
<35:7>	ptag	R/W	0	0	Bits <35:7> of the physical address
<6:1>	—	RO	0	0	Reserved
<0>	valid	R/W	0	0	valid bit for physical tag

DataHi Register (Icache)

Implemented at reg# = 29, select = 1 (Icache)

The repair solution (i.e. Icache CacheErr[badcol]) must be known to interpret the raw Icache data. For a perfect Icache, raw Icache data[64] is the parity bit of the data in the instruction cache and [65] is not used.

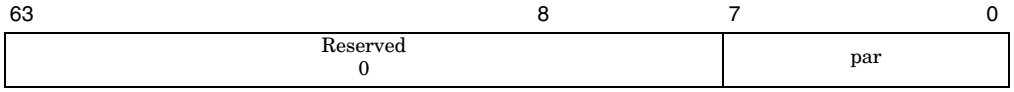


Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RO	0	0	Reserved
<1:0>	val	R/W	0	0	Raw Icache data <65:64>.

DataHi Register (Dcache)

Implemented at reg# = 29, select = 3 (Dcache)

The eight parity bits read from the CACHE instruction. Dcache implemented. DataHi is unpredictable after a load to a noncacheable (i.e. I/O) address. To successfully read out the Dcache, there must not be a load to a noncacheable address between the CACHE instruction and the read of DataHi.



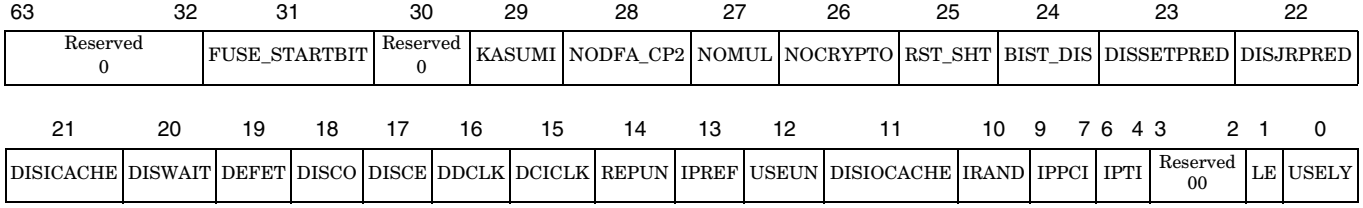
Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RO	0	0	Reserved
<7:0>	par	R/W	0	0	Parity bits after cache idxldtag

CvmCtl Register

Cvm Control Register, Implemented at reg# = 9, select = 7

Controls the Icache and instruction issue.

Field Descriptions



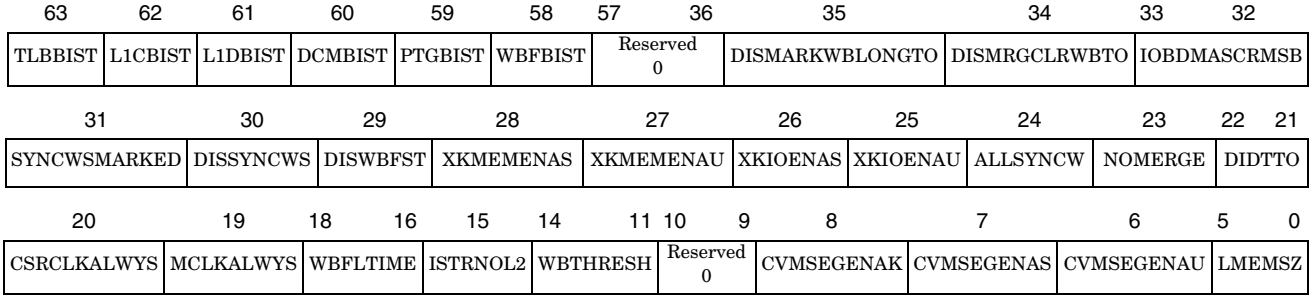
Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RO	0x0	0x0	Reserved.
<31>	FUSE_START BIT	RO	0	0	Must be 1.
<30>	—	RO	0	0	Reserved. Must be 0.
<29>	KASUMI	RO	1	1	When set, KASUMI operations are enabled. CvmCtl[KASUMI] =0 whenever CvmCtl[NOCRYPTO] = 1.
<28>	NODFA_CP2	RO	1	1	Always set to 1.
<27>	NOMUL	RO	0	0	When set, large mul operations are disabled
<26>	NOCRYPTO	RO	0	0	When set, crypto operations are disabled
<25>	RST_SHT	RO	0	0	When set, reset delays are short (should be 0)
<24>	BIST_DIS	RO	0	0	When set, BIST is disabled (should be 0)
<23>	DISSETPRED	R/W	0	0	When set, disables power-saving feature of Icache.
<22>	DISJRPRED	R/W	0	0	When set, JALs no longer push the jump stack (i.e. the jump stack remains frozen).

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<21>	DISICACHE	R/W	0	0	When set, Icache is disabled (operates in pass-through mode). Normally, this bit should not be set. When DISICACHE = 0, and is being written to 1, one of the following must be true: ● The linear instruction stream including the MTC0/DMTC0 instruction that writes CvmCtl[DISICACHE] = 1 is running out of I/O space or DMSEG ● The MTC0/DMTC0 instruction that writes CvmCtl[DISICACHE] = 1 is followed immediately with a SYNCI instruction (in the same cache block). ● There was a one cache line backward branch before the MTC0/ DMTC0 instruction that writes CvmCtl[DISICACHE] = 1
<20>	DISWAIT	R/W	0	0	When set, wait is disabled
<19>	DEFET	R/W	0	0	When set, fetch under fill is disabled
<18>	DISCO	R/W	0	0	When set, disable the conditional clocks (i.e. always clock) mul/crypto
<17>	DISCE	R/W	0	0	When set, disable the conditional clocks (i.e. always clock) issue
<16>	DDCLK	R/W	0	0	When set, disable the conditional clocks (i.e. always clock) exec
<15>	DCICLK	R/W	0	0	When set, disable the conditional clocks (i.e. always clock) issue unit CSRs
<14>	REPUN	R/W	0	0	When set, LH+LHU+LW+LWU+LD+SH+SW+SD unaligned accesses handled by HW
<13>	IPREF	R/W	0	0	When set, instruction prefetching is disabled
<12>	USEUN	R/W	0	0	When set, CVM ULW+ULD+USW+USD insts, else MIPS LWL+LWR+LDL+LDR+SWL+SWR+SDL+SDR
<11>	DISIOCACHE	R/W	0	0	When set, the iocache feature is disabled, which cause noncompliance for EJTAG and may cause extra I/O-space fetches.
<10>	IRAND	R/W	0	0	When set, the Icache uses random replacement, else use “bitmask LRU”
<9:7>	IPPCI	R/W	0x7	0x7	Selects the Cause[IP] bit to which the Performance Counter Interrupt request is merged. Legal values are CvmCtl[IPPCI] = 2 .. 7, where CvmCtl[IPPCI]=*n* selects Cause[IP<*n*>] for the interrupt. IntCtl[IPPCI] is a read-only copy of CvmCtl[IPPCI].
<6:4>	IPTI	R/W	0x7	0x7	Selects the Cause[IP] bit to which the Timer Interrupt request is merged. Legal values are CvmCtl[IPTI] = 2 .. 7, where CvmCtl[IPTI]=*n* selects Cause[IP<*n*>] for the interrupt. IntCtl[IPTI] is a read-only copy of CvmCtl[IPTI].
<3:2>	—	RO	0x0	0x0	Reserved
<1>	LE	R/W	0	0	When set, the core defaults to LE (i.e. Config[BE] is a read-only copy of the inverse of CvmCtl[LE].)
<0>	USELY	R/W	0	0	When clear use *L for unaligned opcode for BE, *R for LE. When set unaligned opcode is always the *L opcodes

CvmMemCtl Register

Cvm Memory Control Register, Implemented at reg# = 11, select = 7

Controls the cnMIPS™ cores' memory system.



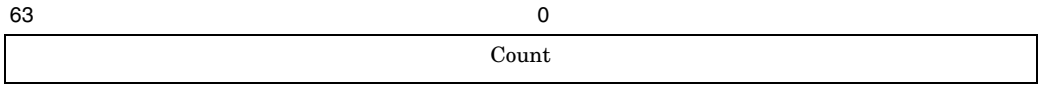
Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	TLBBIST	RO	x	x	1 = BIST fail, 0 = BIST pass
<62>	L1CBIST	RO	x	x	1 = BIST fail, 0 = BIST pass
<61>	L1DBIST	RO	x	x	1 = BIST fail, 0 = BIST pass
<60>	DCMBIST	RO	x	x	1 = BIST fail, 0 = BIST pass
<59>	PTGBIST	RO	x	x	1 = BIST fail, 0 = BIST pass
<58>>	WBFBIST	RO	x	x	1 = BIST fail, 0 = BIST pass
<57:36>	—	R/W	0	0	Reserved
<35>	DISMARKWBLONGTO	R/W	0	0	If set, marked write-buffer entries time out the same as other entries; if clear, marked write-buffer entries use the maximum timeout.
<34>	DISMRGCLRWBTO	R/W	0	0	If set, a merged store does not clear the write-buffer entry timeout state.
<33:32>	IOBDMASCRMSB	R/W	0x0	0x0	Two bits that are the MSBs of the resultant CVMSEG LM word location for an IOBDMA. The other 8 bits come from the SCRADDR field of the IOBDMA.
<31>	SYNCWSMARKED	R/W	0	0	If set, SYNCWS and SYNCS only order marked stores; if clear, SYNCWS and SYNCS only order unmarked stores. SYNCWSMARKED has no effect when DISSYNCWS is set.
<30>	DISSYNCWS	R/W	0	0	If set, SYNCWS acts as SYNCW and SYNCS acts as SYNC.
<29>	DISWBFST	R/W	0	0	If set, no stall happens on write buffer full. **NOTE:** This bit should always be cleared to 0.
<28>	XKMEMENAS [1]	R/W	0	0	If set (and SX set), supervisor-level loads/stores can use XKPHYS addresses with VA<48>==0
<27>	XKMEMENAU [1]	R/W	0	0	If set (and UX set), user-level loads/stores can use XKPHYS addresses with VA<48>==0
<26>	XKIOENAS [1]	R/W	0	0	If set (and SX set), supervisor-level loads/stores can use XKPHYS addresses with VA<48>==1
<25>	XKIOENAU [1]	R/W	0	0	If set (and UX set), user-level loads/stores can use XKPHYS addresses with VA<48>==1
<24>	ALLSYNCW	R/W	0	0	If set, all stores act as SYNCW (NOMERGE must be set when this is set) RW, reset to 0 if set, all stores act as SYNCW (NOMERGE must be set when this is set).
<23>	NOMERGE	R/W	0	0	If set, no stores merge, and all stores reach the coherent bus in order.
<22:21>	DIDTTO [2]	R/W	0	0	Selects the bit in the counter used for DID time-outs 0 = 2^{31}, 1 = 2^{30}, 2 = 2^{29}, 3 = 2^{14}. Actual time-out is between 1× and 2× this interval. For example, with DIDTTO=3, expiration interval is between 16K and 32K.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<20>	CSRCKALWYS	R/W	0	0	If set, the (mem) CSR clock never turns off.
<19>	MCLKALWYS	R/W	0	0	If set, the core's memory clock never turns off.
<18:16>	WBFLTIME	R/W	0x4	0x4	Selects the bit in the counter used for write buffer flush time-outs (WBFLT+11) is the bit position in an internal counter used to determine expiration. The write buffer expires between 1× and 2× this interval. For example, with WBFLT = 0, a write buffer expires between 2K and 4K cycles after the write buffer entry is allocated.
<15>	ISTRNOL2	R/W	0	0	If set, do not put Istream in the L2 cache.
<14:11>	WBTHRESH	R/W	0xC	0xC	The write buffer threshold. The legal range is 0x2–0xE. Values 0x0, 0x1, and 0xF are illegal
<10:9>	—	R/W	0x0	0x0	Reserved
<8>	CVMSEGENAK [1]	R/W	0	0	If set, CVMSEG is available for loads/stores in kernel/debug mode. CVMSEGENAK must be enabled when either CVMSEGENAS or CVMSEGENAU are enabled?
<7>	CVMSEGENAS [1]	R/W	0	0	If set, CVMSEG is available for loads/stores in supervisor mode.
<6>	CVMSEGENAU [1]	R/W	0	0	If set, CVMSEG is available for loads/stores in user mode.
<5:0>	LMEMSZ	R/W	0x0	0x0	Size of local memory in cache blocks, 54 (6912 bytes) is max legal value.

1. Note that both CvmMemCtl[CVMSEGENA*] and CvmMemCtl[XK*] enable the corresponding range (CVMSEG and XKPHYS with VA<48>==1) only for loads/store operations, and NOT for instruction references.
2. Note that DID time-outs are the only source of bus-error exceptions. These bus-error exceptions may not be recoverable, but the handler can extract information for diagnostic purposes.

CvmCount Register

Implemented at reg# = 9, sel = 6



Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	count	RW	0	0	Cycle counter. Value may also be read at RDHWR 31. Refer to Appendix A.

Reset to 0x0. Increments by one every cycle. See "Multicore Debug Register" on page 186 for more details about the behavior of CvmCount during debug.

Multicore Debug Register

Implemented at reg# =22, select =0

63 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved 0	DExcC	CGSTP	CvGSTP	CvDM	GSDB	Reserved 0	MskM2	MskM1	MMC0	Reserved 0	Pls2	Pls1	Pls0	Reserved 0	MCD2	MCD1	MCD0

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:17>	—	RO	0	0	Reserved
<16>	DExcC	R/W	0	0	Set to one when Debug[DExcCode] is valid and should be interpreted
<15>	CGSTP	R/W	0	0	If set, Count does not increment when GSTOP is set
<14>	CvGSTP	R/W	0	0	If set, CvmCount does not increment when GSTOP is set
<13>	CvDM	R/W	0	0	If set, CvmCount does not increment when Debug[DM] set
<12>	GSDB	R/W	0	0	If one, SDBBP execution pulses global MCD0 wire
<11>	—	RO	0	0	Reserved
<10>	MskM2	R/W	0	0	When set, pulses on the global MCD2 wire set MCD2.
<9>	MskM1	R/W	0	0	When set, pulses on the global MCD1 wire set MCD1.
<8>	MMC0	R/W	0	0	When set, pulses on the global MCD0 wire set MCD0.
<7>	—	RO	0	0	Reserved
<6>	Pls2	RAZ/W	0	0	Writing a 1 pulses global MCD2 wire.
<5>	Pls1	RAZ/W	0	0	Writing a 1 pulses global MCD1 wire.
<4>	Pls0	RAZ/W	0	0	Writing a 1 pulses global MCD0 wire.
<3>	—	RO	0	0	Reserved
<2>	MCD2	R/W1C	0	0	Reset to zero. If set, enter debug mode.
<1>	MCD1	R/W1C	0	0	Reset to zero. If set, enter debug mode.
<0>	MCD0	R/W1C	0	0	Reset to zero. If set, enter debug mode.

4.12 cnMIPS™ Core EJTAG DRSEG Registers

The memory-mapped EJTAG registers are located in the debug register segment (drseg), which is a subsegment of the debug segment (dseg). They are accessible by the debug software when the processor is executing in Debug Mode. These registers provide both miscellaneous debug control and control of hardware breakpoints.

Table 4–29 EJTAG DRSEG Registers Summary

Register	Implemented/Not Implemented	Comments
Debug Control Register (DCR)	Implemented	See page 187
Instruction Breakpoint Status (IBS)	Implemented	See page 187
Instruction Breakpoint Address (IBA)	Four implemented	See page 187
Instruction Breakpoint Address Mask (IBM)	Four implemented	See page 187
Instruction Breakpoint ASID (IBASID)	Four implemented	See page 188
Instruction Breakpoint Control (IBC)	Four implemented	See page 188
Data Breakpoint Status (DBS)	Implemented	See page 188
Data Breakpoint Address (DBA)	Four implemented	See page 188
Data Breakpoint Address Mask (DBM)	Four implemented	See page 189
Data Breakpoint ASID (DBASID)	Four implemented	See page 189
Data Breakpoint Control (DBC)	Four implemented	See page 189
Data Breakpoint Value (DBV)	Four implemented	See page 189

Debug Control Register (DCR)

Address: Offset 0x0000

31 30	29	28 18	17	16	15 10	9	8 6	5	4	3	2	1	0
Reserved 0	ENM	Reserved 0	DataBrk 1	InstBrk 1	Reserved 0	PCS 1	PCR	Reserved 0	IntE	NMIE	NMIpend	SRstE 0	ProbEn

- **ENM** - !CvmCtl[LE]
- **DataBrk = 1** (4 data hardware breakpoints implemented)
- **InstBrk = 1** (4 instruction hardware breakpoints implemented)
- **PCS = 1** (PC sampling is supported)
- **PCR** - PC Sampling rate. Values 0 to 7 map to values 25 to 212 cycles, respectively (i.e. a PC sample is written out every 32, 64, 128, 256, 512, 1024, 2048, or 4096 cycles respectively). The external probe or software is allowed to set this value to the desired sample rate.
- **IntE** - Hardware and software interrupt enable for Non-Debug Mode.
- **NMIE** - Non-Maskable Interrupt (NMI) enable for Non-Debug Mode.
- **NMIpend** - Indication for pending NMI.
- **SRstE = 0** (Soft-reset masking not implemented)
- **ProbEn** - Indicates ECR[ProbEn]

Instruction Breakpoint Status (IBS) Register

Address: Offset 0x1000

63 31	30	29 28	27 24	23 15	14 4	3 0
Reserved 0	ASIDsup 1	Reserved 0	BCN 0100	Reserved 0	BS14...BS4 0	BS3...BS0

- **ASIDsup = 1** (ASID compare implemented)
- **BCN = 4** (4 instruction breakpoints implemented)
- **BS [14:4] = 0 (**Not implemented)
- **BS [3:0]** - Implemented

Instruction Breakpoint Address (IBA0...3) Register

Address: Offset 0x1100 + 0x100 × n

63 0
IBA

- **IBA** - Instruction Breakpoint Address (IBA0...3) register has the virtual address used in the condition for instruction breakpoint 0...3. It is located at drseg segment offset 0x1100 + 0x100 × n.

Instruction Breakpoint Address Mask (IBM0...3) Register

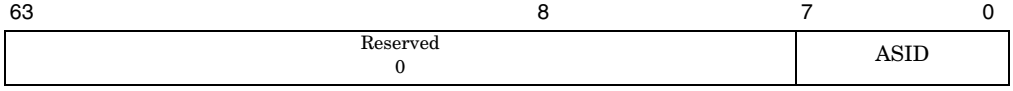
Address: Offset 0x1108 + 0x100 × n

63 0
IBM

- **IBM** - Instruction Breakpoint Address Mask (IBM0...3) register has the address compare mask used in the condition for instruction breakpoint 0...3. The address that is masked is in the IBA0...3 register. The IBM0...3 register is located at drseg segment offset 0x1108 + 0x100 × 0...3.

Instruction Breakpoint ASID (IBASID0...3) Register

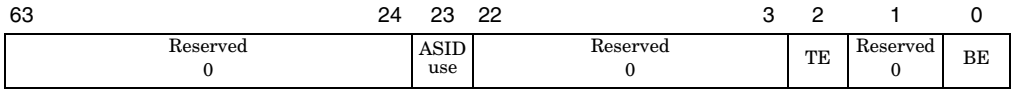
Address: Offset 0x1110 + 0x100 × n



- **ASID** - The Instruction Breakpoint ASID (IBASID0...3) register has the ASID value used in the compare for instruction breakpoint 0...3. It is located at drseg segment offset 0x1110 + 0x100 × 0...3.

Instruction Breakpoint Control (IBC0...3) Register

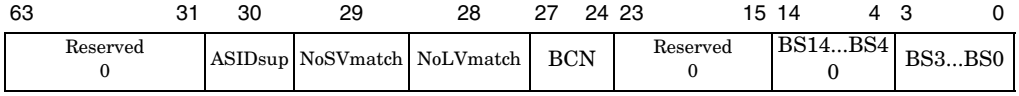
Address: Offset 0x1118 + 0x100 × n



- **ASIDuse** - Implemented
- **TE** - Use instruction breakpoint 0...3 as triggerpoint. See 4–35Table 4–35 Breakpoint Match Behavior on page 198.
- **BE** - Use instruction breakpoint 0...3 as breakpoint. See 4–35Table 4–35 Breakpoint Match Behavior on page 198.

Data Breakpoint Status (DBS) Register

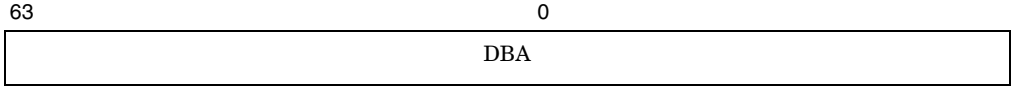
Address: Offset offset 0x2000.



- **ASIDsup = 1** (ASID compare implemented)
- **NoSVmatch = 0** (Store values compare supported)
- **NoLVmatch = 0** (Load values compare supported)
- **BCN = 4** (4 data breakpoints implemented)
- **BS [14:4] = 0** - (Not implemented)
- **BS [3:0]** - Implemented

Data Breakpoint Address (DBA0...3) Register

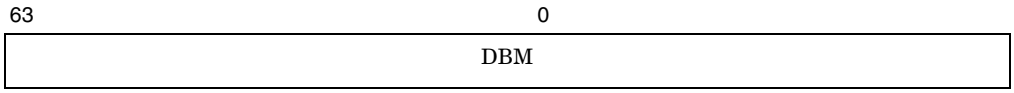
Address: Offset 0x2100 + 0x100 × n



- **DBA** - Data Breakpoint Address (DBA0...3) register has the virtual address used in the condition for data breakpoint 0...3. This register is located at drseg segment offset 0x2100 + 0x100 × 0...3.

Data Breakpoint Address Mask (DBM0...3) Register

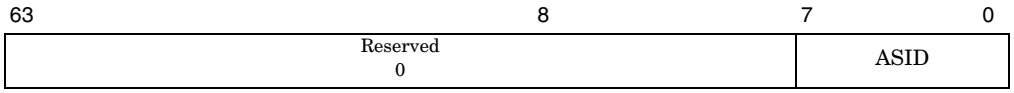
Address: Offset 0x2108 + 0x100 × n



- **DBM** - Data Breakpoint Address Mask (DBM0...3) register has the address compare mask used in the condition for data breakpoint 0...3. The address that is masked is in the DBA0...3 register. The DBM0...3 register is located at drseg segment offset 0x2108 + 0x100 × 0...3.

Data Breakpoint ASID (DBASID0...3) Register

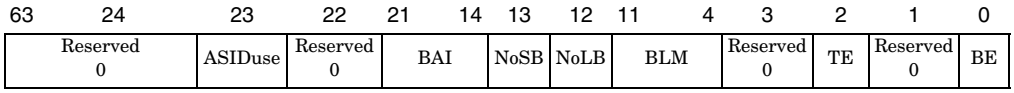
Address: Offset 0x2110 + 0x100 × n



- **ASID** - The Data Breakpoint ASID (DBASID0...3) register has the ASID value used in the compare for data breakpoint 0...3. It is located at drseg segment offset 0x2110 + 0x100 × 0...3.

Data Breakpoint Control (DBC0...3) Register

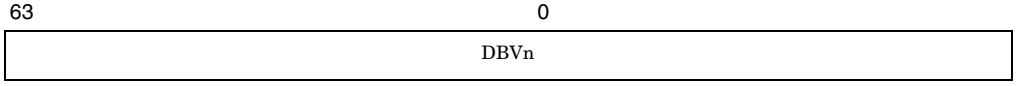
Address: Offset 0x2118 + 0x100 × n



- **ASIDuse = 1** - (ASID compare Implemented)
- **BAI[7:0]** - Implemented (byte lines in value compare)
- **NoSB** - Controls whether condition for data breakpoint is ever fulfilled on a store access:
- **NoLB** - Controls whether condition for data breakpoint is ever fulfilled on a load access:
- **BLM[7:0]** - (Byte lines in value compare)
- **TE** - Use instruction breakpoint n as triggerpoint. See 4–35Table 4–35 Breakpoint Match Behavior on page 198.
- **BE** - Use instruction breakpoint n as breakpoint. See 4–35Table 4–35 Breakpoint Match Behavior on page 198.

Data Breakpoint Value (DBV0...3) Register

Address: Offset 0x2120 + 0x100 × n



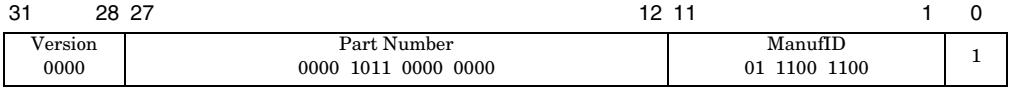
The Data Breakpoint Value (DBV0...3) register has the value used in the condition for data breakpoint 0...3.

4.13 cnMIPS™ Core EJTAG TAP Registers

Table 4–30 EJTAG TAP Registers Summary

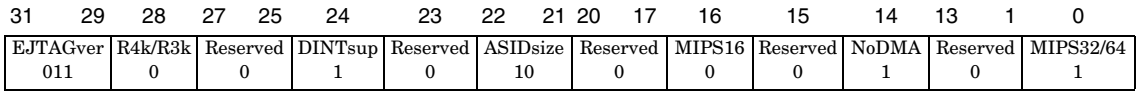
Register	Implemented/Not Implemented	Comments
Device ID	Implemented	See page 190
Implementation	Implemented	See page 190
Data	Implemented	See page 191
Address	Implemented	See page 191
EJTAG Control	Implemented	See page 191
PC Sample	Implemented	See page 191
EJTAG Boot Indication	Implemented	See page 192
Bypass	Implemented	See page 192
Fastdata	Implemented	See page 192

Device ID Register Format



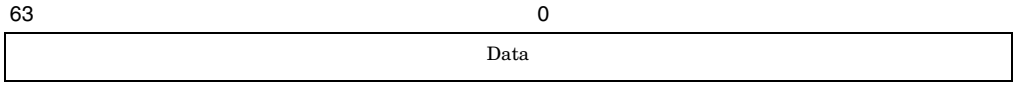
- **Version = 0** (Identifies the version of a specific device)
- **Part Number = 0xB00** (Identifies the part number of a specific device)
- **ManufID = 0x1CC**

Implementation Register Format (TAP Instruction IMPCODE)



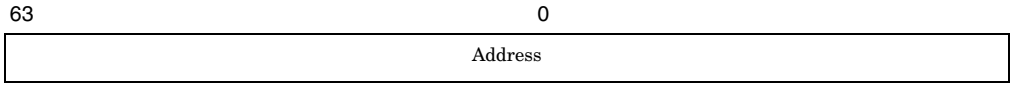
- **EJTAGver = 3** (EJTAG Version 3.10)
- **R4k/R3k = 0** (R4K privileged environment)
- **DINTsup = 1** (DINT is supported from TAP)
- **ASIDsize = 2** (8-bit ASID)
- **NoDMA = 1** (Implemented)
- **MIPS16 = 0** (No MIPS16e support)
- **MIPS32/64 = 1** (64-bit processor)

Data Register (TAP Instruction DATA, ALL, or FASTDATA)



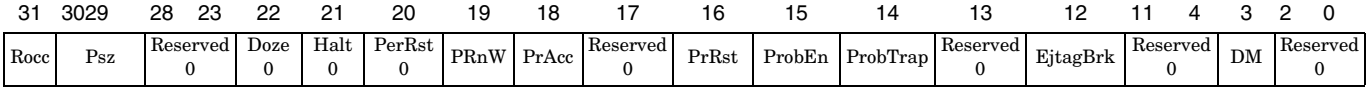
The read/write Data register is used for opcode and data transfers during processor accesses. A TAP write to the Data register is ignored when there is no DMSEG access pending.

Address Register (TAP Instruction ADDRESS or ALL)



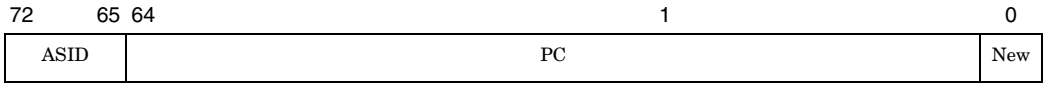
The read-only Address register provides the address for a processor access.

EJTAG Control Register (ECR) (TAP Instruction CONTROL or ALL)



- **Rocc** - This bit must be cleared to acknowledge that the reset was detected.
- **Psz** - Indicates the size of a pending processor access, in combination with the Address register.
- **Doze = 0** (No low power mode)
- **Halt = 0** (Internal bus clock does not stop)
- **PerRst = 0** (Peripheral reset not implemented)
- **PRnW** - Indicates read or write of a pending processor access.
- **PrAcc** - Indicates a pending processor access and controls finishing of a pending processor access.
- **PrRst** - Implemented (core reset supported)
- **ProbEn** - Controls whether the probe handles accesses to the dmseg segment through servicing of processors accesses.
- **ProbTrap** - Controls location of the debug exception vector.
- **EjtagBrk** - Requests a Debug Interrupt exception to the processor when this bit is written as 1.
- **DM** - Indicates if the processor is in Debug Mode.

PC Sample Register Format (TAP Instruction PCSAMPLE)



- **ASID** - Address Space ID of the sampled PC.
- **PC** - Program counter value.
- **New** - Processor writes a 1 to this field whenever a new sample is written into this register. The probe replaces with a zero when it reads out the sample value.

EJTAG Boot Indication

EJTAG Boot is Implemented.

Bypass Register

Implemented. The Bypass register is a 1-bit read-only register, which provides a minimum shift path through the TAP.

Fastdata Register

Implemented. The width of the Fastdata register is 1-bit. During a Fastdata access, the Fastdata register is written and read, i.e., a bit is shifted in and a bit is shifted out.

4.14 cnMIPS Core Pipelines

The cnMIPS cores are dual-issue processors. The instruction classes are:

Arith	= CPU Arithmetic (Table 4–4, except Imul*/Idiv listed below), CPU Logical Instructions (Table 4–8), CPU Instruction Control (Table 4–6), CPU Insert/Extract Instructions (Table 4–9), CPU shift (Table 4–11)
BrJ	= CPU Branch and Jump Instructions (Table 4–5), CPU Trap Instructions (Table 4–12)
Load/Store	= CPU Load, Store, and Memory Control Instructions (Table 4–7, except ULD,ULW,USD,USW)
ULoad/UStore	= ULD,ULW,USD,USW
Cmov	= MOVN, MOVZ
MFCOP0	= MFC0, DMFC0, RDPGPR, WRPGPR, ERET, DERET, SDBBP, RDHWR
MTCOP0	= MTC0, DMTC0, CACHE, DI, EI, TLB*, WAIT
COP2	= DMFC2, DMTC2
Imul	= MUL, DMUL, VMULU, MTHI, MTLO, MTM*, MTP*, POP, DPOP, CLZ, CLO, DCLZ, DCLO
Imul2	= MULT, MULTU, DMULT, DMULTU, MTM0, MFHI, MFLO
Imul3	= V3MULU
Imul4	= VMM0
Idiv	= DDIV, DDIVU, DIV, DIVU

Pipeline 0 can issue instructions of any class:

Class	Result Latency (cycles)	Unit Busy (cycles)	Comment
Arith	1	1	Fully pipelined
BrJ		2+	Taken branch takes at least 2 cycles.
Load/Store	2	1	Fully pipelined. Load-result latency is two cycles. Store operations have no result. Hardware can issue a load or store operation on any cycle.
ULoad/UStore	3	2	Takes two cycles on pipe 0, no instruction can issue on pipe 1 the same cycle as the first.
Cmov	2	1	Fully pipelined
MFCOP0	4	1	Fully pipelined
MTCOP0		3	No other instruction can issue for three cycles.
COP2	4	1	Fully pipelined
Imul	5	1	Fully pipelined, No other Imul*/Idiv can issue this cycle.
Imul2		2	No other Imul*/Idiv can issue for two cycles.
Imul3	5	3	No other Imul*/Idiv can issue for three cycles.
Imul4	5	4	No other Imul*/Idiv can issue for four cycles.
Idiv		72	No other Imul*/Idiv can issue for 72 cycles.

Pipeline 1 can issue instructions of this class:

Class	Result Latency (cycles)	Unit Busy (cycles)	Comment
Arith	1	1	Fully pipelined
Cmov	2	1	Fully pipelined
Imul	5	1	Fully pipelined, no other Imul*/Idiv can issue this cycle.
Imul2		2	No other Imul*/Idiv can issue for two cycles.
Imul3	5	3	No other Imul*/Idiv can issue for three cycles.
Imul4	5	4	No other Imul*/Idiv can issue for four cycles.
Idiv		72	No other Imul*/Idiv can issue for 72 cycles.

4.15 Special MUL Topics

The following sequence can be used to save/restore multiplier context.

```
// save multiplier context
la        $ka, multiplier_context
v3mulu    $v0, $0, $0    // multiply by zero so p2-p0 will get walked out
v3mulu    $v1, $0, $0
sd        $v0, 0($ka)
v3mulu    $v0, $0, $0
sd        $v1, 8($ka)
ori       $v1, $0, 1
v3mulu    $v1, $v1, $0    // now multiply by 1 so p2-p0=mpl2-mpl0
sd        $v0, 16($ka)
v3mulu    $v0, $0, $0
sd        $v1, 24($ka)
v3mulu    $v1, $0, $0
sd        $v0, 32($ka)
mflo      $v0
sd        $v1, 40($ka)
mfhi      $v1
sd        $v0, 48($ka)
sd        $v1, 56($ka)

// restoring multiplier context
la        $ka, multiplier_context
ld        $v0, 56($ka)
ld        $v1, 48($ka)
mthi      $v0
mtlo      $v1
ld        $v0, 40($ka)
ld        $v1, 32($ka)
mtm2      $v0
ld        $v0, 24($ka)
mtm1      $v1
ld        $v1, 16($ka)
mtm0      $v0
ld        $v0, 8($ka)
mtp2      $v1
ld        $v1, 0($ka)
mtp1      $v0
mtp0      $v1

mul rd, rs, rt
  Fully pipelined
  5 cycle latency

dmul rd, rs, rt
  Fully pipelined
  5 cycle latency

vmulu rd, rs, rt
  Fully pipelined
  5 cycle latency

v3mulu rd, rs, rt
  Next mul/div can issue in 3 cycles
  5 cycle latency
```

```
vmm0 rd, rs, rt
  Next mul/div can issue in 4 cycles
  5 cycle latency

mtm0 rs, rt
  Next mul/div can issue in 2 cycles

mtm1 rs
  Fully pipelined

mtm2 rs
  Fully pipelined

mtp0 rs
  Fully pipelined

mtp1 rs
  Fully pipelined

mtp2 rs, rt
  Fully pipelined

mult(u) rs, rt
  Next mul/div can issue in 2 cycles

dmult(u) rs, rt
  Next mul/div can issue in 2 cycles

madd(u) rs, rt
  Next mul/div can issue in 4 cycles

msub(u) rs, rt
  Next mul/div can issue in 4 cycles

div(u) rs, rt
  Next mul/div can issue in 72 cycles

ddiv(u) rs, rt
  Next mul/div can issue in 72 cycles

mflo rd
  Fully pipelined
  5 cycle latency

mfhi rd
  Fully pipelined
  5 cycle latency

mtlo rs
  Next mul/div can issue in 2 cycles

mthi rs
  Next mul/div can issue in 2 cycles

pop rd,rs
  fully pipelined
  5 cycle latency

dpop rd,rs
  fully pipelined
  5 cycle latency

clz rd,rs
  fully pipelined
  5 cycle latency

clo rd,rs
  fully pipelined
  5 cycle latency

dclz rd, rs
  fully pipelined
  5 cycle latency

dclo rd, rs
  fully pipelined
  5 cycle latency
```

4.16 COP2 Latencies

Table 4–31 lists the latencies (i.e. the number of clock cycles until the next issue to the unit) of the COP2 class of instructions (DMFC2 and DMTC2, refer to Table 4–3). If a COP2-class instruction is not listed in Table 4–31, it has a latency of one cycle.

Table 4–31 COP2 Latencies

Operation	Unit	Number of Cycles
CVM_MT_3DES_DEC	3DES/KAS	25
CVM_MT_3DES_DEC_CBC	3DES/KAS	25
CVM_MT_3DES_ENC	3DES/KAS	25
CVM_MT_3DES_ENC_CBC	3DES/KAS	25
CVM_MT_AES_DEC_CBC1	AES	See Table 4–32
CVM_MT_AES_DEC1	AES	
CVM_MT_AES_ENC_CBC1	AES	See Table 4–32
CVM_MT_AES_ENC1	AES	
CVM_MT_CRC_BYTE	CRC	1
CVM_MT_CRC_BYTE_REFLECT	CRC	1
CVM_MT_CRC_DWORD	CRC	2
CVM_MT_CRC_DWORD_REFLECT	CRC	2
CVM_MT_CRC_HALF	CRC	1
CVM_MT_CRC_HALF_REFLECT	CRC	1
CVM_MT_CRC_VAR	CRC	2
CVM_MT_CRC_VAR_REFLECT	CRC	2
CVM_MT_CRC_WORD	CRC	1
CVM_MT_CRC_WORD_REFLECT	CRC	1
CVM_MT_CRC_POLYNOMIAL	CRC	4
CVM_MT_CRC_POLYNOMIAL_REFLECT	CRC	4
CVM_MT_GFM_XORMUL1	HSH/GFM	36
CVM_MT_HSH_STARTMD5	HSH/GFM	148
CVM_MT_HSH_STARTSHA	HSH/GFM	100
CVM_MT_HSH_STARTSHA256	HSH/GFM	149
CVM_MT_HSH_STARTSHA512	HSH/GFM	181
CVM_MT_KAS_ENC	3DES/KAS	17
CVM_MT_KAS_ENC_CBC	3DES/KAS	17

Table 4–32 lists the latencies of the COP2 instructions that use the AES unit, since they depend on key length, encrypt or decrypt, and whether the operation is the first after a key load.

Table 4–32 AES Unit Latencies

AES Key Length	First Operation After Key Load?	Encrypt/ Decrypt	Number of Cycles
128-bit	No	—	31
	Yes	Decrypt	141
	Yes	Encrypt	65
192-bit	No	—	37
	Yes	Decrypt	165
	Yes	Encrypt	74
256-bit	No	—	43
	Yes	Decrypt	206
	Yes	Encrypt	102

4.17 cnMIPS Core Hardware Debug Features

The cnMIPS core's hardware debug support includes the traditional MIPS debug features as well as the MIPS EJTAG features. *Refer to EJTAG Specification, Revision 3.10*. The traditional MIPS debug features provide compatibility with existing debuggers. The cnMIPS™ core's EJTAG support includes cnMIPS core-specific extensions that enable concurrent multicore debugging.

Non-EJTAG cnMIPS™ Core Hardware Features

The traditional MIPS debug features are listed in Table 4–33, and the EJTAG debug features are listed in Table 4–34.

Table 4–33 MIPS Debug Features

Register	Implemented / Comments
BREAK instruction	Implemented
Trap instructions	Implemented
Watch registers	1 inst-only, 1 data-only implemented
MMU Traps	Implemented

EJTAG Hardware Debug Features:

The EJTAG hardware-debug features are listed in Table 4–34.

Table 4–34 EJTAG Debug Features

Register	Implemented / Comments
EJTAG dmseg	Implemented
EJTAG drseg	Implemented
EJTAG inst breakpoints	4 implemented
EJTAG data breakpoints	4 implemented, both address and value. The EJTAG data value breakpoints are not precise for loads.
EJTAG single step	Implemented
EJTAG debug interrupts	Implemented. No external wire, only through the TAP.

CN50XX includes a single JTAG external (i.e. pin) interface for EJTAG support, though each core has it's own EJTAG TAP controller.

4.17.1 Multicore Debug Support

The cnMIPS core hardware supports very fast debug interrupts, called multicore debug (i.e. MCD) interrupts, to aid debugging parallel applications.

The hardware includes 3 global wires, called MCD wires, that each core can pulse every cycle. Each core also samples the “wired-OR” value, each wire every cycle. When this value is a one, each core that is not mask-disabled (via Multicore Debug[Mask_MCD0,Mask_MCD1,Mask_MCD2]) sets a state bit corresponding to the wire (i.e. sets Multicore Debug[MCD0,MCD1,MCD2]).

If any of Multicore Debug[MCD0,MCD1,MCD2] are non-zero on a given core (and the core is not already in debug mode), a debug exception is requested on the core. Software can clear Multicore Debug[MCD0, MCD1, MCD2] by writing a one to them; this is necessary to ensure that no further debug interrupts occur after exiting the debug handler.

These MCD interrupts occur at the same priority level as the debug interrupts (i.e. DINT) described in the MIPS spec. The exception location is also the same as a debug interrupt and MCD0, MCD1, and MCD2 bits are similar to the DINT bit, but the detailed behavior is different:

- The DINT bit is read-only, but Multicore Debug[MCD0, MCD1, MCD2] must be cleared by the debug handler
- DINT must be clear when Multicore Debug[DExcC] is set, but Multicore Debug[MCD0,MCD1,MCD2] need not be.

There are three ways that the global MCD wires can be pulsed:

1. Software can write a one to the Pls0, Pls1, or Pls2 bits to pulse any combination of the three MCD wires.
2. If Multicore Debug[GSDB] is set and a SDBBP instruction is executed, the hardware pulses the MCD0 wire.
3. If IBCn/DBCn[TE] is set and a breakpoint matches, the hardware pulses the MCD0 wire. The following table describes the detailed behavior on a breakpoint match based on the IBCn/DBCn[BE,TE] value:

Table 4–35 Breakpoint Match Behavior

BE	TE	Comment
0	0	Nothing happens on a match
0	1	MCD0 is pulsed on a match. BS bits are also set in IBS/DBS. No direct local exception occurs. (This mode may not be used.)
1	0	A local breakpoint exception occurs due to the breakpoint match, causing the core to enter debug mode. MCD0 is not pulsed. BS bits are set in IBS/DBS. (This mode will be used when debugging, but not multicore.)
1	1	A local breakpoint exception occurs due to the breakpoint match, causing the core to enter debug mode. MCD0 is also pulsed. BS bits are also set in IBS/DBS. (This mode will be used when debugging multicore.)

The debug exception handler can follow these rules to determine the cause(s) of a given debug exception after reading the Debug and/or Multicore Debug registers:

1. Any of Multicore Debug[MCD0,MCD1,MCD2] could be set at any time, indicating that the corresponding MCD state bit is set.

2. If Multicore Debug[DExcC] is set, all of Debug[DDBSImpr, DDBLImpr, DINT, DIB, DDBS, DDBL, DBp, DSS] will be clear, and Debug[DExcCode] will contain a valid code. (This is the case for a debug mode exception.)
3. If none of Debug[DDBSImpr,DDBLImpr, DINT, DIB, DDBS, DDBL, DBp, DSS] are set, then the exception was either due to MCD*, or Multicore Debug[DExcC] being set and Debug[DExcCode] is valid.
4. No more than one of Debug[DIB, DDBS, DDBL, DBp, DSS] can be set.
5. If Multicore Debug[DExcC] is clear, any combination of Debug[DDBLImpr, DINT] may be set.
6. At least one of Debug[DDBLImpr, DINT, DIB, DDBS, DDBL, DBp, DSS] and Multicore Debug[MCD0, MCD1, MCD2, DExcC] will be set.

4.17.2 System Debug Characteristics

Note that Multicore Debug[CvDM, CGSTP, CGSTP] bits determine whether the Count and CvmCount registers are affected by the GSTOP state, and the watchdog timers may be affected by the GSTOP state, as described in Central Interrupt.

Though cnMIPS cores implement the CountDM and STOPEN functions to make the core instruction flow while debugging as similarly as possible to the non-debug instruction flow, there still are many ways that debug timing and instruction flows will differ from non-debug. No other CN50XX hardware is affected by debug mode or GSTOP: packets may continue to arrive, hardware coprocessor units will continue to operate, and external interrupts (e.g. PCI) may arrive while the cores are in debug mode and/or GSTOP is enabled.

Another significant factor affecting debug/nondebug instruction flow is the impact of delayed load/IOBDMA operations. The cores can issue these delayed load/IOBDMA operations as follows:

- Fetch & Add Unit (FAU) requests have the tagwait feature. With this feature, the hardware attempts to delay servicing the FAU request until after the last pending tag switch occurs. Refer to “Fetch and Add Unit (FAU)” on page 148.
- New work requests have a wait feature. With this feature, the hardware attempts to delay the new work request until work is available.

These delayed load/IOBDMA operations have time-outs that fail the request. The time-out is necessary to ensure that subsequent SYNCIOBDMA, SYNCIOALL, or SYNC requests do not hang forever. (If they did hang, the core would no longer be able to execute any instructions.) These time-outs are necessarily shorter than any of the other time-outs so that the cores can respond quickly to interrupts, exceptions, and any other flow redirections. During debug it may not only be probable for these time-outs to expire, but also perhaps necessary for these time-outs to expire, to make forward progress. Refer to “Forward Progress Constraints” on page 251. This may cause considerable changes to the instruction flow while debugging. Tagwait FAU requests may often time-out when debugging, but never time-out during normal operation. Refer to “Fetch and Add Unit (FAU)” on page 148.

Note that, the core-debug-mode handler need not use noncacheable references to avoid invalidating load-linked/store-conditional sequences, as the MIPS specifications suggest. The debug handler can use ordinary cached loads/stores without invalidating load-linked / store conditional sequences.

4.18 cnMIPS Core Load-Linked / Store-Conditional

LL and LLD instructions to DRAM set the lock flag. The hardware clears the lock flag under any of the following conditions:

- Another core writes the same (128-byte) cache block, causing the block to be invalidated from this cache.
- ERET execution
- SC/SCD execution
- TLBWI/TLBWR execution
- A store to the block using a different virtual address than was used by the LL/LLD.
- A store to the locked block (same virtual address as was used by the LL/LLD) that has a late exception (TLB modified, address error, TLB invalid, TLB modified, watchpoint, or breakpoint), but hit in the cache.
- Increase in CvmMemCtl[LMEMSZ].
- Anything that invalidates the entire Dcache:
  - CACHE (op == 1, 9, 17, 21)
  - Status[ERL] change

LL/LLD instructions to I/O space do not set the lock flag and act exactly as LW/LD, respectively.

SC/SCD instructions to I/O space always fail, no write ever occurs (and clear the lock flag).

4.19 cnMIPS Core Exceptions

Table 4–36 lists all the MIPS-defined and Cavium Networks-specific exceptions, and whether they are implemented on the cnMIPS core.

Table 4–36 cnMIPS Core Exceptions

Exception	Implemented/ Not Implemented	Comments
Reset	Implemented	
Soft Reset	Implemented	cnMIPS cores can be reset from a chip-wide soft reset (CIU_SOFT_RST[SOFT_RST]), watchdog time-out expiration, core-selected soft reset (CIU_PP_RST[RST,RST0]), or core-local EJTAG TAP soft reset (ECR[PrRst]). In the core-selected and core-local soft-reset cases (where only the local core is reset), the coherent memory bus and other coherent bus users are not reset. If the core being reset is actively using the bus the chip behavior could be adversely affected. Before performing a core-selected or core-local soft reset, ensure that the core being reset has executed a SYNCIOBDMA instruction, and has entered an idle loop.
Debug Single Step	Implemented	
Debug Interrupt (DINT)	Implemented	cnMIPS cores can receive core-selected debug interrupts (CIU_DINT[DINT]), EJTAG TAP ECR[EjtagBrk] debug interrupts, and can take a debug interrupt from EJTAGBOOT.
Multicore Debug (MCD) Interrupt	Cavium-specific.	Multicore debug interrupts are very similar to DINT, except that Debug[DINT] is not set after the interrupt is taken. (MultiCoreDebug[MCD*] is set instead.)
Debug Data Break Load/Store Imprecise (DDBLImpr/ DDBSImpr)DDBLImpr	DDBLImpr implemented DDBSImpr not implemented.	cnMIPS core's EJTAG breakpoints are all precise except for load value matches.
Non-Maskable Interrupts	Implemented	cnMIPS core may receive core-selected non-maskable interrupts (CIU_NMI[NMI]) or may receive a non-maskable interrupt from the CIU watchdog for the core.
Machine Check	Implemented	cnMIPS core implements synchronous machine-checks. Debug mode machine-checks can occur.
Interrupt	Implemented for Cause[IP4,IP3,IP2].	cnMIPS cores receive two interrupt wires from CIU - Cause[IP4,IP3,IP2]. The CIU combines all possible chip-wide interrupt wires into Cause[IP4,IP3,IP2]. This includes: per-group work queue interrupts, GPIO interrupts, 2 mailbox (inter-core) interrupts, UART interrupts, PCI interrupts, PCI MSI interrupts, watchdog interrupts, TWSI interrupts, RML interrupts, trace buffer interrupts, GMX/IPD packet-drop interrupts, key zeroization interrupts, general timer interrupts. Cause[IP2], Cause[IP3], and Cause[IP4] can be programmed independently to include any combination of these interrupt sources.
Deferred Watch	Implemented	
Debug Instruction Break	Implemented	CN50XX may take an instruction breakpoint for the instruction in the delay slot of a branch-likely instruction, even though the branch was taken, and therefore the instruction was not really executed. (Note that branch-likely instructions are deprecated in the MIPS architecture and generally should not be used.)
Watch on Instruction fetch	Implemented	CN50XX may take a watch on an instruction fetch for the instruction in the delay slot of a branch-likely instruction, even though the branch was taken, and therefore the instruction was not really executed. (Note that branch-likely instructions are deprecated in the MIPS architecture and generally should not be used.)
Address error in Instruction fetch	Implemented	The cnMIPS core may flush the Icache on an address error on an instruction fetch. Debug mode address error instructions can occur.
TLB/XTLB Refill on Instruction Fetch	Implemented	Debug mode TLB/XTLB refill on instruction fetch exceptions can occur.

Table 4–36 cnMIPS Core Exceptions

Exception	Implemented/ Not Implemented	Comments
TLB Invalid on Instruction Fetch	Implemented	Debug mode TLB invalid on instruction fetch exceptions can occur.
Cache error on Instruction Fetch	Implemented	The cnMIPS core automatically corrects the error by flushing the Icache, so the cache error handling software need not attempt to correct the error. The Cache error register (register = 27, select = 0) contains diagnostic information. Debug-mode cache error on instruction fetch exceptions can occur.
Bus error on Instruction Fetch	Implemented	CN50XX instruction fetch bus errors only occur on a time-out of an outstanding instruction fetch fill. These errors are generally non-recoverable. Debug mode bus error on instruction fetch exceptions can occur.
Debug breakpoint (SDBBP execution)	Implemented	Debug mode debug breakpoint exceptions can occur.
Coprocessor Unusable	Implemented	CN50XX may generate Coprocessor unusable exceptions for all of Coprocessors 0, 1, and 2. Debug mode coprocessor unusable exceptions can occur.
MDMX Unusable	Not Implemented	CN50XX does not implement the MDMX ASE.
Reserved Instruction	Implemented	Debug mode reserved instruction exceptions can occur.
Integer Overflow	Implemented	Debug mode Integer overflow exceptions can occur.
Trap	Implemented	Debug mode Trap exceptions can occur.
System Call	Implemented	Debug mode system call exceptions can occur.
Breakpoint	Implemented	Debug mode breakpoint exceptions can occur.
Floating Point	Not Implemented	cnMIPS™ cores do not implement floating-point.
Coprocessor 2	Not Implemented	cnMIPS core's coprocessor 2 units do not generate exceptions.
Precise Debug Data Break	Implemented	All debug data break exceptions are precise except for load value match breakpoints.
Watch on Data Access	Implemented	

Table 4–36 cnMIPS Core Exceptions

Exception	Implemented/ Not Implemented	Comments
Address Error on Data Access	Implemented with Cavium-specific additions.	When CvmCtl[REPUN] is clear, the core generates address errors for the LD, LW, LWU, LH, LHU, SD, SW, and SH instructions exactly as per the MIPS specs. When CvmCtl[REPUN] is set, address error exceptions due to address alignment in physical memory or CVMSEG LM (NOTE: NOT I/O or DSEG) locations do not happen for the LD, LW, LWU, LH, LHU, SD, SW, and SH instructions since the core hardware automatically completes the memory unaligned loads/ stores. When CvmCtl[REPUN] is set, the core generates address errors for all unaligned IO/DSEG accesses for the LD, LW, LWU, LH, LHU, SD, SW, and SH instructions, as per the MIPS specifications. However, the priority of these unaligned address errors is lower than the priority of TLB refill, TLB invalid, and TLB modified exceptions when CvmCtl[REPUN] is set, unlike the MIPS specifications. When CvmCtl[USEUN] is set, the core generates address errors for all unaligned IO/DSEG accesses for the ULD, ULW, USD, and USW instructions. The priority of these unaligned address errors is lower than the priority of TLB refill, TLB invalid, and TLB modified exceptions. When enabled (by the appropriate CvmMemCtl[CVM*]), CVMSEG I/O references by anything other than an aligned SD to the 0xFFFF FFFF FFFF A200 virtual address (i.e. any non-IOBDMA) cause address errors. When enabled (by the appropriate CvmMemCtl[CVM*]), CVMSEG LM references larger than that allowed by CvmMemCtl[LMEMSZ] cause address errors. Stores that take an address error solely due to the limit imposed by CvmMemCtl[LMEMSZ] may corrupt other cache locations, however, since the core hardware does not stop the store sufficiently early. When enabled (by the appropriate CvmMemCtl[CVM*]), all LL, LLD, SAA, SAAD, SC, and SCD accesses to CVMSEG cause address errors. Debug mode address error on data access exceptions can occur.
TLB/XTLB Refill on Data Access	Implemented	Debug mode TLB/XTLB Refill on Data Access exceptions can occur.
TLB Invalid on Data Access	Implemented	Debug mode TLB Invalid on Data Access exceptions can occur.
TLB Modified on Data Access	Implemented	Debug mode TLB Modified on Data Access exceptions can occur.
Cache Error on Data Access	Implemented	The cnMIPS core automatically corrects the error by flushing the Dcache, so the cache error handling software need not attempt to correct the error. The Cache error register (register = 27, select = 1) contains diagnostic information. Debug mode Cache Error on Data Access exceptions can occur.
Bus error on Data Access	Implemented	Data access bus errors only occur on a time-out of an outstanding Dcache fill. These errors are generally non-recoverable. Debug mode Bus error on Data Access exceptions can occur.
Precise Debug Data Value Compare Break	Not implemented	Debug Data Value Compare breakpoints are imprecise on the core.

Chapter 5

Packet Order / Work Unit (POW)

This chapter contains the following subjects:

Overview

The Packet Order/Work (POW) unit is a CN50XX coprocessor providing the following important functions:

Work Queueing

Work is described by an associated work-queue entry and may be created by either hardware units or core software. The centralized packet-input hardware creates a work-queue entry and submits work for each packet arrival. Core software can create work-queue entries and submit work as desired. The packet output (PKO), Timer (TIM), and PCI hardware units can also submit work-queue entries created by core software after completing an operation/instruction.

POW implements eight input work queues. The different work queues can be used to provide different service levels.

The input work queues can be infinitely large, overflowing to DRAM when necessary.

Work Scheduling/Descheduling

Core software requests work from POW. POW selects (i.e. schedules) work for the core and returns a pointer to the work-queue entry that describes the work to core software. This off-loads much overhead from the cores and the coherent memory bus.

All work is not equal since the POW hardware supports 16 groups. Each piece of work has an associated group identifier. A configuration variable for each core specifies the groups that the associated core will accept when it requests work. This configuration variable is a 16-bit bitmask, one bit per group, so core software can specify all possible combinations of groups. The POW hardware does not schedule a piece of work if the core doesn't accept the group associated with the work.

Groups provide a means to execute different functions on different cores, though all cores share the same POW hardware. For example, packet processing may be pipelined from one group of cores to another group of cores, with the first group performing the first stage of work and the next group performing the next stage of work.

A core can deschedule a piece of work. Deschedule means that the software running on this core will not complete the work at this time, and the POW hardware should reschedule it later. The POW hardware reschedules previously descheduled work at higher priority than it schedules new work from an input queue. Deschedule can be useful in a number of circumstances:

- It can transfer work from one core group to another. This is one mechanism to implement “work pipelining”.
- It can avoid consuming a core for work that requires a large synchronization delay.
- It can make work interruptible.

Ordering and Synchronization of Work

The POW hardware associates a 32-bit tag value and a tag type with each piece of work. The work-queue entry and the request to add work to an input work queue contain the initial tag value. (This initial tag value may be created by either the centralized input packet processing hardware or by core software.) Core software can

also later switch the tag/type as the work progresses through different phases of the application. If the same tag value sequences are used by two packets, the packets are ordered.

There are three different tag types:

- **ORDERED** - Ordering is guaranteed with this tag type. (Atomicity is not.)
- **ATOMIC** - Ordering and atomicity are guaranteed with this tag type. Two pieces of work holding the same ATOMIC tag cannot be scheduled simultaneously.
- **NULL** - No ordering is guaranteed with this tag type, and work cannot be in-flight (with respect to the POW hardware) with this tag type.

The POW hardware, in combination with core software, uses these tag/type values to order and synchronize related work, and allow unrelated work to be unordered and unsynchronized. This is essential for efficient multi-core execution. Two pieces of work may be / are related and will be ordered and synchronized when they share the same tag value and tag type. Two pieces of work may be unrelated and will execute entirely in parallel when they have different tag or tag type values.

For example, the tag value may be a hash of the standard TCP five-tuple (IP source address, IP destination address, IP protocol, TCP source port, TCP destination port) defining a “flow”. The same flow will have the same tag value, so it may be ordered and synchronized. Different flows will likely have different tag values, so will likely not be ordered and synchronized, and can be executed completely in parallel on different cores.

At different code phases, core software can change the tag value via a tag switch transaction with separated switch request and switch completion wait operations. The POW hardware completes a requested switch when the required ordering and atomicity constraints for the work are met. This separated switch transaction allows software to overlap the switch request latency with other profitable work and also allows software the option to deschedule the work while a tag switch is pending, thus avoiding long synchronization delays.

5.1 POW Work Flow, Operations, and Ordering

Figure 5–1 shows an abstracted view of the POW unit, focusing on the states of work as it flows through the POW Unit, and touching on the most important operations that cores can execute to transform work. Often, work flows through the states in the figure from top to bottom - work is first in the input queues, then in-flight, and finally descheduled or completed.

At any given time, only one piece of work can be scheduled to a particular core. This is shown in the center of Figure 5–1 (see “in unit, sched”). Clearly, the number of scheduled items is limited to the number of cores (up to 2). A core is not scheduled if it is executing unscheduled work or if it completes scheduled work without requesting new work. Scheduled work is a subset of the POW in-flight work. Any core may deschedule a scheduled item at any point (see “in unit, desched” in Figure 5–1). Descheduled work remains in-flight, and will be rescheduled later, but is not currently executing on a core. Work that is scheduled remains scheduled after the completion of a tag switch transaction (SWTAG), unless the switch has a next tag state of NULL. A tag switch with a next tag switch to NULL causes work to immediately become unscheduled on the core. Figure 5–1 abstracts SWTAG transactions by showing them as a single arrow; in reality, a deschedule operation can occur after SWTAG transaction starts, but before it completes.

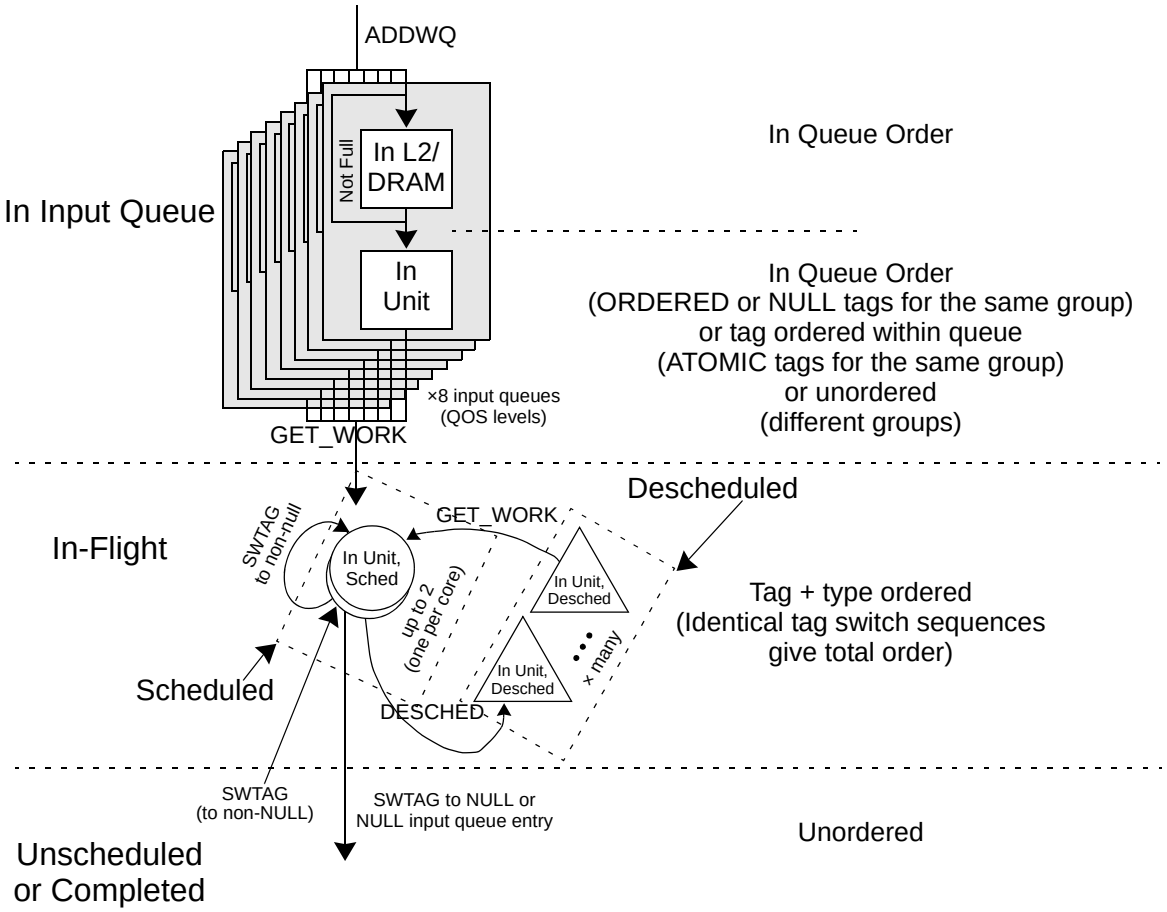


Figure 5–1 States of Work Flowing Through the POW

Work typically enters the POW unit through one of the input queues. The top of Figure 5–1 shows the eight POW input work queues. The POW unit internal entries are shared by in-flight work and work in input queues. Both software and hardware can add input queue entries (ADDWQ). Though the POW unit size is limited, the POW hardware maintains the illusion of an infinite input work queue. When space is not available in the POW unit, the POW hardware adds the input queue entries to an L2/DRAM list maintained by hardware. If space is available in the POW unit when work is added, the POW hardware buffers the work internally immediately and avoids the overhead of the memory list. If the POW hardware puts work in a memory list, it later automatically (and in the background) moves the work from L2/DRAM into the unit as soon as space becomes available in the unit, in the order that the work was originally added.

Work is typically scheduled to a core when core software executes a GET_WORK transaction to request new work. The POW hardware can schedule in-unit input queue entries to cores in response to these requests. The POW hardware can also schedule descheduled work to cores in response to GET_WORKs. The POW hardware prioritizes descheduled work above input queue work. The POW scheduler never schedules a descheduled item that has a pending tag switch, and never schedules an input queue entry with an ATOMIC tag unless it can immediately have the tag. In other words, POW only schedules work when it can make forward progress. Input work-queue entries with the NULL tag type are a special case. The POW hardware immediately unschedules NULL type input queue work returned for a GET_WORK.

Work also enters the POW unit when an unscheduled (from the perspective of the POW hardware) core executes a SWTAG transaction. This is shown by the upward arrow in the bottom of Figure 5–1. Work that enters the POW hardware unit this way is immediately scheduled, and is then not distinguishable from other scheduled work.

Figure 5–1 also shows another interesting aspect of the POW hardware on the right side. The ordering guarantees for work as it flows through the POW unit.

First, if work is in an input queue in memory (at the top in Figure 5–1), POW keeps it strictly in-order on a per queue basis.

Second, when work is in-flight (either scheduled or descheduled), ordering is strictly based on tag and tag type values. POW does not force any ordering nor synchronize in-flight work that uses different tag values or different tag type values. This in-flight work freely executes in parallel.

NOTE: The group identifier of work does not affect the ordering of in-flight work, it only affects the cores to which a descheduled item can be rescheduled.

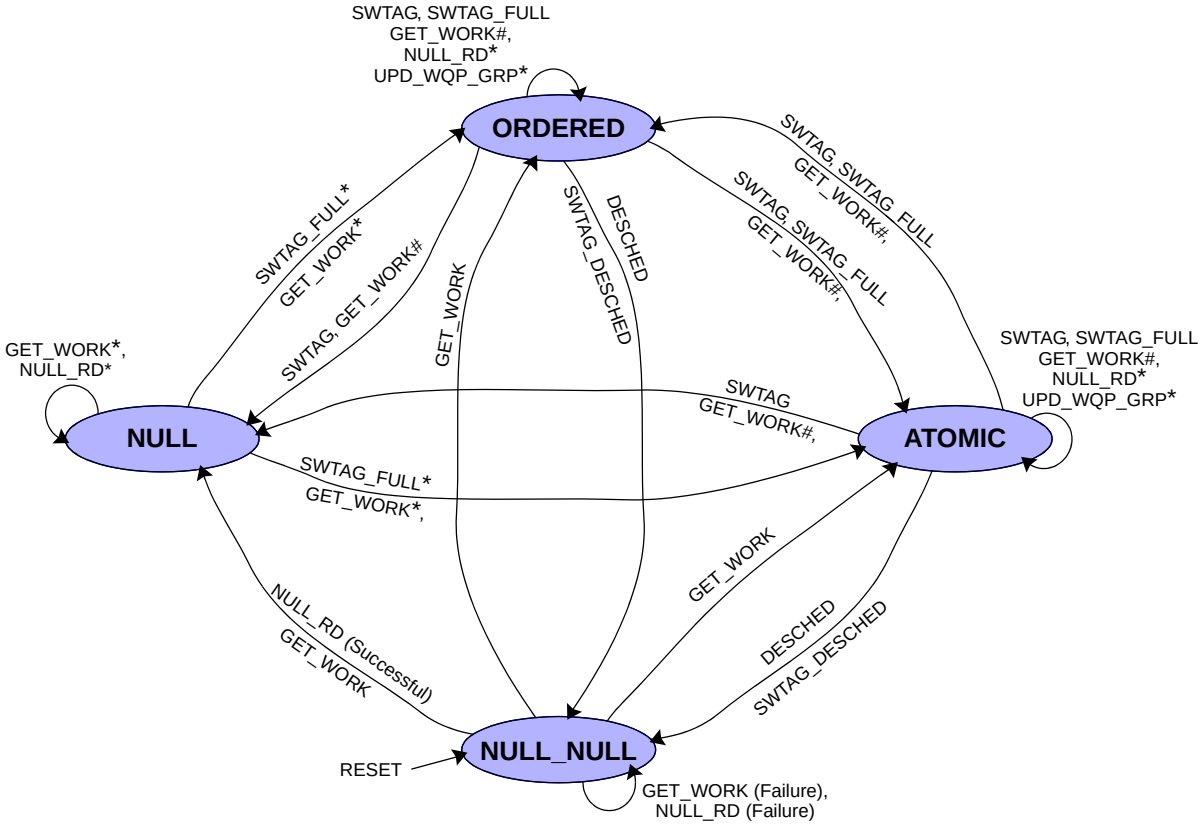
Third, when work is both in an input queue and in the POW unit (i.e. between the memory input queues and the in-flight work), the work ordering guarantees are mixed. The POW hardware work scheduler skips past in-unit input queue entries that cannot be immediately scheduled when it searches for schedulable work. The POW scheduler never skips past ORDERED and NULL input queue work, so the POW scheduler schedules work with these types (and the same group) strictly in per input queue order. The POW scheduler skips input queue work with the ATOMIC tag type and a tag that cannot immediately be scheduled, and so only guarantees tag order for ATOMIC input queue work (that has the same group). The POW work scheduler skips over input queue work that is not in the desired group, so no ordering is implied between two input queue entries in different groups. Finally, at the bottom of Figure 5–1, unscheduled work is not synchronized by POW hardware and so is completely unordered.

The POW hardware maintains order across tag switches. Any in-flight work that executes the identical series of tag switches (each with the same tag/type values) while in-flight will be ordered identically through each switch. With proper configuration and software support, the POW hardware can totally order the processing of all packets in a flow. CN50XX provides total per-flow work ordering support for input packets (perhaps all the way to output) as long as the following conditions are true: (1) All packets from the same flow enter POW via the same input queue with the same initial tag value and group, and (2) The software processes packets from the same flow with the same sequence of non-NULL tag switches.

Figure 5–2 depicts an abstracted view of the POW core state (i.e. the POW states visible to a core) and the operations that affect it, focusing on the legal major operations in each state. A state has an arc tagged with a particular operation when it is legal to issue the operation in the state.

NOTE: The POW ADDWQ and NOP commands do not affect POW core state and are legal at any time. The POW CLR_NSCHED command does not affect POW core state and has its own issue rules.

The abstracted states in Figure 5–2 closely mirror the tag types available in the tag switch operation. A new work request that receives work with an ORDERED, ATOMIC, or NULL tag puts the core into the ORDERED, ATOMIC, or NULL POW



*** = can start while a SWTAG or SWTAG_FULLFULL is pending to the state**

 # = hardware inserts an implicit switch to NULL prior to the GET_WORK

Figure 5–2 The POW States Visible to a Core

core state, respectively. A tag switch to an ORDERED, ATOMIC, or NULL tag type puts the core in the ORDERED, ATOMIC or NULL POW core state, respectively. These operations are depicted by arcs entering these states.

Figure 5–2 introduces the NULL_NULL state and the NULL_RD transaction. NULL_NULL is a special state entered only after a deschedule or reset. NULL_NULL and NULL_RD are required because a deschedule operation detaches an internal POW entry from a core and there may not be another entry available. (See Section 5.3 for more description of the POW hardware internals). NULL_NULL is similar to NULL, with the clear difference that it is illegal to SWTAG when in the NULL_NULL state. NULL_RD causes the POW hardware to attempt to convert the state to NULL when it is in the NULL_NULL state. (NULL_RD will fail when there are no more internal POW entries — see Section 5.10 for forward-progress implications.)

The GET_WORK arcs exiting the ORDERED and ATOMIC states are special and marked with “#”. These transactions implement multiple functions to release the prior work and schedule new work for this core. The POW hardware actually executes an implicit switch to NULL before executing the GET_WORK in these two cases. This implicit switch to NULL releases the prior work, so that the hardware always starts a GET_WORK from the NULL or NULL_NULL states. Note the implication that a GET_WORK from ORDERED or ATOMIC that does not

successfully return work will change to the NULL state. A GET_WORK transaction from NULL_NULL that does not successfully return work will stay in the NULL_NULL state.

As in Figure 5–1, Figure 5–2 abstracts SWTAG, GET_WORK, and NULL_RD transactions as a single arc, though all these operations can have separate request and completion times. This is because the initial request solely determines the legal operations that can follow. The only question is whether the next legal transaction can start before the POW hardware completes the previous transaction. A tag switch transaction has explicitly-separated request and completion operations, but the get work and Null Rd transactions are separated only with core IOBDMAs. IOBDMAs are described in Section 4.7. Here are the rules regarding transaction start time for the POW transactions that affect POW core state:

- SWTAG_DESCHED, DESCHED, UPD_WQP_GRP, and NOP transactions do not have separated start and completion times, so can be followed immediately by any legal command.
- The transactions marked "*" can start before the prior SWTAG is complete.
- In all other cases in Figure 5–2, A following transaction must not start before the prior transaction is complete.

The hardware behavior is unpredictable when the rules evident in Figure 5–2 are violated by core software. Note some specific restrictions:

- It IS NOT LEGAL to initiate a deschedule from the NULL or NULL_NULL POW core state.
- It IS NOT LEGAL to initiate any tag switch from the NULL_NULL state.
- It IS NOT LEGAL to initiate a tag switch with tag type of NULL from the NULL POW core state.
- It IS NOT LEGAL to issue any tag switch or get work operation while there is a pending switch with an ORDERED or ATOMIC tag type.
- It IS NOT LEGAL to initiate any transaction while a get work transaction is pending.
- It IS NOT LEGAL to initiate any transaction while a Null Rd operation is pending.
- It IS NOT LEGAL to initiate a SWTAG_FULL or SWTAG_DESCHED transaction with tag type of NULL.

Table 5–1 details the available POW operations:

Table 5–1 POW Operations

Operation	Description
ADDWQ(tag_type, tag, wqp, grp, qos)	This adds work to the input queue selected by the QOS. Tag_type can legally be ATOMIC, ORDERED, or NULL. QOS is a 3-bit value, grp is a 4-bit value, and tag is a 32-bit value. The work-queue pointer (wqp) must be a 64-bit aligned pointer into L2/DRAM and must point to a legal work-queue entry. See "Work-Queue Entry Format" on page 220. Furthermore, the work-queue entry group, tag type, and tag fields in the work-queue entry in L2/DRAM must exactly match the corresponding values supplied with the ADDWQ or the POW hardware may produce unpredictable results. The input work-queues are infinite, so this transaction never fails.

Table 5–1 POW Operations (Continued)

Operation	Description
GET_WORK(wait)	This transaction attempts to get work for the requesting core. The value of the POW_PP_GRP_MSK0/1[GRP_MSK] CSR for the core at the time of the GET_WORK specifies the groups that are acceptable. The wait option causes the POW hardware to delay responding to the request until either work becomes available or the request times out. In any case, the POW hardware returns a failure response if it was unable to find work for the core, or a pointer to the work-queue entry if it successfully found work for the core. **NOTE:** It is possible, though unlikely, for a time-out to occur when the wait bit is clear, as well as when the wait bit is set, if the work search takes too long. The POW_PP_GRP_MSK that specifies the acceptable groups for a core must not be written between the start and completion of the GET_WORK, or unpredictable results may occur. Otherwise, the CSR can be written at any time. The POW_NW_TIM[NW_TIM] CSR specifies the configurable time-out counter interval that controls a single counter used for both cores. The POW hardware times out a GET_WORK request after two interval timer expirations, so the effective time-out interval varies between one and two times the configured interval.
NULL_RD	This transaction attempts to change to the NULL state when in the NULL_NULL state. It is a NOP from all other states. The POW hardware will return NULL_NULL when it could not successfully allocate an internal POW entry. Successful NULL_RDs always leave the core in the NULL state. Unsuccessful NULL_RDs or ones converted to NOPs do not change the core state.
SWTAG(new_tag_type, new_tag)	This starts a tag switch transaction. The POW hardware completes the tag switch transaction later when it clears the pending switch bit for the core (refer to Section 5.5 for more information onPOW tag switch-pending indications). An exception is a SWTAG to NULL, whose completion POW hardware never transmits. A SWTAG must not be used when switching from the NULL state. • A SWTAG from an ATOMIC tag releases the ATOMIC tag immediately once the tag switch transaction starts, perhaps long before the SWTAG transaction completes. • A SWTAG to an ATOMIC tag completes when the work acquires the new ATOMIC tag. At most one piece of work holds an ATOMIC tag at any time. The FIFO order is the acquisition order for the tag. • A SWTAG from an ORDERED tag cannot complete until all work ordered earlier in the old tag's FIFO start a SWTAG transaction. • A SWTAG to an ORDERED tag occurs immediately when switching from an ATOMIC or NULL tag, and occurs once the ordering constraints of the old tag are met when switching from an ORDERED tag.
SWTAG_FULL(new_tag_type, new_tag, new_wqp, new_grp)	This is identical to SWTAG, except that the transaction additionally updates the work-queue pointer (new_wqp) and 4-bit group identifier (new_grp) for the work that is held in the POW. SWTAG_FULL must be used for all switches from the NULL state. The POW hardware never interprets or uses the work-queue pointer supplied in this transaction, but it may deliver it to SW later to complete a GET_WORK. The POW hardware stores <35:3> of the work-queue pointer. SWTAG_FULL must not be used for switches to NULL.

Table 5–1 POW Operations (Continued)

Operation	Description
DESCHED(no_sched)	This executes a deschedule transaction. When the no_sched bit is set on DESCHED (or SWTAG_DESCHED) operations, the POW hardware does not schedule the packet to a core until a subsequent CLEAR_NSCHED operation clears the no_sched bit for the POW entry. The POW entry can be determined with a POW status load with get_cur=1 prior to the DESCHED (refer to Section 5.11.1). The index field in <50:40> identifies the POW ID. Note that it is recommended that the core be in ATOMIC state rather than ORDERED state at the time of the DESCHED. (See the "POW Performance Considerations" on page 228, below.)
SWTAG_DESCHED(new_tag_type, new_tag, new_grp, no_sched)	This is identical to a SWTAG followed by a DESCHED, except that it also updates the group identifier. It must follow the same start rules as does SWTAG (shown in Figure 5–2). SWTAG_DESCHED is well-suited for transferring work from one group to another - work pipelining.
UPD_WQP_GRP (new_wqp, new_grp)	Update the work-queue pointer (new_wqp) and group identifier (new_grp) for the work that is held in the POW.
CLR_NSCHED (wqp, index)	Clears the nosched bit for the POW entry selected by index. CLR_NSCHED is a NOP under any the following conditions: • the POW entry is not on a deschedule list, or • the wqp in the POW entry does not match the supplied wqp Before initiating a CLR_NSCHED operation, software must guarantee that all *DESCHEDs and CLR_NSCHEDs are complete. software can read the pend_desched and pend_nosched_clr bits via POW status loads to determine when these conditions are true. (Refer to Sections 5.8 and 5.11.1 for more details on POW status loads.) After a CLR_NSCHED operation, software must guarantee that the CLR_NSCHED is complete before issuing any subsequent POW operations. It can do this by checking the pend_nosched_clr via POW status reads. Note also that index will typically be determined by POW status loads prior to the *DESCHED that set the no_sched bit. A POW status load with get_cur=1 returns the index field in <50:40>.
NOP	No Operation

5.2 Software Architecture Example

This section describes one specific example of software usage of some of the POW hardware features. We describe this example only to illustrate a possible usage of some of the POW hardware features, not to advocate any of the particular techniques. The POW hardware synchronization support is very flexible and can be used in a huge variety of ways.

Figure 5–3 shows the simple synchronization architecture for the Firewall/VPN packet processing example. This simple architecture assumes that there are at most six application phases during the processing of each individual packet: defrag, IPSEC decrypt, Lookup, Process, IPSEC encrypt, and output queue. It also assumes IPv4, but could be modified/generalized to IPv6.

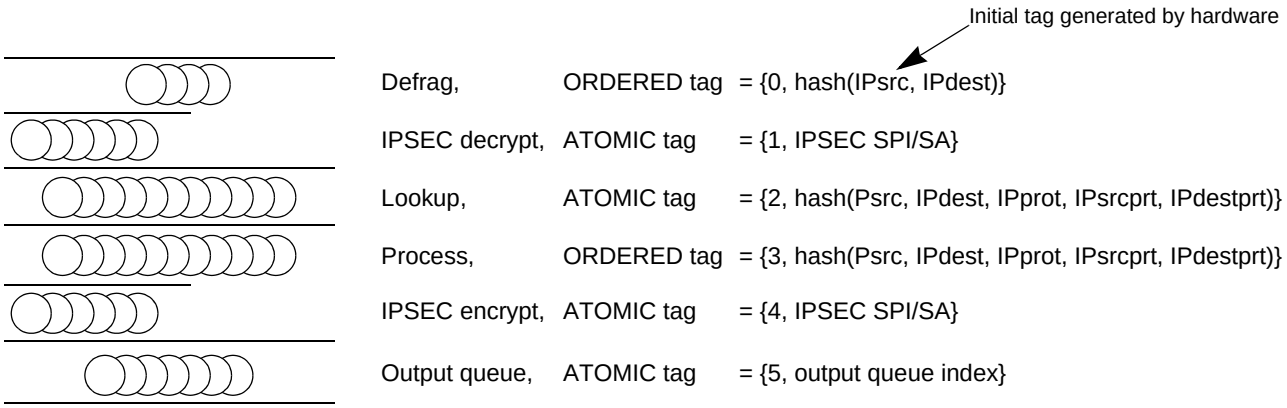


Figure 5–3 Architecture for the Firewall/VPN

The example of software architecture in Figure 5–3 also uses some of the possible 32 tag bits to force unique tags in the different processing phases. For example, software could choose to partition the 32 tag bits in the following way:

Phase ID - Unique identifier for the phase
- 0x0 = Defrag
- 0x1 = IPSEC Decrypt
- 0x2 = Lookup,
- 0x3 = Process
- 0x4 = IPSEC encrypt
- 0x5 = Output queue

Phase TAG - tag value for within the phase

This phase tagging is not required by the hardware. It guarantees that different software processing phases execute in parallel in a pipelined fashion. The different PHASE ID values guarantee different tag values, so packets execute simultaneously with any other packet that is in a different phase. Of course, multiple packets can also execute freely within the same flow if they have different PHASE TAG values.

As is always possible with the POW hardware, the different PHASE TAGs architected in Figure 5–3 guarantee processing order among all packets in the same flow (and the same direction). Some phases also have ATOMIC tags to serialize access to shared data structures in some of the phases. Generally speaking, the phase tag value for each phase in the figure is either:

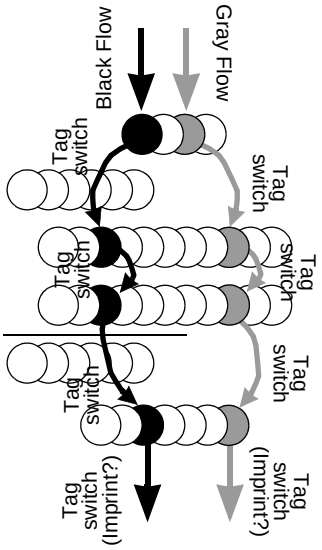
a. A hash result using the maximum number of packet fields to differentiate flows as much as possible, but still guarantee that two packets from the same flow produce the same hash value, or

b. An index into a critical data structure, or

c. BOTH of the above.

The goal is to expose many different tag values to the POW hardware so it can parallelize as much work as possible, yet still guarantee strict in-order processing for packets in the same flow, and still guarantee that access to critical data structures is appropriately synchronized.

Figure 5–4 depicts the execution sequence using this software architecture for packets from two different flows: the gray flow and the black flow that pass through to different output queues without any IPSEC processing by CN50XX. The Figure 5–4 example execution shows that the two different flows select different PHASE TAG values in every phase. Thus, the POW hardware will schedule the packets from the two flows entirely in parallel.



Initial tag generated by hardware

Defrag,	ORDERED tag	= {0, hash(IPsrc, IPdest)}
IPSEC decrypt,	ATOMIC tag	= {1, IPSEC SPI/SA}
Lookup,	ATOMIC tag	= {2, hash(Psrc, IPdest, IPprot, IPsrcprt, IPdestprt)}
Process,	ORDERED tag	= {3, hash(Psrc, IPdest, IPprot, IPsrcprt, IPdestprt)}
IPSEC encrypt,	ATOMIC tag	= {4, IPSEC SPI/SA}
Output queue,	ATOMIC tag	= {5, output queue index}

Figure 5–4 Software Execution Sequence

A tag switch defines the transition from one phase to the next in Figure 5–4, though the cores can work on the packet during a tag switch transaction and make the boundaries fuzzy. The packets from the gray and black flows execute four different phases to complete a packet, and four different tag switches occur for each packet.

The software need not issue a tag switch transaction for the first phase since the hardware creates the proper initial tag (“Packet Input Processing/Input Packet Data Unit (PIP/IPD)” on page 265 describes the programmability of the centralized Packet Input Processing and Input Packet Data Unit (PIP/IPD) that generates the initial tags), and the POW hardware only schedules the work after it has the initial tag. Though the last switch (to NULL that removes the work from POW) must execute inside POW, software need not specifically issue it since it can be an implicit part of the get work request for the next packet. So this architecture requires exactly three tag switches to execute the example pass-thru packets.

Note also that in the example execution in Figure 5–4, the software simply skips the IPSEC phases since no IPSEC is required. As long as all packets in the same flow (and the same direction) skip the same IPSEC phases, the switches in the IPSEC phases can be avoided and the packets in the flow can still be kept in order by the POW hardware. Other flows may require IPSEC processing, so may require up to two additional tag switches.

To explain the assumed ordering and synchronization requirements of the simple example architecture of Figure 5–4 in more detail, in the remainder of this section we briefly describe the functions assumed for each phase, and how the architected tags for each phase give the desired synchronization and ordering behavior.

5.2.1 Defragmentation

This stage defragments input packets. All input packets enter this phase, some of which may be fragmented, and fewer de-fragmented packets exit.

The POW ORDERED tag type architected for this phase does not serialize the work within the defrag phase in any way, so some other means of memory synchronization (e.g. load-linked/store-conditional) would likely be required by software to correctly access the defragmentation data structure.

Any inbound packets in the same flow, fragmented or not, must have identical values for the IP source and destination address fields, and so will have the same PHASE TAG and will be ordered. Both unfragmented and de-fragmented packets from the same flow will enter the next stage in order.

The fragments that enter but do not exit this stage will immediately switch to NULL and not execute in any other phases. When the last fragment creates a completely defragged packet, the de-fragged packet is processed in subsequent phases in place of the input fragment, so the de-fragged packet assumes the work order of the input fragment. This gives the de-fragged packet the ideal ordering; as if it instantly appeared with the last input fragment, even though the software processing time needed to create the defragged packet may be large.

5.2.2 IPSEC Decryption

This stage performs IPSEC decryption for packets that need it. If not needed, this phase is skipped entirely.

In general, inbound IPSEC processing requires some synchronization between different packets that use the same IPSEC tunnel for shared structures, like the anti-replay window. The architecture specifies the simplest possible solution for this synchronization; a single ATOMIC tag that covers the IPSEC SA for the entire packet decryption. The PHASE TAG value uniquely identifies the SA, so after the inbound tag switch completes, the core can freely read and write the tunnel data structure until the next tag switch.

All packets from the same flow will have the same IPSEC SPI/SA value and so will remain ordered as they exit this phase. Packets from other flows that use the same tunnel are also serialized through this phase, and will exit this phase in order, but will likely have a different tag in the next phase so will probably be unordered going into the next phase.

5.2.3 Lookup

This example assumes a stateful firewall, and this stage finds the flow (identified by a 5-tuple in this example) records and updates the state. On the first packet of a flow, a fast-path flow record will not exist for the flow, so the software will need to validate the flow and cache it for the fast-path flow records for later packets. The PHASE TAG architected for this phase selects a particular hash bucket. Thus, the ATOMIC tag serializes accesses to a bucket, and no further synchronization on the hash data structure may be required. No packets from a flow will observe (or modify) the flow state before the previous packets from the flow have updated the fast-path flow record.

With this cached architecture, it will likely take much longer to initially validate and cache the flow state than it will take to process subsequent packets. Thus, some packets may need to wait for the validation to complete before their lookup can continue. With some protocols, like TCP, these waits will not happen within the same flow, but long waits are still possible with other protocols or with packets from a different flow that, unluckily, happen to collide in the same hash bucket. The deschedule operation provided by the POW hardware can be useful in these long-wait circumstances. The software can deschedule the current work and execute other work, and the POW hardware will reschedule the work later when the ATOMIC tag is available.

5.2.4 Process

This stage is for some processing of the packet that may be required before sending the packet out. For example, the packet may require NAT translation or TCP sequence number adjustment.

This stage uses an ORDERED tag with a hashed flow identifier for the PHASE TAG. This will force order for packets in the same flow as they exit this stage, as precisely as possible.

Of course, the firewall software may choose to drop a packet in either the lookup or process stages without executing any further stages.

5.2.5 IPSEC Encrypt

This stage performs IPSEC encryption for packets that need it. If not needed, this phase is skipped entirely.

This stage's tag usage is very similar to the IPSEC decryption stage.

5.2.6 Output Queue

This stage places the packet into an output queue to be sent out. The tag is ATOMIC to synchronize critical data structures needed for the enqueue. The PHASE TAG identifies the exact output queue, so only references that use the same output queue are serialized.

The firewall software may choose to perform QOS calculations (e.g. RED) at this stage to determine whether to drop the packet based upon queue sizes.

5.3 POW Internal Architecture

NOTE: For illustration purposes, Figure 5–5 shows five hypothetical cores. CN50XX has only two cores and would perform the tasks differently.

Figure 5–5 has a conceptual picture of the internal POW architecture. The actual implementation differs for performance reasons. The largest component of the POW unit is the 256 internal POW entries. As the figure shows, POW entries are either in input queues, in-flight, attached to a core that is in the NULL state, or in the free list. Each POW entry contains at least the following information:

- A pointer to the work-queue entry in L2/DRAM (WQP).
- The current tag and tag type
- The current group.
- Pointers to link the entry into various lists.

The left side of Figure 5–5 shows the input queues. There are eight input queues. For each input queue, there is both a memory list and an in-unit list. As the POW hardware adds new work into the unit, either directly from an ADDWQ or from the head of the associated memory queue, it allocates an internal POW entry and fills it with the necessary information. Once the POW hardware allocates an internal POW entry for a piece of work, the work remains in the unit while it is in the input queue or in-flight. The POW hardware cannot overflow to L2/DRAM after that point. Core operations only cause the POW entry to attach/detach to/from particular cores, and move between lists.

NOTE: Once the POW unit loads work into the unit, it no longer reads or writes the work-queue entry locations in L2/DRAM. The work-queue entry in L2/DRAM is only used while work is in an input queue, but never while the work is in the unit. The POW hardware must carry the work-queue pointer along at all points when it is inside the unit, since the work-queue entry pointer is what (indirectly) describes the actual work that needs to be performed.

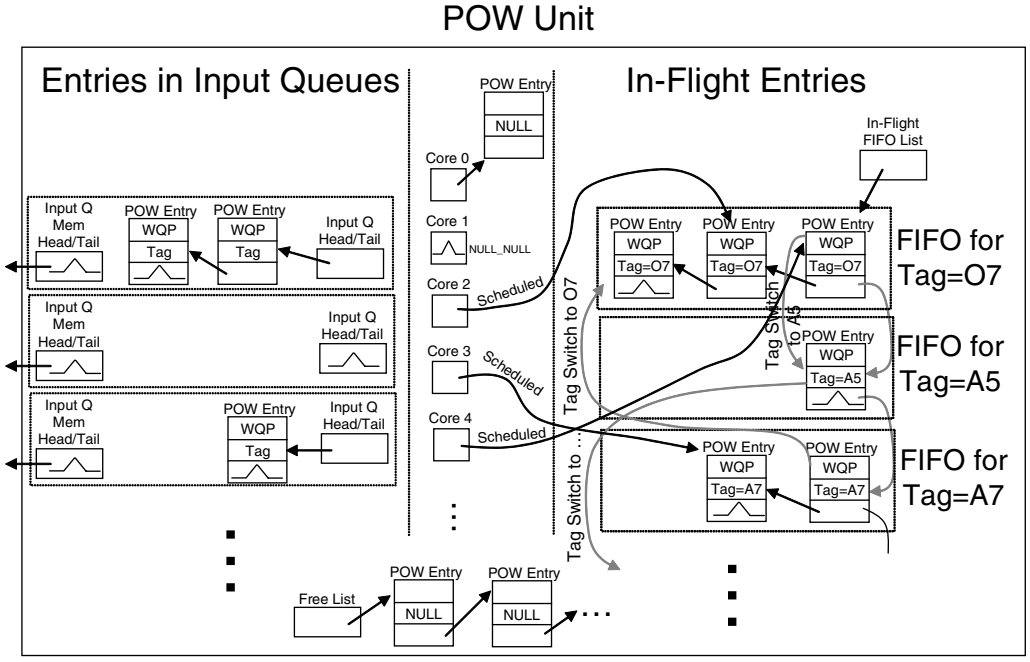


Figure 5–5 Internal POW Architecture Components

The right side of Figure 5–5 shows the in-flight POW entries. Conceptually, in-flight POW entries are always organized in FIFOs, with one FIFO associated with each unique in-flight tag and tag type value combination. For example, Figure 5–5 shows an O7 (tag = 7, type = ORDERED), A5 (tag = 5, type = ATOMIC), and A7 (tag = 7, type = ATOMIC) FIFO. Work first enters a FIFO when the POW hardware either schedules work from an input queue or switches from the NULL core state. SWTAG then causes POW entries to move from FIFO to FIFO. Note that there are no NULL FIFOs; work is no longer in-flight after a switch to NULL.

The FIFO orders in Figure 5–5 imply all ordering and synchronization constraints. For an ORDERED tag, the FIFO order indicates the order of the work holding the tag. For an ATOMIC tag, the FIFO order indicates the order that the work will switch to the ATOMIC tag, and the head of the list is the only work that has successfully switched to the ATOMIC tag. The POW hardware must move the head of one FIFO to the tail of another FIFO as part of any non-NULL tag switch. This FIFO move completes a switch to an ORDERED tag, but for an ATOMIC tag the work must

further become the head of the new FIFO before the switch is complete. The work at the head of a FIFO never has a pending switch, or else the POW hardware would immediately execute the FIFO move required by the switch.

For example, in Figure 5–5, since the O7 FIFO is ORDERED, each of the three pieces of work has successfully switched to the O7 tag. The head of the O7 FIFO cannot have a pending switch, but the remaining work can have a pending switch to a different non-NULL tag. If non-head work has a pending FIFO move (because of a pending non-NULL SWTAG), the move cannot occur until the head moves. The A5 FIFO has only one entry so there can be no pending SWTAGs. The one piece of work holds the ATOMIC tag. In the A7 FIFO, the switch to A7 is complete for the head entry in the FIFO, but is pending for the other work. No work in any ATOMIC FIFO can have a pending SWTAG to a different tag.

The POW hardware implements the in-flight FIFOs as pure first-in, first-out lists, with one exception. The POW hardware can, in some cases, remove work from an ORDERED list out-of-order when the work has a pending SWTAG to NULL.

The middle of Figure 5–5 also depicts the POW core states. A core is either scheduled to in-flight work (i.e. in the ORDERED or ATOMIC states), in the NULL state with an attached POW entry, or in the NULL_NULL state. For example, Core 0 is in the NULL state; it has an attached POW entry, but the entry is not scheduled. Core 1 is in the NULL_NULL state; it neither has an attached POW entry nor is scheduled. Core 2 may currently be in the ORDERED state, or may have a pending switch to another state. Core 3 has a pending switch to the ATOMIC state. Core 4 is in the ORDERED state and has no pending switches.

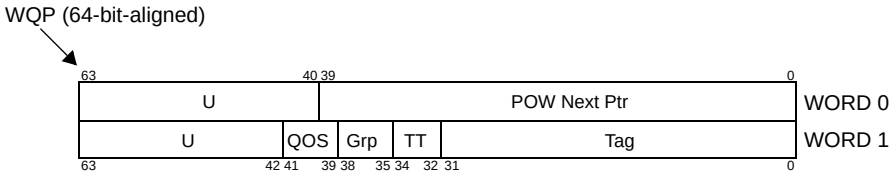
The final POW architectural component shown in Figure 5–5 is the free list. The POW hardware uses the available POW entries to create in-unit input work-queue entries and to successfully complete core NULL_RD requests. If there are no free POW entries, the POW hardware can neither create in-unit work entries nor successfully complete core Null Rd requests for cores in the NULL_NULL state. If there are fewer than 2 free POW entries available, the POW hardware cannot create in-unit input work-queue entries, but can still successfully complete NULL_RD requests. This reservation of the last 2 free-list entries makes it less likely that NULL_RD requests will fail. The POW hardware frees any POW entry previously attached to a core when it successfully completes a GET_WORK for the core. (The core would always have been in the NULL state when a POW entry frees, either because the GET_WORK initiated from the NULL state, or because of a switch to NULL implicitly inserted by the hardware prior to a GET_WORK transaction from ORDERED or ATOMIC.)

5.4 Work-Queue Entry Format

The work-queue entry in L2/DRAM is the primary descriptor that describes work. Though POW hardware does much to assist work queueing, scheduling, and synchronization, it doesn't dictate much about the format of the work-queue entry. POW hardware only uses the L2/DRAM work-queue entry in specific circumstances, and it uses few fields in those cases. The central packet input processing and input packet data unit (PIP/IPD) may be the primary CN50XX hardware component that constrains the work-queue entry format. (Chapter 7 describes the work-queue entries created by PIP/IPD. This section strictly discusses the requirements of the POW unit hardware.)

First, It is only possible for the POW hardware to read/write to the L2/DRAM locations containing the work-queue entry at two times: (1) when processing the ADDWQ, and (2) when moving work from a memory input queue to a POW entry in the unit. At all other times the WQP held in a POW entry need only be a 64-bit aligned pointer (with at least <35:3> stored). The POW hardware stores the pointer and delivers the pointer to the work when a core executes a GET_WORK transaction.

Second, Figure 5–6 depicts the format requirements for the work-queue entry when it is used by the POW hardware. The POW Next Ptr field (in WORD 0) is used by hardware to link work in a memory input queue, so must not be used by software when work is created with ADDWQ. The POW hardware always reads the value of the QOS field in WORD 1 from L2/DRAM when a hardware unit submits an ADDWQ, and so must be set appropriately by software in only that case. The POW hardware may also read the Grp, TT, and Tag fields from L2/DRAM for any ADDWQ. Software must guarantee that these L2/DRAM work-queue entry fields exactly match the same fields also supplied when software submits the ADDWQ transaction, or unpredictable results may occur.



U – Never used by POW hardware

POW Next Ptr – Written and read by POW hardware to implement memory input queue

QOS – Selects the input queue for the work. Read from L2/DRAM by the POW hardware when a hardware unit initiates an ADDWQ.

Grp, TT, Tag – Select the group, tag type, and tag for the work. Read from L2/DRAM by the POW hardware either when a hardware unit initiates an ADDWQ, or when moving work from a memory input queue into the unit.

Figure 5–6 Format Requirements for the Work Queue Entry

5.5 Core and Fetch-and-Add Pending Switch Bits

The core/fetch-and-add(FAU)/POW hardware SWTAG implementation includes switch pending bits held local to a core and inside the Fetch-and-Add unit inside the I/O bridge (IOB) block. The pending switch bits inside a core get set when the core starts a switch transaction by executing a store to a particular (I/O space) physical address destined for the POW hardware (refer to Section 5.11.3). The FAU hardware also sets its pending switch bit for a core when it sees the same store.

The SWTAG implementation also includes a dedicated bus that indicates switch completion. The pending switch bits in the core and in the FAU hardware get cleared by the dedicated switch completion bus.

The FAU hardware uses its pending switch bit to delay selected requests until after a tag switch completes. The core software uses the pending switch bit to determine when the switch completes.

It can locally sample its pending switch bit with the

```
RDHWR rt, $30
```

MIPS instruction, which returns a one when the pending switch bit is clear, and a zero when the pending switch bit is set. Core software must use this instruction to determine that the prior switch transaction is complete.

NOTE: A core can have, at most, one outstanding (non-NULL) switch, so only one pending switch bit is required.

The cores starts SWTAG transactions by executing I/O stores (<48> of the physical address set) with a major did value of 12 (in <47:43> of the physical address) selecting the POW unit. I/O stores with this major did value are POW unit stores. The subDID field is in <42:40> of the physical address.

Here are the precise rules regarding the pending switch bit in a core:

- POW unit stores with subdid equal zero set the bit
- POW unit stores with subdid equal three clear the bit
- A tag switch completion from POW clears the bit

Here are the rules regarding the pending switch bits in the Fetch-and-Add logic:

- POW unit stores with subdid equal zero sets the bit corresponding to the core
- A tag switch completion from POW clears the bit corresponding to the core

A SWTAG transaction starts with a POW unit store to subdid zero that sets the pending switch bits in both the core and the FAU. Normally, the tag switch completes with the tag switch completion from POW on the special switch bus, which clears all the pending switch bits. Here are the exception cases:

- The POW hardware never sends a switch completion for a SWTAG to NULL.
- The POW hardware will not send a switch completion for a piece of work after the core executes a DESCHED transaction to deschedule and detach the work from the core.

Given this pending switch bit behavior, software should use the following rules for POW unit I/O stores:

- Subdid zero should be used to start non-NULL SWTAG and SWTAG_FULL transactions. This sets the pending switch bit in the core and in the Fetch-and-Add unit.
- Subdid three should be used to start all DESCHED and SWTAG_DESCHED transactions. This clears the pending switch bit in the core, ensuring that it is not stale for any subsequent work.
- Subdid one should be used to start ADDWQ, NULL SWTAG, UPD_WQP_GRP, CLR_NSCHED, and NOP transactions. These will affect no pending switch bit.

Core software should not initiate any DESCHED transaction while there is an in-flight Fetch-and-Add request that is waiting for a tag switch completion. (However, it is legal to issue a DESCHED after a Fetch-and-Add request fails due to a time out on a tag switch wait.) Note also that the Fetch-and-Add pending switch bit may conservatively remain set after a DESCHED, so core software should only use the Fetch-and-Add tagwait feature when a tag switch is known to be outstanding.

5.6 POW Interrupts

The highest-performance CN50XX core software will avoid the high cost of interrupts on packet receipt. Instead, this software will poll the POW hardware scheduler to find work. However, the POW hardware does support coalescing interrupts for those applications that require them. The interrupts can occur based on either work counts or time.

The goal of the POW interrupt hardware is to produce interrupts when work is present and can be executed by a core. All work is not considered equal with respect to the interrupts generated by the POW hardware. Rather, the POW hardware accounts for and controls interrupts on a per-group basis. Software can configure and accept interrupts separately for each core.

There are two distinct ways a piece of work in a particular group is “executable” in the POW unit:

- IQ executable
- DS executable.

IQ Executable

The work is present in one of the 8 POW input work queues and is held in one of the 256 entries inside the POW unit. Note specifically that work held in one of the 8 input queues, but not yet held inside the POW unit (i.e. resides in L2/DRAM), is NOT considered “executable”. This is because the POW hardware cannot schedule work to a core if it is not currently in the POW unit.

Note that though work was/is IQ executable and an interrupt was/is generated, it is still possible for a subsequent interrupt handler request for work to fail to return work from the interrupting group. Here are the ways this is possible:

1. Of course, if the interrupt handler did not include the interrupting group as an acceptable one (in POW Core Group Mask Registers) before the request, the hardware cannot return work for the interrupting group. Also, if the interrupt handler requests work from more than just the interrupting group, the POW hardware may return work from a different group.

2. This core (or any other core using the same group) may have been given new work from the group while the hardware was generating the interrupt, but before the interrupt handler requests work.

3. IQ work with an atomic tag may be constantly "IQ executable", though have a tag conflict with some other piece of in-flight work that constantly prevents the POW hardware from scheduling it.

4. The POW hardware may time out if it is forced to search too deep from among the DS or IQ executable items to supply work for a core.

   This can happen while attempting to acquire a particular "IQ Executable" piece of work if there is:

   - too many work items for different groups "earlier" in the in-unit input queue(s), or
   - too many work items for the same group, but with an atomic tag conflict "earlier" in the in-unit input queue(s).

The next section describes the POW hardware scheduling search priority/order and will explain the concept of "earlier" in the input queue(s).

DS Executable

The work was previously descheduled, and is now ready to be rescheduled again.

DS executable work is generally higher priority than IQ executable work since the POW hardware schedules previously descheduled work before it schedules new input queue work. Also, tag conflicts with descheduled (or other in-flight) work can prevent work (in this or any other group) from executing/proceeding, unlike IQ executable work.

Note that a descheduled item with the no_sched bit set is not considered to be executable, as it cannot be scheduled to a core.

Here is the work-queue interrupt logic inside POW that is replicated for each of the 16 groups:

- A counter of the number of work-queue entries that are IQ executable (POW_WQ_INT_CNT(0..15)[IQ_CNT]) and an associated interrupt threshold (POW_WQ_INT_THR(0..15)[IQ_THR]).
- A temporary IQ executable interrupt threshold disable bit. (POW_WQ_INT[IQ_DIS<*>]). With the IQ executable interrupt threshold disable bit, software can prevent continual interrupts when the IQ executable count exceeds its threshold, but cannot be reduced due to cases (3) or (4) outlined above.

  While the disable count bit is set, the threshold check on the IQ executable count cannot generate an interrupt until the next time expiration. This bit can be written to one by software, and is cleared by hardware. The hardware clears the IQ executable interrupt threshold disable bit under these conditions:

  a. The IQ executable count is zero for the group, or
  b. The hardware decrements the time counter for this group to zero.

- A counter of the number of work-queue entries that are DS executable (POW_WQ_INT_CNT(0..15)[DS_CNT]) and an associated interrupt threshold (POW_WQ_INT_THR(0..15)[DS_THR]).

- A time counter (POW_WQ_INT_CNT(0..15)[TC_CNT]), an associated time threshold (POW_WQ_INT_THR(0..15)[TC_THR]), and a time threshold interrupt enable bit (POW_WQ_INT_THR(0..15)[TC_EN]).

  The time counter decrements whenever either the IQ executable count is non-zero or the DQ executable count is non-zero for the group.

  The hardware sets the time counter to the threshold value whenever:

  a. both the IQ and DS counts are zero, or
  b. the interrupt bit for this group is written with a 1 (to clear), or
  c. the time threshold value is written, or
  d. the temporary IQ executable interrupt threshold disable bit is written to a one (to set it), or
  e. the time counter is zero

- An interrupt bit (POW_WQ_INT[WQ_INT<*>]).

  This bit is set by hardware and can be cleared by software. The POW hardware sets this bit under any/all of the following three conditions:

  a. The IQ executable count is >= its threshold and the IQ threshold check is enabled (the threshold check is disabled when the threshold is zero or when the temporary IQ executable interrupt threshold disable bit is set).
  b. The DS executable count is >= its threshold and the DS threshold check is enabled (the threshold check is disabled when the threshold is zero).
  c. It decrements the time counter to zero and the time count check is enabled.

The POW hardware decrements the per-group time counters at a frequency selected by a single counter (POW_WQ_INT_PC[PC]) and associated period (POW_WQ_INT_PC[PC_THR]) that is shared by all groups. The POW hardware loads the counter with POW_WQ_INT_PC[PC_THR]*256 + 255 initially and whenever POW_WQ_INT_PC is written, and decrements the counter once every cycle otherwise. When the counter reaches zero, the hardware reloads the shared counter and decrements each of the per-group time counters by one if the associated IQ executable or DS executable counts are non-zero, whether the time threshold interrupt is enabled or not.

As touched upon earlier in this section, software must handle any discrepancies that occur from the fact that the IQ executable count for a group may not match the number of successful work requests for the group that follow. Here are some possible ideas, not requirements, to alleviate possible interrupt forward-progress issues if they arise:

- Use the temporary IQ executable interrupt threshold disable bit to prevent continual interrupts from cases (3) or (4) above.
- Use only the time threshold interrupts and not the executable count threshold interrupts to avoid continual interrupts from cases (3) or (4).
- Use dedicated input queues containing only interrupt traffic and/or dedicated input queues containing only specific groups to reduce the probability of problems with (4) above.
- Don't use atomic tags for the interrupt word in the IQ if (3) above is a problem.

5.7 POW QOS Features

The POW hardware contains a number of quality-of-service (QOS) features to prioritize input queues differently and keep work flowing through the unit smoothly. The main features are thresholds and scheduling.

5.7.1 Thresholds

Having thresholds is a QOS feature to limit the number of in-unit POW entries consumed by individual input queues (i.e. to limit POW_QOS_THR*n*[BUF_CNT]). In-unit POW entries are consumed by work in input queues, in-flight work. In-flight work may be descheduled, and POW_QOS_THR*n*[DES_CNT] indicates the total number of descheduled work items in all groups. POW_QOS_THR*n*[FREE_CNT] indicates the number of available POW entries - a total number available for all groups.

The POW hardware implements the following thresholds separately for each input queue:

- A maximum number of in-unit work-queue entries (POW_QOS_THR*n*[MAX_THR] + 1) for the queue. POW_QOS_THR*n*[BUF_CNT] will not exceed this threshold. If there are more than POW_QOS_THR*n*[MAX_THR] + 1 entries in the queue, those entries must reside in L2/DRAM, not in the POW unit.
- A minimum number of free in-unit work-queue entries (POW_QOS_THR*n*[MIN_THR]) for the queue. If the number of available in-unit entries (i.e. POW_QOS_THR*n*[FREE_CNT]) is smaller than POW_QOS_THR*n*[MIN_THR] for the queue, no more entries from the queue will be loaded into the unit.

5.7.2 Scheduling

The second QOS feature is the mechanisms to control the input-queue traversal. The POW hardware traverses all input queues in parallel when it is searching for new input-queue work to schedule to the cores. The POW hardware scheduling can implement, on a core-by-core basis, mixtures of static priority (based on the input queue) and weighted round-robin priority. This can programmably prevent cores from receiving work from particular input queues.

When scheduling work to a core, POW hardware first scans the list of descheduled work. If a previously descheduled, currently schedulable item exists for a group that that the core resides in, the POW hardware schedules this work to the core. If such descheduled items are not available, the POW hardware attempts to schedule work from an input queue to the core in the following manner.

The input-queue search proceeds in configurable rounds. The software can configure the 32 rounds via the POW_QOS_RND*n*[RND_P*] fields. Each configured round is an 8-bit bit-mask indicating the input queues that participate in the round. In general, the more rounds that a queue can participate in (as selected by the POW_QOS_RND*n*[RND*] configuration), the higher is the priority of the queue.

The following pseudo-code conceptually describes the scheduling algorithm that the POW hardware implements for a GET_WORK operation when schedulable input-queue work is selected for the core:

```
get_work(core) {

for(iq = 0; iq < 8; iq++) {
   size[iq] = POW_QOS_THR[iq].BUF_CNT;         // # POW entries IQ has in unit
   pri[iq] = POW_PP_GRP_MSK[core].QOS<iq>_PRI; // priority of this IQ for this core
   if(pri[iq] == 0xF) size[iq] = 0;            // IQ may be disabled for the core
   if(size[iq]) wqe[iq] = first WQE in iq;
}
found_pri = 0xF;
round = current_round;
iq = current_iq;
grp_msk = POW_PP_GRP_MSK[core].GRP_MSK;

do {
   // check if this iq participates in this round
   mask = (POW_QOS_RND[round / 4] >> ((round & 0x3) * 8)) & 0xFF;
   if((mask & (1 << iq)) && size[iq]) {
     // check if this is the best candidate for this core, or:
     //   - the core is in the selected group, and
     //   - there are no (atomic) conflicts with in-flight work, and
     //   - we have not already found work of the same or higher priority
     if((grp_msk & (1 << wqe[iq].grp)) &&
        ((wqe[iq].type != ATOMIC) || no_matching_inflight_atomic_tag(wqe[iq].tag)) &&
        (pri[iq] < found_pri)) {
       found_wqe = wqe[iq];
       found_pri = pri[iq];
     }

     wqe[iq] = next WQE in iq;
     size[iq] --;
   }

   // advance to the next round/iq
   if(iq == 0) {
     iq = 7;
     round = (round + 1) % 32;
   }
   else
     iq--;
   // we are done once we find a work-queue entry
   // at the highest possible priority
   for(i = 0; i < 8; i++) {
     if(size[i] && (pri[i] < found_pri))
       break;
   }
   found = (i == 8) && (found_pri < 0xF);
} while(!found);

current_round = round;
current_iq = iq;
return(found_wqe);
}
```

The POW hardware scans the input queues starting at a current round and the current input queue. The remaining queues that participate in the current round are sequentially scanned for work first. After that, the rounds are scanned circularly. Each input queue is scanned in order, and different input queues are scanned in the order indicated by the round configuration.

The POW hardware selects the highest-priority schedulable input queue work for the core. Whenever the POW hardware schedules work from an input queue to a core, it updates the current queue/round so that the next work request starts from the queue/round that sequentially follows the position where the last schedulable work was found. The current queue/round is shared by all cores.

These are some notable aspects of the input queue scheduling algorithm:

- The POW hardware can only schedule in-unit work-queue entries, whose count for a given input queue is indicated by POW_QOS_THR*n*[BUF_CNT]

- Each core can reside in any combination of groups, indicated by POW_PP_GRP_MSK*n*[GRP_MSK]
- Each core can establish its own individual priority for each input queue, indicated by POW_PP_GRP_MSK*n*[QOS*_PRI]. The hardware can find work more quickly when the core uses fewer priorities, as the higher-priority queues must be traversed in entirety before lower-priority work can be scheduled.
- POW_QOS_RND*n* configuration always indicates the order that the input queues are traversed, regardless of the selected priorities.
- Weighted-round-robin is used between queues of the same priority. The POW_QOS_RND*n* configuration establishes the weighting.

  For example, the highest-weight queue might participate in all 32 rounds, while a low-weight queue might participate in only one of the 32 rounds. Assuming there are schedulable items in both these equal-priority-example queues, the POW hardware schedules 32 items from the high-weight queue and then schedules one item from the low-weight queue.

5.8 POW Debug Visibility

The POW hardware provides complete visibility into its current state through POW load operations. This visibility can be used for POW debug dumps and for probing current status. Section 5.11 describes the details of the POW status loads, POW memory loads, and POW index/pointer loads that return the POW state. This section describes them qualitatively.

Figure 5–5 figuratively indicates much of the POW state. The more detailed list is:

- 256 POW entries, linked in various in-unit queues and tag lists (i.e. the FIFO for a given tag)
- eight in-unit input queues, one per QOS level
- 16 external (i.e., overflow) input queues, two per QOS level
- 16 in-unit descheduled head queues, one per group
- a nosched descheduled head queue
- 2 core states, one per core
- a free list for POW entries
- a 2-entry reserved free queue[1] for servicing NULL_RD requests from the cores.
- a prefetched entry[1] used for ADDWQs.
- four prefetched entries[1] used for moving input queue entries from L2/DRAM into the unit.

POW status loads return the current state for individual cores. The POW core state includes:

- State bits indicating the operations that the POW is currently attempting to process. For example, there is a bit indicating that this core has a pending switch and has not left the original tag list.
- The current state / tag type of the core, as depicted in Figure 5–2.

1 These 7 entries are not directly visible to software. All other POW entries can be accounted for by the in-unit input queues, free list, core-headed tag lists, or tag lists associated with the descheduled heads, though. The prefetched and reserved entries reside among the remaining unaccounted-for POW entries.

- The current work-queue pointer, group, and tag for the POW entry attached to the core (if any POW entry is attached).
- The next/pending work-queue pointer, group, tag type, and tag (if there are pending operations).
- Head and tail bits, forward and reverse links to maintain the tag list that the POW entry is in (if any POW entry is attached).

Note that when a POW entry is attached to a core, the POW core state supercedes the POW entry status. (When the POW hardware attaches a POW entry to a core, it loads the POW entry state into the POW core state and does not update the POW entry state until it detaches it from the core.)

POW Memory loads return the current state of an individual POW entry through POW memory loads. The state of a POW entry includes:

- The current work-queue pointer, group, tag type, and tag of the POW entry.
- A bit indicating that the POW entry has a pending switch and has not left its original tag list, together with the tag type and tag for the new tag list.
- A forward link for the input, free, or descheduled head list that the POW entry is currently on (if any).
- A tail bit and forward link for the tag list that the POW entry is currently on (if any).
- A nosched bit.

POW index/pointer loads return the heads and tails for the following lists:

- the 8 in-unit input queues.
- the 16 external input queues, and which is the current head for reloading. (The hardware alternates between the two memory queues associated with a given input queue when it reloads. This is a higher-performance implementation than would be a single input-memory list per QOS level, as it allows the hardware to traverse the two input-memory lists in parallel.)
- the 16 descheduled head queues.
- the nosched descheduled head queue
- the free list.

To get a consistent snapshot of the POW state, it is best to capture it while all POW activity and operations have ceased. If these types of captures are not possible, then capture the POW state as quickly as possible to get a view that is as consistent as possible. Do this by first reading the core state, then the heads/tails of the various lists of POW entries, and then traversing the lists.

Note that for the POW hardware to correctly service the visibility (and CSR) reads, software must restrict itself to at most one outstanding load/IOBDMA inflight from each individual core. This is most simply done by only using loads to read the visibility information, and never using IOBDMAs.

5.9 POW Performance Considerations

The features provided by the POW hardware are so integral to the use of CN50XX that its use can have important performance effects. This section describes how the POW performance interacts and influences, actually determines how other on-chip components will work.

First, the POW hardware does not ideally reschedule descheduled work with an ORDERED tag. The POW hardware can reschedule only the head of an ORDERED tag FIFO in some cases, though the ORDERED tag semantics allow it to schedule all the work in the FIFO. Core software is recommended to always use ATOMIC tags when descheduling. If ORDERED behavior was desired, core software can immediately switch to an ORDERED tag after it receives the rescheduled work.

Second, some notes on the raw CN50XX hardware latencies, as it is desirable for software to overlap the latencies with other useful operations:

- CN50XX requires approximately 40 cycles from the start of a SWTAG until it completes.
- CN50XX requires approximately 60 cycles from the start of a GET_WORK until it completes.

Third, some notes on the POW hardware behavior for the performance-critical SWTAG and GET_WORK transactions:

- SWTAG transactions are highest priority and always complete in near minimum latency (excluding bus contentions transferring the request to the unit) if there are no synchronization/ordering conflicts.
- The POW hardware prefetches work for both cores during cycles that would otherwise be idle. Thus, GET_WORK transactions can complete in near minimum latency (excluding bus contentions that transfer the load/IOBDMA to/from the unit) in many cases. However, changes to POW_PP_GRP_MSK*n* CSRs and SWTAG transactions both invalidate the prefetch.
- GET_WORKs can be slower for cores that cannot accept all groups.
- GET_WORK prefetches are slower when the prefetcher must skip over a number of input-queue entries to find work for the core. This can be a problem when input queue work has a conflicting ATOMIC tag, or for cores that cannot accept the group of the work in the input queue. Some applications may find it advantageous to avoid ATOMIC tags in the input queue. It may also be desirable to restrict work for particular groups to particular input queues, since this can ensure that there is some work for all/more groups at or near the head of an input queue, and will make work for all or more of the acceptable groups more easily visible to the prefetcher.

5.10 Forward Progress Constraints

Care is necessary to guarantee that core software makes forward progress when it uses POW synchronization features. This section gives examples of some pitfalls and suggests some rules for core software to use that can avoid forward progress problems. These may not cover all circumstances, but touches on all the sources that can limit forward progress.

First, multi-constraint synchronization deadlocks must be avoided by core software. The POW hardware alone cannot be a source of multi-constraint synchronization deadlocks since each piece of work, and each core, has at most one active tag with a well-defined set of ordering / synchronization constraints. Of course, it is always possible for core software to create deadlock conditions if it combines POW synchronization constraints with other synchronization/locking constraints. (In CN50XX, core software can synchronize through means other than provided by the POW hardware, like load-linked/store-conditional sequences and Fetch-and-Add hardware sequences.) For example, if work that holds a POW atomic tag critical

section cannot proceed until it enters another critical section, but the other critical section is held by work that is waiting for a tag switch to the same POW atomic lock, neither piece of work can make forward progress.

Second, when descheduling is used by any work using a particular tag value, all work using that particular tag value may need to use descheduling also. For example, a deadlock is present if a descheduled piece of work holds an atomic tag, and both cores are waiting for a pending switch to the same atomic tag, and neither of the cores eventually deschedule. The work scheduled in the cores cannot make forward progress because of the descheduled work, and the descheduled work can never be scheduled because the cores never become available. It may be advantageous for software to partition the available tag/type values into deschedulable ones and non-deschedulable ones by convention. Non-deschedulable tag switches/waits can be faster for shortly-held and/or non-conflicting tags since they need not be encumbered with any code support for descheduling. Deschedulable tag switches may require higher overhead in the non-conflicting case, but still may be faster, for long-held conflicting tags.

Third, livelock and/or infinite loops must be avoided by core software. Core software needs to eventually make forward progress on scheduled work. It is always possible for core software to create a continual stream of tag switches or deschedules so that work never completes. This may be due to either an infinite loop or livelock condition. The POW scheduler assists somewhat in this task since it never reschedules work that has a pending tag switch, and never schedules an input queue entry that has an atomic tag conflict.

Fourth, in-unit input queue entries and descheduled items must eventually schedule to core software. The POW scheduler's capability to search and find acceptable input work is limited by the number of entries in the unit, the group value, and tag conflicts. The most obvious way to find this pitfall is with groups or the no_sched bit. The POW scheduler will never schedule work that has the no_sched bit set or is to a group number that is not acceptable by cores, and work that cannot be scheduled due to this constraint will consume an entry in the POW unit. Of course, work can also schedule more freely when it has fewer tag synchronization constraints, so it is better to use more tags rather than fewer. A more subtle issue, which may be more of a performance problem than a hard forward-progress problem, can occur when input queue work requests a highly-contended atomic tag value. The POW scheduler will not schedule this input queue item whenever there is any in-flight work holding or switching to the same atomic tag. Another more subtle issue may occur when some input queues have entries in the POW unit, and remaining input queues do not have any entries in the unit because space is not available. Core software may eventually need to service the in-unit input queue entries before the POW scheduler can select the in-memory input queue entries.

Fifth, the in-flight full condition. This occurs when there are so many in-flight entries that the POW hardware cannot place any input queue entries in the unit, and furthermore may not be able to successfully allocate a POW entry for a Null Rd command. The POW hardware can, however, schedule descheduled items and service tag switches for existing POW entries when in an in-flight full condition. Core software must eventually accept and complete descheduled work to relieve an in-flight full condition. Deadlock could occur, for example, if both cores simply stall for successful Null Rd commands following a deschedule or prior to a switch to NULL. The Null Rds may always fail due to the in-flight full condition, and the in-flight full condition may never clear out since no POW entries are freed when work completes. More simply, core software must not require that POW schedule an input queue entry before it can accept descheduled work.

5.11 POW Operations

This section shows the detailed bit formats and codes for the various transactions.

5.11.1 Load Operations

NOTE:
- Unused and unaddressed fields in the address are reserved and must be set to 0s by the software.
- Unused fields in the results are actually 0s.
- The POW load result is unpredictable when anything but a 64-bit load (i.e. anything other than an LD) is used.

GET_WORK Load Operations

Physical Address for a GET_WORK Load

48	47 43	42 40	39 4	3	2 0
1	Major DID 011 00	subDID 00 0	Reserved 0000 0000 0000 0000 0000 0000 0000 0000 0000	Wait	00

- **Wait** - If set, don't return load result until work is available or time-out

NOTE: The POW hardware behavior is unpredictable when anything but a 64-bit load (i.e. anything other than an LD) is used for work request loads or NULL_RD loads.

Result for a GET_WORK Load

63	62 40	39 0
no_work	Reserved 000 0000 0000 0000 0000 0000	addr

- **no_work** - Set when no new work-queue entry was returned
- **addr** - The work-queue pointer. Must be aligned on a 64-bit boundary. Unpredictable when no_work is set.

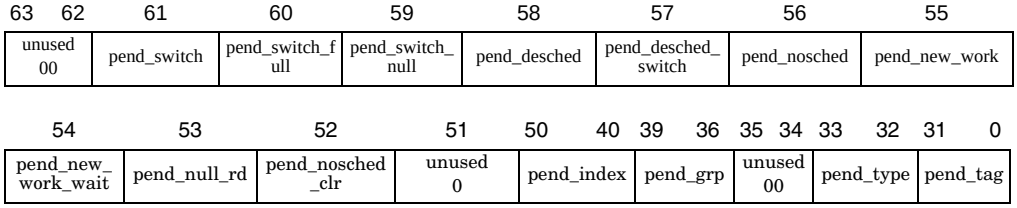
POW Status Load Operations

Physical Address for a POW Status Load

48	47 43	42 40	39 10	9 6	5	4	3	2 0
1	Major DID 0110 0	subDID 001	Reserved 0000 0000 0000 0000 0000 0000 0000 00	coreid	get_rev	get_cur	get_wqp	0

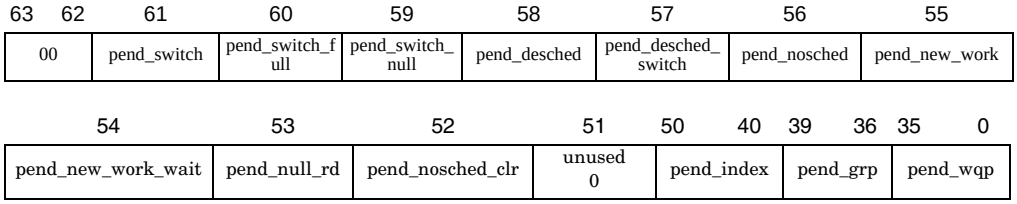
- **get_rev** - If set and **get_cur** is set, return reverse tag-list pointer rather than forward tag-list pointer
- **get_cur** - If set, return current status rather than pending status
- **get_wqp** - If set, get the work-queue pointer rather than tag/type

Result for a POW Status Load (when get_cur==0 and get_wqp==0)



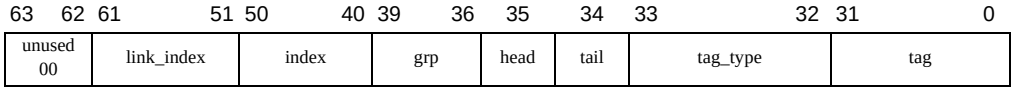
- **pend_switch** - set when there is a pending non-NULL SWTAG or SWTAG_FULL, and the POW entry has not left the list for the original tag.
- **pend_switch_full** - set when SWTAG_FULL and **pend_switch** is set.
- **pend_switch_null** - set when there is a pending NULL SWTAG, or an implicit switch to NULL.
- **pend_desched** - set when there is a pending DESCHED or SWTAG_DESCHED.
- **pend_desched_switch** - set when there is a pending SWTAG_DESCHED and **pend_desched** is set.
- **pend_nosched** - set when nosched is desired and **pend_desched** is set.
- **pend_new_work** - set when there is a pending GET_WORK.
- **pend_new_work_wait** - when **pend_new_work** is set, this bit indicates that the wait bit was set.
- **pend_null_rd** - set when there is a pending NULL_RD.
- **pend_nosched_clr** - set when there is a pending CLR_NSCHED.
- **pend_index** - this is the index when **pend_nosched_clr** is set.
- **pend_grp** - this is the **new_grp** when (**pend_desched** AND **pend_desched_switch**) is set.
- **pend_type** - this is the tag type when **pend_switch** or (**pend_desched** AND **pend_desched_switch**) are set.
- **pend_tag** - this is the tag when **pend_switch** or (**pend_desched** AND **pend_desched_switch**) are set.

Result for a POW Status Load (when get_cur==0 and get_wqp==1)



- **pend_wqp** - this is the **wqp** when **pend_nosched_clr** is set.

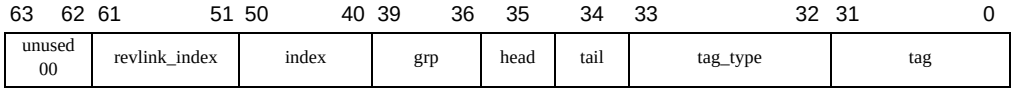
Result for a POW Status Load (when get_cur==1, get_wqp==0, and get_rev==0)



NOTE: The **link_index**, **grp**, and **tag** fields are unpredictable when the core's state is NULL or NULL_NULL.
The **index** field is unpredictable when the core's state is NULL_NULL.

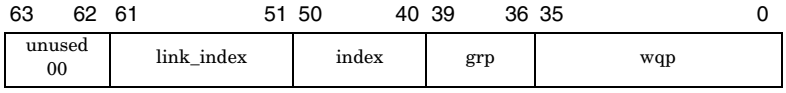
- **link_index** - points to the next POW entry in the tag list when **tail** == 0 (and **tag_type** is not NULL or NULL_NULL),.
- **index** - the POW entry attached to the core.
- **grp** - the group attached to the core (updated when new tag list entered on SWTAG_FULL).
- **head** - set when this POW entry is at the head of its tag list (also set when in the NULL or NULL_NULL state).
- **tail** - set when this POW entry is at the tail of its tag list (also set when in the NULL or NULL_NULL state).
- **tag_type** - the tag type attached to the core (updated when new tag list entered on SWTAG, SWTAG_FULL, or SWTAG_DESCHED).
- **tag** - the tag attached to the core (updated when new tag list entered on SWTAG, SWTAG_FULL, or SWTAG_DESCHED).

Result for a POW Status Load (when get_cur==1, get_wqp==0, and get_rev==1)



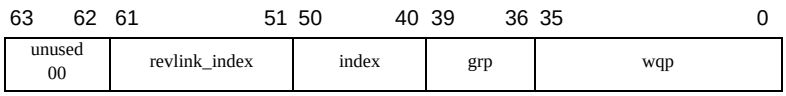
- **revlink_index** - points to the prior POW entry in the tag list when **head** == 0 (and **tag_type** is not NULL or NULL_NULL). This field is unpredictable when the core's state is NULL or NULL_NULL.

Result for a POW Status Load (when get_cur==1, get_wqp==1, and get_rev==0)



- **wqp** - the wqp attached to the core (updated when new tag list entered on SWTAG_FULL).

Result for a POW Status Load (when get_cur==1, get_wqp==1, and get_rev==1)



POW Memory Load Operations

Physical Address for a POW Memory Load

48	47 43	42 40	39 16	15 5	4	3	2 0
1	Major DID 0110 0	subDID 010	Reserved 0000 0000 0000 0000 0000 0000	index	get_des	get_wqp	0

- **get_des** - If set, return deschedule information rather than the standard response for work-queue index (invalid if the work-queue entry is not on the deschedule list).
- **get_wqp** - If set, get the work-queue pointer rather than tag/type (no effect when **get_des** set).

Result For POW MemoryLoad (get_des == 0 and get_wqp == 0)

63 51	50 40	39 36	35	34	33 32	31 0
unused 0000 0000 0000 0	next_index	grp	unused 0	tail	tag_type	tag

NOTE: All these fields are unpredictable when the POW entry is currently scheduled to a core.

The **grp, tail, tag_type**, and **tag** fields are unpredictable when the POW entry is on a free list.

The **next_index** field is unpredictable when the POW entry is on the reserved free list.

- **next_index** - the next entry in the input, free, descheduled_head list (unpredictable if entry is the tail of the list).
- **grp** - the group of the POW entry.
- **tag_type** - the tag type of the POW entry.
- **tag** - the tag of the POW entry.

Result For POW MemoryLoad (get_des == 0 and get_wqp == 1)

63 51	50 40	39 36	35 0
unused 0000 0000 0000 0	next_index	grp	wqp

NOTE: All these fields are unpredictable when the POW entry is currently scheduled to a core.

The **grp** and **wqp** fields are unpredictable when the POW entry is on a free list.

The **next_index** field is unpredictable when the POW entry is on the reserved free list.

- **wqp** - the WQP held in the POW entry.

Result For POW MemoryLoad (get_des == 1)

63 51	50 40	39 36	35	34	33 32	31 0
unused 0000 0000 0000 0	fwd_index	grp	nosched	pend_switch	pend_type	pend_tag

NOTE: These fields are unpredictable when the POW entry is not descheduled (i.e., when the POW entry is currently scheduled to a core, in an input queue, or on a free list).

- **fwd_index** - the next entry in the tag list connected to the descheduled head.
- **pend_tag** - the next tag for the new tag list when **pend_switch** is set.
- **nosched** - the nosched bit for the POW entry.
- **pend_type** - the next tag type for the new tag list when **pend_switch** is set.
- **pend_tag** - the next tag for the new tag list when **pend_switch** is set.

POW Index/Pointer Load Operations

Physical Address for a POW Index/Pointer Load

48	47 43	42 40	39 9	8 5	4	3	2 0
1	Major DID 0110 0	subDID 011	Reserved 0000 0000 0000 0000 0000 0000 0000 000	qosgrp	get_des_ get_tail	get_rmt	000

NOTE: The POW hardware behavior is unpredictable when anything but a 64-bit load (i.e. anything other than an LD) is used for work request loads or NULL_RD loads.

- **qosgrp** -
  - when {**get_rmt** ==0 AND **get_des_get_tail** == 0}, this field selects one of eight POW internal-input queues (0-7), one per QOS level; values 8-15 are illegal in this case;
  - when {**get_rmt** ==0 AND **get_des_get_tail** == 1}, this field selects one of 16 deschedule lists (per group);
  - when **get_rmt** ==1, this field selects one of 16 memory-input queue lists. The two memory-input queue lists associated with each QOS level are:

    qosgrp = 0, qosgrp = 8: QOS0
    qosgrp = 1, qosgrp = 9: QOS1
    qosgrp = 2, qosgrp = 10: QOS2
    qosgrp = 3, qosgrp = 11: QOS3
    qosgrp = 4, qosgrp = 12: QOS4
    qosgrp = 5, qosgrp = 13: QOS5
    qosgrp = 6, qosgrp = 14: QOS6
    qosgrp = 7, qosgrp = 15: QOS7

- **get_des_get_tail** - if set and **get_rmt** is clear, return deschedule list indexes rather than indexes for the specified qos level; if set and **get_rmt** is set, return the tail pointer rather than the head pointer for the specified qos level.
- **get_rmt -** if set, return remote pointers rather than the local indexes for the specified qos level.

Result For POW Index/Pointer Load (get_rmt == 0/get_des_get_tail == 0)

63 52	51	50	49 48	38	37 36	26	25	24	23 22	12	11 10	0
unused 0000 0000 0000	free_ val	free_ one	unused 0	free_ head	unused 0	free_ tail	loc_ val	loc_ one	unused 0	loc_ head	unused 0	loc_tail

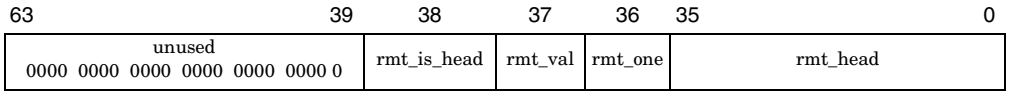
- **free_val** - set when there is one or more POW entries on the free list.
- **free_one** - set when there is exactly one POW entry on the free list.
- **free_head** - when **free_val** is set, indicates the first entry on the free list.
- **free_tail** - when **free_val** is set, indicates the last entry on the free list.
- **loc_val** - set when there is one or more POW entries on the input Q list selected by **qosgrp**.
- **loc_one** - set when there is exactly one POW entry on the input Q list selected by **qosgrp**.
- **loc_head** - when **loc_val** is set, indicates the first entry on the input Q list selected by **qosgrp**.
- **loc_tail** - when **loc_val** is set, indicates the last entry on the input Q list selected by **qosgrp**.

Result For POW Index/Pointer Load (get_rmt == 0/get_des_get_tail == 1)

63 52	51	50	49	48 38	37	36 26	25	24	23 2212	11	10	0
unused 0000 0000 0000	nosched _val	nosched _one	unused 0	nosched _head	unused 0	nosched _tail	des_ val	des_ one	unused 0	des_ head	unused 0	des_tail

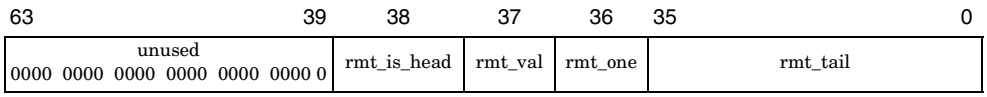
- **nosched_val** - set when there is one or more POW entries on the nosched list.
- **nosched_one** - set when there is exactly one POW entry on the nosched list.
- **nosched_head** - when **nosched_val** is set, indicates the first entry on the nosched list.
- **nosched_tail** - when **nosched_val** is set, indicates the last entry on the nosched list.
- **des_val** - set when there is one or more descheduled heads on the descheduled list selected by **qosgrp**.
- **des_one** - set when there is exactly one descheduled head on the descheduled list selected by **qosgrp**.
- **des_head** - when **des_val** is set, indicates the first descheduled head on the descheduled list selected by **qosgrp**.
- **des_tail** - when **des_val** is set, indicates the last descheduled head on the descheduled list selected by **qosgrp**.

Result For POW Index/Pointer Load (get_rmt == 1/get_des_get_tail == 0)



- **rmt_is_head** - set when this DRAM list is the current head (i.e. is the next to be reloaded when the POW hardware reloads a POW entry from DRAM). The POW hardware alternates between the two DRAM lists associated with a QOS level when it reloads work from DRAM into the POW unit.
- **rmt_val** - set when the DRAM portion of the input Q list selected by **qosgrp** contains one or more pieces of work.
- **rmt_one** - set when the DRAM portion of the input Q list selected by **qosgrp** contains exactly one piece of work.
- **rmt_head** - when **rmt_val** is set, indicates the first piece of work on the DRAM input Q list selected by **qosgrp**.

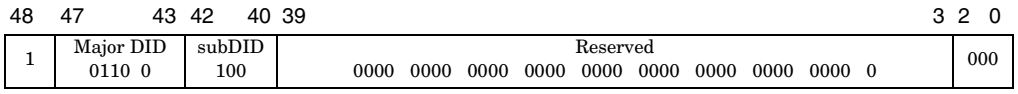
Result For POW Index/Pointer Load (get_rmt == 1/get_des_get_tail == 1)



- **rmt_is_head** - set when this DRAM list is the current head (i.e. is the next to be reloaded when the POW hardware reloads a POW entry from DRAM). The POW hardware alternates between the two DRAM lists associated with a QOS level when it reloads work from DRAM into the POW unit.
- **rmt_val** - set when the DRAM portion of the input Q list selected by **qosgrp** contains one or more pieces of work.
- **rmt_one** - set when the DRAM portion of the input Q list selected by **qosgrp** contains exactly one piece of work.
- **rmt_tail** - when **rmt_val** is set, indicates the last piece of work on the DRAM input Q list selected by **qosgrp**.

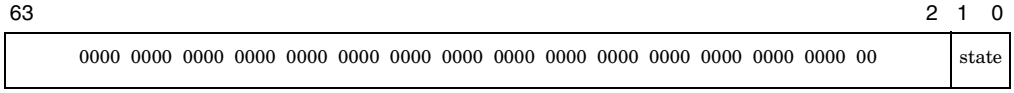
NULL_RD Load Operations

Physical Address for a NULL_RD Load



NOTE: The POW hardware behavior is unpredictable when anything but a 64-bit load (i.e. anything other than an LD) is used for work request loads or NULL_RD loads.

Result For NULL_RD Load



- **state** - NULL_NULL (3) on failure

5.11.2 IOBDMA Operations

GET_WORK IOBDMA Operations

IOBDMA GET_WORK Addressing

63 56	55 48	47 43	42 40	39 4	3	2 0
scraddr	len 0000 0001	Major DID 0110 0	subDID 000	Reserved 0000 0000 0000 0000 0000 0000 0000 0000 0000	Wait	000

- **scraddr -** Defined in "cnMIPS™ Core" on page 147.
- **Wait** - If set, don't return load Result until work is available or time-out

Result for an IOBDMA GET_WORK

63	62 40	39 0
no_work	Reserved 000 0000 0000 0000 0000 0000	addr

- **no_work** - Set when no new work-queue entry was returned
- **addr** - The work-queue pointer. Must be aligned on a 64-bit boundary. Unpredictable when no_work is set.

NULL_RD IOBDMA Operations

IOBDMA NULL_RD Addressing

63 56	55 48	47 43	42 40	39 0
scraddr	len 0000 0001	Major DID 0110 0	subDID 100	Reserved 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

- **scraddr -** Defined in "cnMIPS™ Core" on page 147.

Result For NULL_RD IOBDMA

63 2	1 0
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 00	state

- **state** - NULL_NULL (3) on failure

5.11.3 Store Operations

Physical Address to Store to POW

48	47 43	42 40	39 36	35 0
1	Major DID 0110 0	subDID	Reserved 0000	addr

- **subdid** == 0 for SWTAG_FULL, SWTAGs not switching to NULL
  subdid == 3 for DESCHED, SWTAG_DESCHED
  subdid == 1 otherwise
- **addr** - The address of the work-queue entry. Not used for SWTAG, SWTAG_DESCHED, DESCHED, NOP, addr<2:0> must be zero.

NOTE: The POW hardware behavior is unpredictable when anything but a 64-bit store (i.e. anything other than an SD) is used.

Store Data on a Store to POW

63	62	61 60	48 47	44 43	42 41	39 38	35 34	32 31	0
no_sched	Reserved 00	index	op	Rsvd 00	qos	grp	type	tag	

- **no_sched** - Don't reschedule this entry. no_sched is used for SWTAG_DESCH and DESCHED
- **index** - Contains index of entry for a CLR_NSCHED
- **op** - The operation to perform

  The following different op tag types are used:
  - SWTAG = 0
  - SWTAG_FULL = 1
  - SWTAG_DESCHED = 2
  - DESCHED = 3
  - ADDWQ = 4
  - UPD_WQP_GRP = 5
  - CLR_NSCHED = 7
  - NOP = 15
- **qos** - The QOS level for the packet. qos is only used for ADDWQ.
- **grp** - The group that the work-queue entry will be scheduled to. grp is used for ADDWQ, SWTAG_FULL, SWTAG_DESCH, and UPD_WQP_GRP
- **type** - The type of the tag. type is used for everything except DESCH, UPD_WQP_GRP, NOP, and CLR_NSCHED

  The following different tag types are used:
  - ORDERED = 0
  - ATOMIC = 1
  - NULL = 2
  - NULL_NULL = 3
- **tag** - The actual tag. tag is used for everything except DESCHED, UPD_WQP_GRP, NOP, and CLR_NSCHED

5.12 POW ECC Codes

Table 5–2 shows the POW 11-bit ECC code.

Table 5–2 POW 11-Bit ECC Code

CCCCC 4:0	10:8	7:4	3:0	
00001	100	1011	0111	0x4B7
00010	101	0101	1011	0x55B
00100	110	0110	1101	0x66D
01000	111	1000	1110	0x78E
0000	111	1111	0000	0x7F0
000000	000	0000	0000	
xxxxxx	xxx	xxxx	xxxx	
210000	111	1111	0000	← Syndromes
008421	FCA	9653	EDB7	

5.13 POW Registers

The POW registers are listed in Table 5–3.

Table 5–3 POW Registers

Register	Address	CSR Type[1]	Detailed Description
POW_PP_GRP_MSK0 POW_PP_GRP_MSK1	0x0001670000000000 0x0001670000000008	NCB	See page 241
POW_WQ_INT_THR0 ... POW_WQ_INT_THR15	0x0001670000000080 ... 0x00016700000000F8	NCB	See page 241
POW_WQ_INT_CNT0 ... POW_WQ_INT_CNT15	0x0001670000000100 ... 0x0001670000000178	NCB	See page 243
POW_QOS_THR0 ... POW_QOS_THR7	0x0001670000000180 ... 0x00016700000001B8	NCB	See page 243
POW_QOS_RND0 ... POW_QOS_RND7	0x00016700000001C0 ... 0x00016700000001F8	NCB	See page 245
POW_WQ_INT	0x0001670000000200	NCB	See page 245
POW_WQ_INT_PC	0x0001670000000208	NCB	See page 246
POW_NW_TIM	0x0001670000000210	NCB	See page 246
POW_ECC_ERR	0x0001670000000218	NCB	See page 248
POW_NOS_CNT	0x0001670000000228	NCB	See page 249
POW_PF_RST_MSK	0x0001670000000230	NCB	See page 249
POW_WS_PC0 ... POW_WS_PC15	0x0001670000000280 ... 0x00016700000002F8	NCB	See page 249
POW_WA_PC0 ... POW_WA_PC7	0x0001670000000300 ... 0x0001670000000338	NCB	See page 249
POW_IQ_CNT0 ... POW_IQ_CNT7	0x0001670000000340 ... 0x0001670000000378	NCB	See page 249
POW_WA_COM_PC	0x0001670000000380	NCB	See page 250
POW_IQ_COM_CNT	0x0001670000000388	NCB	See page 250
POW_TS_PC	0x0001670000000390	NCB	See page 250
POW_DS_PC	0x0001670000000398	NCB	See page 250
POW_BIST_STAT	0x00016700000003F8	NCB	See page 251

1. NCB-type registers are accessed directly across the I/O Bus.

POW Core Group Mask Registers
POW_PP_GRP_MSK0/1

Selects which groups a core belongs to (one per core). A 1 in any bit position sets the core's membership in the corresponding group. A value of 0x0 prevents the core from receiving new work.

NOTE: Disabled or nonexistent cores should have this field set to 0xFFFF (the reset value) to maximize POW performance.

This register also contains the QOS level priorities for each core:

- 0x0 is highest priority.
- 0x7 is the lowest priority.
- 0xF prevents that core from receiving work from that QOS level.
- 0x8 – 0xE are reserved and should not be used.

For a given core, priorities should begin at 0x0 and remain contiguous throughout the range. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved
<47:44>	QOS7_PRI	R/W	0x0	—	Core0/1 priority for QOS level 7.
<43:40>	QOS6_PRI	R/W	0x0	—	Core0/1 priority for QOS level 6.
<39:36>	QOS5_PRI	R/W	0x0	—	Core0/1 priority for QOS level 5.
<35:32>	QOS4_PRI	R/W	0x0	—	Core0/1 priority for QOS level 4.
<31:28>	QOS3_PRI	R/W	0x0	—	Core0/1 priority for QOS level 3.
<27:24>	QOS2_PRI	R/W	0x0	—	Core0/1 priority for QOS level 2.
<23:20>	QOS1_PRI	R/W	0x0	—	Core0/1 priority for QOS level 1.
<19:16>	QOS0_PRI	R/W	0x0	—	Core0/1 priority for QOS level 0.
<15:0>	GRP_MSK	R/W	0xFFFF	0xFFFF	Core0/1 group mask

POW Work-Queue Interrupt Threshold Registers
POW_WQ_INT_THR(0..15)

Contains the thresholds for enabling and setting work-queue interrupts (one per group). For more information on this register, refer to Section 5.6.

NOTE: Up to two of the POW's internal storage buffers can be allocated for hardware use and are therefore not available for incoming work-queue entries. Additionally, any core that is not in the NULL_NULL state consumes a buffer. Thus in a two-core system, it is not advisable to set either [IQ_THR] or [DS_THR] to greater than 256 – 2 – 2 = 252. A higher threshold may prevent the interrupt from ever triggering.

See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:29>	—	RAZ	—	—	Reserved
<28>	TC_EN	R/W	0x0	—	Time counter interrupt enable for group(0..15). This field must be zero when [TC_THR] is 0.
<27:24>	TC_THR	R/W	0x0	—	Time counter interrupt threshold for group(0..15). When this field is equal to 0, POW_WQ_INT_CNT(0..15)[TC_CNT] is zero.
<23:20>	—	RAZ	—	—	Reserved
<19:12>	DS_THR	R/W	0x0	—	Deschedule count threshold for group(0..15). When this field is 0, the threshold interrupt is disabled.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<11:8>	—	RAZ	—	—	Reserved
<7:0>	IQ_THR	R/W	0x0	—	Input queue count threshold for group(0..15). When this field is 0, the threshold interrupt is disabled.

POW Work-Queue Interrupt Count Registers
POW_WQ_INT_CNT(0..15)

Contains a read-only copy of the counts used to trigger work-queue interrupts (one per group). For more information on this register, refer to Section 5.6. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:28>	—	RAZ	—	—	Reserved.
<27:24>	TC_CNT	RO	0x0	—	Time counter current value for group(0..15). Hardware sets this field to the value of POW_WQ_INT_THR(0..15)[TC_THR] whenever: • Corresponding POW_WQ_INT_CNT*[IQ_CNT] is equal to 0 and corresponding POW_WQ_INT_CNT*[DS_CNT] is equal to 0. • Corresponding POW_WQ_INT[WQ_INT<*>] is written with a 1 by software. • Corresponding POW_WQ_INT[IQ_DIS<*>] is written with a 1 by software. • Corresponding POW_WQ_INT_THR(0..15) is written by software. • TC_CNT is equal to 1 and periodic counter POW_WQ_INT_PC[PC] is equal to 0. Otherwise, hardware decrements this field whenever the periodic counter POW_WQ_INT_PC[PC] is equal to 0. This field is 0 whenever POW_WQ_INT_THR(0..15)[TC_THR]is equal to 0.
<23:21>	—	RAZ	—	—	Reserved.
<20:12>	DS_CNT	RO	0x0	—	Deschedule executable count for group(0..15)
<11:9>	—	RAZ	—	—	Reserved.
<8:0>	IQ_CNT	RO	0x0	—	Input-queue executable count for group(0..15)

POW QOS Threshold Registers
POW_QOS_THR(0..7)

Contains the thresholds for allocating POW internal storage buffers (one per QOS level). If the number of remaining free buffers drops below the minimum threshold (MIN_THR) or the number of allocated buffers for this QOS level rise above the maximum threshold (MAX_THR), future incoming work-queue entries are buffered externally rather than internally.

This register also contains a read-only count of the current number of free buffers (FREE_CNT), the number of internal buffers currently allocated to this QOS level (BUF_CNT), and the total number of buffers on the deschedule list (DES_CNT) (which is not the same as the total number of descheduled buffers). See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:57>	—	RAZ	—	—	Reserved.
<56:48>	DES_CNT	RO	0x0	—	Deschedule list count. Number of buffers on the deschedule list.
<47:45>	—	RAZ	—	—	Reserved.
<44:36>	BUF_CNT	RO	0x0	—	Buffer count. Number of internal buffers allocated to QOS level (0..7)
<35:33>	—	RAZ	—	—	Reserved.
<32:24>	FREE_CNT	RO	0xF9	—	Free buffer count. Number of total free buffers
<23:20>	—	RAZ	—	—	Reserved.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<19:12>	MAX_THR	R/W	0xFF	—	Max threshold for QOS level(0..7)
<11:8>	—	RAZ	—	—	Reserved.
<7:0>	MIN_THR	R/W	0x0	—	Min threshold for QOS level (0..7)

POW QOS Issue Round Registers
POW_QOS_RND(0...7)

Contains the round definitions for issuing new work. Each round consists of eight bits, with each bit corresponding to a QOS level. There are four rounds contained in each of the eight registers for a total of 32 rounds. The issue logic traverses through the rounds sequentially (lowest round to highest round) in an attempt to find new work for each core. Within each round, the issue logic traverses through the QOS levels sequentially (highest QOS to lowest QOS) skipping over each QOS level with a clear bit in the round mask.

NOTE: Setting a QOS level to all 0s in all issue round registers prevents work from being issued from that QOS level.

See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31:24>	RND_P3	R/W	0xFF	—	Round mask for (round # × 4 + 3)
<23:16>	RND_P2	R/W	0xFF	—	Round mask for (round # × 4 + 2)
<15:8>	RND_P1	R/W	0xFF	—	Round mask for (round # × 4 + 1)
<7:0>	RND	R/W	0xFF	—	Round mask for (round # × 4)

POW Work-Queue Interrupt Register
POW_WQ_INT

Contains the bits (one per group) that set work-queue interrupts and are used to clear these interrupts. Also contains the input queue interrupt temporary disable bits (one per group). For more information on this register, refer to Section 5.6. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:16>	IQ_DIS	R/W1	0x0	—	Input queue interrupt temporary disable mask. The corresponding [WQ_INT<*>] bit cannot be set due to IQ_CNT/IQ_THR check when this bit is set. The corresponding [IQ_DIS] bit is cleared by hardware whenever: • POW_WQ_INT_CNT*[IQ_CNT] is zero, or • POW_WQ_INT_CNT*[TC_CNT] is equal to 1 when periodic counter POW_WQ_INT_PC[PC] is equal to 0.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<15:0>	WQ_INT	R/W1C	0x0	—	Work-queue interrupt bits. The corresponding WQ_INT bit is set by hardware whenever: • POW_WQ_INT_CNT(0..15)[IQ_CNT] ≥ POW_WQ_INT_THR(0..15) [IQ_THR] and the threshold interrupt is not disabled. When IQ_DIS<*> is equal to 1, the interrupt is disabled. When POW_WQ_INT_THR(0..15)[IQ_THR] is equal to 0, the interrupt is disabled. • POW_WQ_INT_CNT(0..15)[DS_CNT] ≥ POW_WQ_INT_THR(0..15)[DS_THR] and the threshold interrupt is not disabled. When POW_WQ_INT_THR(0..15)[DS_THR] is equal to 0, the interrupt is disabled. • POW_WQ_INT_CNT(0..15)[TC_CNT] is equal to 1 when periodic counter POW_WQ_INT_PC[PC] is equal to 0 and POW_WQ_INT_THR(0..15)[TC_EN] is equal to 1.

POW Work-Queue Interrupt Periodic Counter Register
POW_WQ_INT_PC

Contains the threshold value for the work-queue interrupt periodic counter and also a read-only copy of the periodic counter. For more information on this register, refer to Section 5.6. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:60>	—	RAZ	—	—	Reserved
<59:32>	PC	RO	0x0	—	Work-queue interrupt periodic counter
<31:28>	—	RAZ	—	—	Reserved
<27:8>	PC_THR	R/W	0x0	—	Work-queue interrupt periodic counter threshold
<7:0>	—	RAZ	—	—	Reserved

POW New-Work Timer Period Register
POW_NW_TIM

Sets the minimum period for a new-work-request timeout. The period is specified in n-1 notation, with the increment value of 1024 clock cycles.

NOTE: The maximum period for a new work request timeout is 2 × the minimum period.

The new-work-request timeout counter is reset when this register is written.

See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:10>	—	RAZ	—	—	Reserved

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<9:0>	NW_TIM	R/W	0x0	0x3FF	New-work-timer period. 0x0 = 1024 clock cycles 0x1 = 2048 clock cycles 0x2 = 3072 clock cycles ... etc.

POW ECC Error Register
POW_ECC_ERR

This register contains the single- and double-error bits and the corresponding interrupt-enable bits for the ECC-protected POW index memory, plus the syndrome value in the event of an ECC error.

It also contains the remote-pointer-error bit and its interrupt-enable bit. RPE is set when POW detects corruption on one or more of the input-queue lists in L2/DRAM (i.e. POW's local copy of the tail pointer for the L2/DRAM input queue did not match the last entry on the the list). This is caused by L2/DRAM corruption, and is generally a fatal error because it likely caused POW to load bad work-queue entries. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:45>	—	RAZ	—	—	Reserved.
<44:32>	IOP_IE	R/W	0x0	—	Illegal-operation interrupt enable bits for corresponding errors in IOP.
<31:29>	—	RAZ	—	—	Reserved.
<28:16>	IOP	R/W1C	0x0	0x0	Illegal-operation errors. **IOP_IE — IOP — Illegal Operation** <44> — <28> — Received a CSR load operation from the core with a CSR load operation pending. <43> — <27> — Received a DBG load operation from the core with a DBG load operation pending. <42> — <26> — Received ADD_WORK with tag specified as NULL_NULL. <41> — <25> — Received an illegal opcode. <40> — <24> — Received SWTAG/SWTAG_FULL/SWTAG_DESCH/DESCH/UPD_WQP/GET_WORK/NULL_RD from the core with CLR_NSCHED pending. <39> — <23> — Received CLR_NSCHED from the core with SWTAG_DESCH/DESCH/CLR_NSCHED pending. <38> — <22> — Received SWTAG/SWTAG_FULL/SWTAG_DESCH/DESCH/UPD_WQP/GET_WORK/NULL_RD from the core with NULL_RD pending. <37> — <21> — Received SWTAG/SWTAG_FULL/SWTAG_DESCH/DESCH/UPD_WQP/GET_WORK/NULL_RD from the core with GET_WORK pending. <36> — <20> — Received SWTAG_FULL/SWTAG_DESCH from the core with the tag specified as NULL. <35> — <19> — Received SWTAG/SWTAG_FULL/SWTAG_DESCH from the core with the tag specified as NULL_NULL. <34> — <18> — Received SWTAG/SWTAG_FULL/SWTAG_DESCH/GET_WORK from the core with a pending tag switch to ORDERED or ATOMIC. <33> — <17> — Received SWTAG/SWTAG_DESCH/DESCH/UPD_WQP from the core in the NULL state. <32> — <16> — Received SWTAG/SWTAG_FULL/SWTAG_DESCH/DESCH/UPD_WQP from the core in the NULL_NULL state.
<15:14>	—	RAZ	—	—	Reserved.
<13>	RPE_IE	R/W	0	—	Remote-pointer-error interrupt-enable bits.
<12>	RPE	R/W1C	0	0	Remote-pointer error.
<11:9>	—	RAZ	—	—	Reserved.
<8:4>	SYN	RO	—	—	Syndrome value (only valid when DBE or SBE is set)
<3>	DBE_IE	R/W	0	—	Double-bit-error interrupt-enable bit.
<2>	SBE_IE	R/W	0	—	Single-bit-error interrupt-enable bit.
<1>	DBE	R/W1C	0	0	Double-bit error.
<0>	SBE	R/W1C	0	0	Single-bit error.

POW Noschedule Count Register
POW_NOS_CNT

Contains the number of work-queue entries on the noschedule list. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:9>	—	RAZ	—	—	Reserved.
<8:0>	NOS_CNT	RO	0x0	—	Number of work-queue entries on the noschedule list.

POW Prefetch Reset Mask Register
POW_PF_RST_MSK

Resets the work prefetch engine when work is stored in an internal buffer (either when the add work arrives or when the work is reloaded from an external buffer) for an enabled QOS level (1 bit per QOS level). See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved.
<7:0>	RST_MSK	R/W	0x0	—	Prefetch engine reset mask.

POW Work-Schedule Performance Counter Registers
POW_WS_PC(0..15)

One per group; counts the number of work schedules for each group. Write to clear. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	WS_PC	R/W1C	0x0	—	Work-schedule performance counter for group number.

POW Work-Add Performance Counter Registers
POW_WA_PC(0..7)

One per QOS level; counts the number of add new work requests for each QOS level. Write to clear. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	WA_PC	R/W1C	0x0	—	Work-add performance counter for QOS level number.

POW Input-Queue Count Registers
POW_IQ_CNT(0..7)

Contains a read-only count of the number of work-queue entries for each QOS level (one register per QOS level). See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	IQ_CNT	RO	0x0	—	Input-queue count for QOS level number.

POW Work-Add Combined Performance Counter Register
POW_WA_COM_PC

Counts the number of add new work requests for all QOS levels. Write to clear. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	WA_PC	R/W1C	0x0	—	Work add combined performance counter

POW Input-Queue Combined-Count Register
POW_IQ_COM_CNT

Contains a read-only count of the total number of work-queue entries in all QOS levels. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	IQ_CNT	RO	0x0	—	Input-queue combined count

POW Tag-Switch Performance Counter Register
POW_TS_PC

Counts the number of tag switch requests. Write to clear. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	TS_PC	R/W1C	0x0	—	Tag-switch performance counter

POW Deschedule Performance Counter Register
POW_DS_PC

Counts the number of deschedule requests. Write to clear. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31:0>	DS_PC	R/W1C	0x0	—	Deschedule performance counter.

POW BIST Status Register
POW_BIST_STAT

Contains the BIST status for the POW memories: 0 = pass, 1 = fail. Also contains the BIST status for the cores. Each bit in the PP field represents the corresponding core. See Table 5–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:18>	—	RAZ	—	—	Reserved.
<17:16>	PP	RO	0x0	0x0	Physical core BIST status. Each bit is the OR of all BIST results for the corresponding core (0 = pass, 1 = fail).
<15:9>	—	RAZ	—	—	Reserved.
<8>	CAM	RO	0	0	POW CAM BIST status
<7>	NBT1	RO	0	0	I/O bus transmitter memory 1 BIST status
<6>	NBT0	RO	0	0	I/O bus transmitter memory 0 BIST status
<5>	IDX	RO	0	0	Index memory BIST status
<4>	FIDX	RO	0	0	Forward index memory BIST status
<3>	NBR1	RO	0	0	I/O bus receiver memory 1 BIST status
<2>	NBR0	RO	0	0	I/O bus receiver memory 0 BIST status
<1>	PEND	RO	0	0	Pending switch memory BIST status
<0>	ADR	RO	0	0	Address memory BIST status

Chapter 6

Free Pool Unit (FPA)

This chapter contains the following information about the Free Pool Unit (FPA).

Overview

The FPA is a CN50XX unit that maintains eight infinite-size pools of pointers to free L2/DRAM memory. Both core software and other CN50XX hardware units allocate and free pointers from/to the pools. Core software and the centralized input-packet-processing hardware units allocate memory from the pools. Core software, the centralized output packet processing, PCI, and timer (TIM) hardware units free memory to the pools.

The FPA hardware implements a data structure that approximates a logical stack/LIFO for each free pointer pool. The FPA hardware unit stores/caches the top of the stacks in the unit at any time. When a pool is too large to fit in the in-unit store, the FPA builds a tree/list data structure in L2/DRAM, using the freed memory in the pool, to store the extra pointers. Each pool's size is unlimited due to this technique.

The only constraint required by the FPA hardware is that pointers submitted to the free pools must be aligned on a 128-byte boundary and the free memory must be 128 bytes or more. The free memory size can be different in different pools and, in fact, can also be different within the same pool.

Figure 6–1 shows the data structure the FPA hardware builds in memory when the in-unit stores overflow. Thirty-one out of every 32 pieces of free memory are unmodified by the FPA hardware. Their pointers are held in other free memory. One out of every 32 pieces of memory contains 32 pointers. Thirty-one of the 32 pointers point only at available memory, and the last pointer points at both free memory and 32 more pointers.

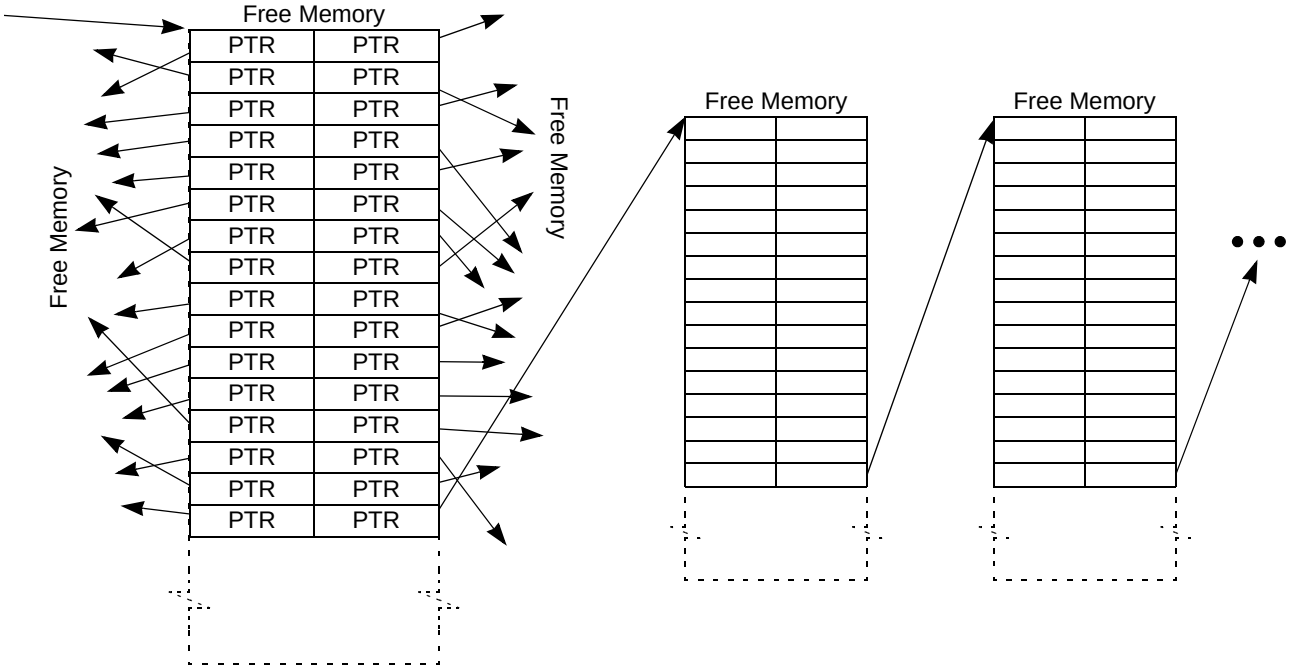


Figure 6–1 FPA Free Memory Pointers

The FPA hardware unit can internally hold up to 512 pointers, with each pool using 64 pointers for its in-unit store.

CN50XX hardware can also automatically issue “don’t-write-back” (DWB) commands on the coherent memory bus (CMB) for free memory blocks to ensure that DRAM bandwidth is not unnecessarily wasted writing cache blocks back to memory. This chapter describes operations that include fields to indicate the number of DWBs to execute, but the I/O bridge (IOB) unit implements the DWB functionality (refer to Section 3.2).

Each pool has high (= 56) and low (= 16) watermarks for its in-unit stores (see Figure 6–2.). CN50XX overflows and underflows the pointers to/from L2/DRAM based on these watermarks. When the pool has more pointers in the unit than the high watermark for the pool allows, CN50XX writes 32 pointers from the pool’s in-unit store into L2/DRAM. When the pool has fewer pointers in the unit than the low watermark for the pool allows, CN50XX reads 32 pointers from L2/DRAM into the pool’s in-unit store.

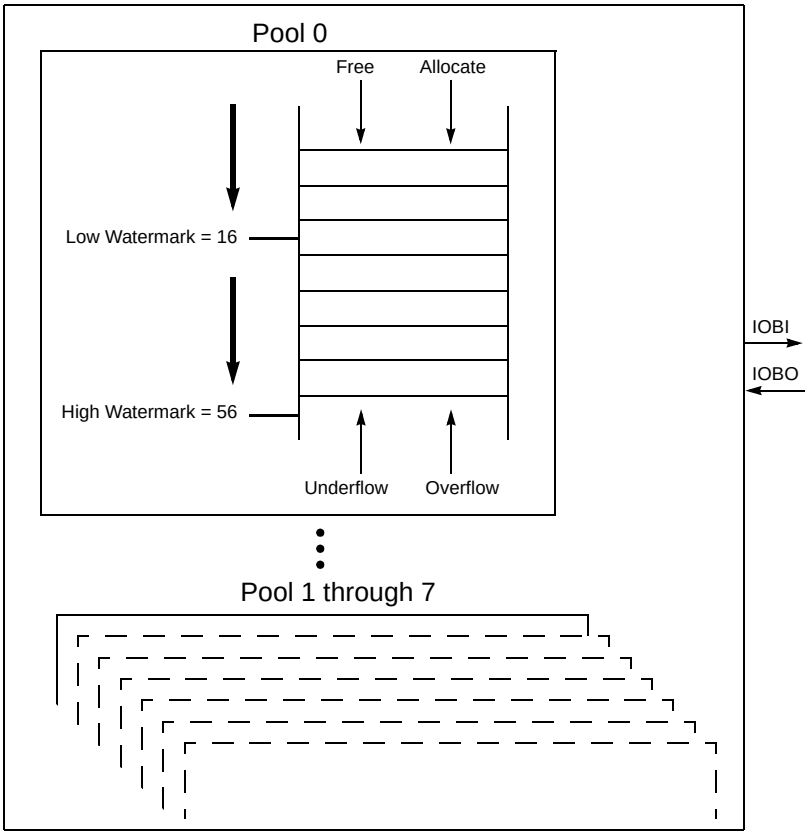


Figure 6–2 FPA Free Memory Pointers

When a core requests a pointer from a pool that does not have enough pointers to satisfy the request, the FPA hardware returns a NULL pointer (all 0s) to the core.

The eight available subDIDs select the pool for a particular operation. Pool 0 is special since the centralized input packet processing hardware uses storage allocated from it to store packet data. The centralized input packet processing hardware also allocates work queue entries from a programmable pool. Whenever pool 0 or the work queue entry pool are empty, the centralized input packet processing hardware cannot receive packets.

CN50XX's centralized input packet processing unit is the only hardware unit that can allocate storage from FPA. The packet-output (see Chapter 8), PCI DMA engine (see Chapter 9), and Timer (see Chapter 10) units free buffers to FPA.

The FPA hardware checks for corruption of the free list in DRAM. These checks can detect ordinary memory corruption, and can also detect pointer duplications in some circumstances. The FPA hardware sets one of FPA_INT_SUM[Q*n*_COFF,Q*n*_PERR,Q*n*_UND] when it detects these errors. In addition to the 32 pointers written to L2/DRAM, CN50XX writes out the page index for the pool (FPA_QUE*n*_PAGE_INDEX) and the pool number. If the values mismatch when it reads pointers back, the FPA hardware sets FPA_INT_SUM[Q*n*_PERR] and loads FPA_QUE_EXP/FPA_QUE_ACT.

Core software can read the FPA_QUE*n*_AVAILABLE[QUE_SIZ] register to find the number of buffers that are currently free in a pool.

Load and IOBDMA operations from the cores allocate memory. (The result is a byte pointer to the free memory.) Store operations free memory.

6.1 Free Pool Unit Operations

This section shows the detailed bit formats and codes for the various transactions.

6.1.1 Load Operations

Physical Address for Load

48	47 43	42 40	39 0
1	Major DID 00101	sub-DID	Reserved 0

- **subdid** - pool number

NOTE: The FPA hardware behavior is unpredictable when anything but a 64-bit load (i.e. anything other than an LD) is used.

Result for a Load

63 40	39 0
0	Ptr

- **Ptr** - Pointer to free memory. If all zeroes, the selected pool is empty.
  - **Ptr** must be aligned on a 128 byte boundary, thus Ptr <6:0> will always be zero.

6.1.2 IOBDMA Operations

IOBDMA Addressing

63 56	55 48	47 43	42 40	39 0
scraddr	len	Major DID 00101	sub-DID	Reserved 0

- **scraddr** - Defined in “cnMIPS™ Cores” on page 143.
- **len** - can legally be from 1 to 16
- **subdid** - pool number

Result for an IOBDMA Request

63 40	39 0
0	Ptr

- **Ptr** - Pointer to free memory. If all zeroes, the selected pool is empty.
  - **Ptr** must be aligned on a 128 byte boundary, thus Ptr <6:0> will always be 0x0.
  - If len is larger than the number of available pointers in the selected pool (i.e. if len > FPA_QUE*n*_AVAILABLE[QUE_SIZ]), then all (len) **Ptr**s returned for the IOBDMA operation are all 0s, indicating that the pool does not have an adequate number of pointers to satisfy the IOBDMA.

6.1.3 Store Operations

Physical Address to Store to FPA

48	47 43	42 40	39 0
1	Major DID 00101	sub-DID	addr

- **subdid** - pool number
- **addr** - pointer to available memory
  - Pointer must be aligned on 128 byte boundary (or unpredictable results)
  - Pointer must point to at least 128 bytes of free memory (or FPA hardware may over-write some useful data with pointers)

NOTE: The FPA hardware behavior is unpredictable when anything but a 64-bit store (i.e. anything other than an SD) is used.

Store Data on a Store to FPA

63 9	8 0
Reserved 0	DWB count

- The lowest N bits contain the **DWB_count**. DWB_count is the number of cache lines for which the hardware (in IOB) should try to execute “don’t-write-back” commands. The hardware starts from the beginning of the free memory (i.e. to where the address points) and marches forward linearly. As the DWB command can modify the value of memory locations, software must ensure that DWB_count × 128 does not exceed the number of available bytes. As the DWB commands consume CMB bandwidth, software should keep the DWB_count low to cover only those cache blocks that may have been modified.

6.2 FPA Registers

The FPA CSRs are listed in Table 6–1.

Table 6–1 FPA Registers

Register	Address	CSR Type[1]	Detailed Description
FPA_INT_SUM	0x0001180028000040	RSL	See page 259
FPA_INT_ENB	0x0001180028000048	RSL	See page 260
FPA_CTL_STATUS	0x0001180028000050	RSL	See page 261
FPA_QUE0_AVAILABLE ... FPA_QUE7_AVAILABLE	0x0001180028000098 ... 0x00011800280000D0	RSL	See page 261
FPA_BIST_STATUS	0x00011800280000E8	RSL	See page 262
FPA_QUE0_PAGE_INDEX ... FPA_QUE7_PAGE_INDEX	0x00011800280000F0 ... 0x0001180028000128	RSL	See page 262
FPA_QUE_EXP	0x0001180028000130	RSL	See page 262
FPA_QUE_ACT	0x0001180028000138	RSL	See page 263

1. RSL-type registers are accessed indirectly across the I/O Bus.

FPA Interrupt Summary Register
FPA_INT_SUM

Contains the different interrupt summary bits of the FPA. See Table 6–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:28>	—	RAZ	—	—	Reserved
<27>	Q7_PERR	R/W1C	0	0	Set when a queue7 pointer read from the stack in the L2C does not have the FPA ownership bit set.
<26>	Q7_COFF	R/W1C	0	0	Set when a queue7 stack end tag is present and the count available is greater than pointers present in the FPA.
<25>	Q7_UND	R/W1C	0	0	Set when a queue7 page count available goes negative.
<24>	Q6_PERR	R/W1C	0	0	Set when a queue6 pointer read from the stack in the L2C does not have the FPA ownership bit set.
<23>	Q6_COFF	R/W1C	0	0	Set when a queue6 stack end tag is present and the count available is greater than pointers present in the FPA.
<22>	Q6_UND	R/W1C	0	0	Set when a queue6 page count available goes negative.
<21>	Q5_PERR	R/W1C	0	0	Set when a queue5 pointer read from the stack in the L2C does not have the FPA ownership bit set.
<20>	Q5_COFF	R/W1C	0	0	Set when a queue5 stack end tag is present and the count available is greater than pointers present in the FPA.
<19>	Q5_UND	R/W1C	0	0	Set when a queue5 page count available goes negative.
<18>	Q4_PERR	R/W1C	0	0	Set when a queue4 pointer read from the stack in the L2C does not have the FPA ownership bit set.
<17>	Q4_COFF	R/W1C	0	0	Set when a queue4 stack end tag is present and the count available is greater than pointers present in the FPA.
<16>	Q4_UND	R/W1C	0	0	Set when a queue4 page count available goes negative.
<15>	Q3_PERR	R/W1C	0	0	Set when a queue3 pointer read from the stack in the L2C does not have the FPA ownership bit set.
<14>	Q3_COFF	R/W1C	0	0	Set when a queue3 stack end tag is present and the count available is greater than pointers present in the FPA.
<13>	Q3_UND	R/W1C	0	0	Set when a queue3 page count available goes negative.
<12>	Q2_PERR	R/W1C	0	0	Set when a queue2 pointer read from the stack in the L2C does not have the FPA ownership bit set.
<11>	Q2_COFF	R/W1C	0	0	Set when a queue2 stack end tag is present and the count available is greater than pointers present in the FPA.
<10>	Q2_UND	R/W1C	0	0	Set when a queue2 page count available goes negative.
<9>	Q1_PERR	R/W1C	0	0	Set when a queue1 pointer read from the stack in the L2C does not have the FPA ownership bit set.
<8>	Q1_COFF	R/W1C	0	0	Set when a queue1 stack end tag is present and the count available is greater than pointers present in the FPA.
<7>	Q1_UND	R/W1C	0	0	Set when a queue1 page count available goes negative.
<6>	Q0_PERR	R/W1C	0	0	Set when a queue0 pointer read from the stack in the L2C does not have the FPA ownership bit set.
<5>	Q0_COFF	R/W1C	0	0	Set when a queue0 stack end tag is present and the count available is greater than pointers present in the FPA.
<4>	Q0_UND	R/W1C	0	0	Set when a queue0 page count available goes negative.
<3>	FED1_DBE	R/W1C	0	0	Set when a single-bit error is detected in FPF1
<2>	FED1_SBE	R/W1C	0	0	Set when a double-bit error is detected in FPF1
<1>	FED0_DBE	R/W1C	0	0	Set when a single-bit error is detected in FPF0
<0>	FED0_SBE	R/W1C	0	0	Set when a double-bit error is detected in FPF0

FPA Interrupt Enable Register
FPA_INT_ENB

The FPA interrupt-enable register. See Table 6–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:28>	—	RAZ	—	—	Reserved
<27>	Q7_PERR	R/W	0	0	When set to 1 and bit[27] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<26>	Q7_COFF	R/W	0	0	When set to 1 and bit[26] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<25>	Q7_UND	R/W	0	0	When set to 1 and bit[25] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<24>	Q6_PERR	R/W	0	0	When set to 1 and bit[24] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<23>	Q6_COFF	R/W	0	0	When set to 1 and bit[23] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<22>	Q6_UND	R/W	0	0	When set to 1 and bit[22] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<21>	Q5_PERR	R/W	0	0	When set to 1 and bit[21] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<20>	Q5_COFF	R/W	0	0	When set to 1 and bit[20] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<19>	Q5_UND	R/W	0	0	When set to 1 and bit[19] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<18>	Q4_PERR	R/W	0	0	When set to 1 and bit[18] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<17>	Q4_COFF	R/W	0	0	When set to 1 and bit[17] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<16>	Q4_UND	R/W	0	0	When set to 1 and bit[16] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<15>	Q3_PERR	R/W	0	0	When set to 1 and bit[15] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<14>	Q3_COFF	R/W	0	0	When set to 1 and bit[14] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<13>	Q3_UND	R/W	0	0	When set to 1 and bit[13] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<12>	Q2_PERR	R/W	0	0	When set to 1 and bit[12] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<11>	Q2_COFF	R/W	0	0	When set to 1 and bit[11] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<10>	Q2_UND	R/W	0	0	When set to 1 and bit[10] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<9>	Q1_PERR	R/W	0	0	When set to 1 and bit[9] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<8>	Q1_COFF	R/W	0	0	When set to 1 and bit[8] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<7>	Q1_UND	R/W	0	0	When set to 1 and bit[7] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<6>	Q0_PERR	R/W	0	0	When set to 1 and bit[6] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<5>	Q0_COFF	R/W	0	0	When set to 1 and bit[5] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<4>	Q0_UND	R/W	0	0	When set to 1 and bit[4] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<3>	FED1_DBE	R/W	0	0	When set to 1 and bit[3] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<2>	FED1_SBE	R/W	0	0	When set to 1 and bit[2] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<1>	FED0_DBE	R/W	0	0	When set to 1 and bit[1] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.
<0>	FED0_SBE	R/W	0	0	When set to 1 and bit[0] of the FPA_INT_SUM register is asserted, the FPA asserts an interrupt.

FPA Control and Status Register FPA_CTL_STATUS

This register provides FPA control and status information. See Table 6–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:18>	—	RAZ	—	—	Reserved
<17>	RESET	R/W	0	0	When set causes a reset of the FPA with the exception of the RSL.
<16>	USE_LDT	R/W	0	0	When cleared to 0, the FPA uses LDT to load pointers from the L2C.
<15>	USE_STT	R/W	0	0	When cleared to 0, the FPA uses STT to store pointers to the L2C.
<14>	ENB	R/W	0	0	Enable. Must be set to 1 after writing all configuration registers and 10 cycles have passed. If any of the configuration registers are written after writing this bit, the FPA may begin to operate incorrectly.
<13:7>	MEM1_ERR	R/W	0x0	0x0	Causes a flip of the ECC bit associated [38:32] respective to bit [6:0] of this field, for FPF FIFO1.
<6:0>	MEM0_ERR	R/W	0x0	0x0	Causes a flip of the ECC bit associated [38:32] respective to bit [6:0] of this field, for FPF FIFO0.

FPA Queue 0-7 Free Page Available Registers FPA_QUE(0..7)_AVAILABLE

This register specifies the number of page pointers that are available in the FPA and local DRAM. See Table 6–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:29>	—	RAZ	—	—	Reserved
<28:0>	QUE_SIZ	RO	0x0	0x0	The number of free pages available in this queue.

BIST Status of FPA Memories Register
FPA_BIST_STATUS

This register provides the result of the BIST run on the FPA memories. See Table 6–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved
<4>	FRD	RO	0	0	FPA_FRD memory BIST status.
<3>	FPF0	RO	0	0	FPA_FPF0 memory BIST status.
<2>	FPF1	RO	0	0	FPA_FPF1 memory BIST status.
<1>	FFR	RO	0	0	FPA_FFR memory BIST status.
<0>	FDR	RO	0	0	FPA_FDR memory BIST status.

FPA Queue Page Index Registers
FPA_QUE(0..7)_PAGE_INDEX

These registers provide the present index page for each of queues 0..7. Each PG_NUM field reflects the number of pages of pointers that have been written to memory for the respective queue. See Table 6–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:25>	—	RAZ	—	—	Reserved
<24:0>	PG_NUM	RO	0x0	—	Page number.

FPA Queue Expected Value Register
FPA_QUE_EXP

When an FPA_INT_SUM[Q*n*_PERR] occurs, this register is latched with the expected value. Once this register is latched with the first error, it is not latched again until all errors have been cleared. See Table 6–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:29>	—	RO	0x0	0x7	Reserved.
<28:26>	EXP_QUE	RO	0x0	—	Expected FPA queue number read from memory.
<25:0>	EXP_INDX	RO	0x0	—	Expected page number read from memory.

FPA Queue Actual Value Register
FPA_QUE_ACT

When an FPA_INT_SUM[Q*n*_PERR] occurs, this register is latched with the actual value read from L2C. Once this register is latched with the first error, it is not latched again until all errors have been cleared. See Table 6–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:29>	—	RO	0x0	0x7	Reserved.
<28:26>	ACT_QUE	RO	0x0	—	FPA queue number read from memory.
<25:0>	ACT_INDX	RO	0x0	—	Page number read from memory.

Chapter 7

Packet Input Processing/Input Packet Data Unit (PIP/IPD)

This chapter contains the following subjects:

Overview

This chapter discusses the CN50XX centralized packet input processing and input packet data unit (PIP/IPD). It receives packet input data from any/all of the RGMII, GMII, MII or PCI interfaces. It can have a combined total of up to 5 input ports for receiving packets between all these sources. This effectively means that PIP/IPD supports a total of up to 5 simultaneous in-flight packets. The packets arriving on the different ports share the same PIP/IPD hardware resources, but logically the PIP/IPD hardware treats the different in-flight packets independently.

The PIP/IPD units allocate and write packet data into buffers in a format that is convenient to higher-layer software. The unit supports a programmable buffer size (with pads at the top and bottom for software use), and can distribute packet data across multiple buffers to support large packet input sizes.

The PIP/IPD also creates and allocates a work-queue entry for each packet. This work-queue entry contains a pointer to the buffered packet, hardware parsing results, and packet error checks. The unit performs many L2–L4 checks, including the TCP/UDP checksum check. The unit can skip over a programmable amount of user-defined input data before parsing the input packet. This is useful for user-defined headers passed with the packets. The unit also implements packet-instruction headers, which allow packet scheduling and decode information to be more closely controlled, and has highly configurable automatic tuple and/or mask packet tag generation.

7.1 Input Ports

The PIP/IPD hardware accepts packets from up to 5 input ports numbered as follows:

- PIP/IPD Ports 0–2 = packet interface ports 0–2
- PIP/IPD Ports 32–33 = PCI interface ports 0–1

The PIP/IPD hardware treats all the packet interface ports identically. The PIP/IPD hardware treats the PCI interface ports slightly differently than the packet interface ports, but in a similar manner. (There is no CRC checking on PCI interface ports, and packet-instruction headers are created differently.)

A packet interface may only use some of the three available ports allocated to it. For example, a packet interface can use at most three ports when it is in RGMII mode.

7.2 Input Packet Formats and Pre-IP Parsing

PIP/IDP supports the three different packet input parse modes.

- uninterpreted
- skip-to-L2
- skip-to-IP

Figure 7–1 depicts the packet formats supported by each parse mode.

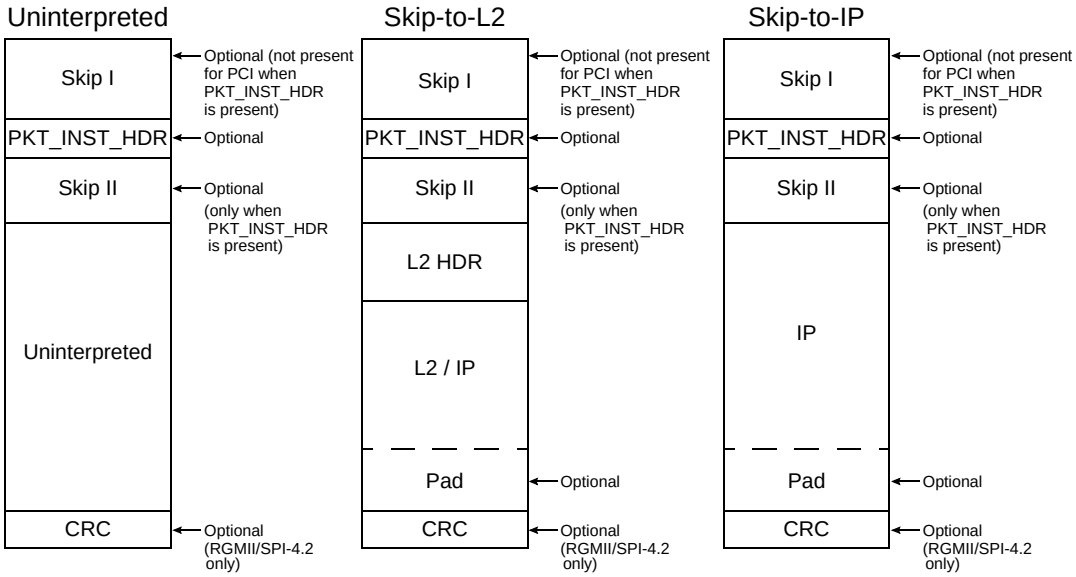


Figure 7–1 Packet Input Format Modes

When a packet does not have an instruction header, the PIP/IPD hardware creates the packet scheduling and packet-decode information based solely on the packet data and configuration. The optional packet instruction header (PKT_INST_HDR) allows an external device to more directly control the packet scheduling and decoding on a packet-by-packet basis (via RAWFULL and RAWSCH packets, defined in Section 7.2.1).

With the skip fields (and pad field), an application can attach some information in addition to the normal packet. The skip-to-L2 mode parses various ethernet-like L2 header formats and can determine whether IP is present in the packet. The skip-to-IP mode directs PIP/IPD to directly parse the contained IP packet.

Much PIP/IPD parsing is disabled for uninterpreted packets. (A raw hardware checksum is still generated, the packet length is still checked against the skip amount, and CRC may still be checked and discarded.)

The Skip I field may be present in all cases, except for PCI ports when a PKT_INST_HDR is present. Skip is not allowed prior to a PKT_INST_HDR when a packet arrives via a PCI input port. The number of Skip I bytes is configured separately for each port that a packet arrives on (PIP_PRT_CFG*n*[SKIP]). It can be any byte amount from 0 (where it is not present) to its maximum.

The Skip II field can only be present when a PKT_INST_HDR is present. The Skip II field can be any byte amount from 0 (where it is not present) up to its maximum.

The total SKIP is the total number of bytes in the Skip I, PKT_INST_HDR, and Skip II fields shown in Figure 7–1. The maximum allowed SKIP is discussed in Section 7.2.8.

The optional pad field is only relevant to the PIP/IPD hardware when the packet is IP. For example, a 40-byte IP packet coming in via the packet interface may have been padded out to the minimum defined packet size of 64 bytes. If any input packet contains a pad beyond the end of the IP packet, the PIP/IPD hardware receives the pad and buffers it like all other packet data. It does not get stripped, so the pad is eventually passed on to core software with the rest of the packet.

The optional CRC/FCS field protects the packet from transmission errors. Section 7.2.6 covers CRC in more detail.

7.2.1 Packet Instruction Header

The packet instruction header contains information to control packet scheduling and decode. Figure 7–2 shows the packet instruction header that is optionally included at the beginning of the packet. As shown, the full PKT_INST_HDR is eight bytes. All eight bytes are not required in all cases, however.

The packet-instruction-header size for a given packet is effectively selected by PKT_INST_HDR[SL], as the value in PKT_INST_HDR[SL] includes both the number of PKT_INST_HDR bytes and Skip II bytes. Refer to the PKT_INST_HDR[SL] description on page 269.

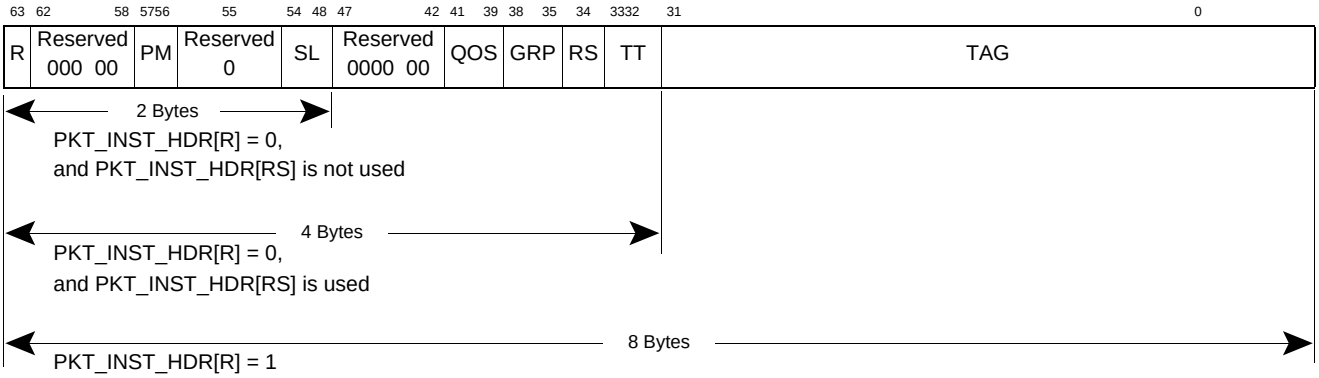


Figure 7–2 Packet Instruction Header Format (PKT_INST_HDR)

The packet instruction header format (PKT_INST_HDR) is similar to the PCI instruction header format (PCI_INST_HDR).

The packet instruction header format is also similar to the format of the packet scheduling information in WORD1 of the resultant work-queue entry. Refer to Section 7.5 for more description of the work-queue entry format created on packet arrival.

For the packet-interface ports (ports 0–2), the presence or absence of an instruction header is selected on a per-port basis (PIP_PRT_CFG(0..2)[INST_HDR]). A PKT_INST_HDR must be present for the corresponding port when PIP_PRT_CFG(0..2)[INST_HDR] is set.

All PCI instructions (refer to Section 9.3) include a PCI_INST_HDR, but the resultant packets that enter via the PCI interface ports (ports 32-33) only include a PKT_INST_HDR in the following cases:

- the PCI instruction selects raw scheduling or decoding (i.e. PCI_INST_HDR[R] is set, or the packet is RAWFULL or RAWSCH), or
- all packets arriving via the PCI port are forced to include PKT_INST_HDRs (i.e. NPI_PORT(32/33)_INSTR_HDR[USE_IHDR] is set).

When a packet has an instruction header, it may be one of two special forms:

- RAWFULL (PKT_INST_HDR[R] is set and PKT_INST_HDR[PM] is uninterpreted)

PKT_INST_HDR directly specifies the packet scheduling information, and PIP/IPD do not generate the decode information by decoding a RAWFULL packet. (The decode information comes from a configuration register (PIP_RAW_WORD) in this case.)

- RAWSCH (PKT_INST_HDR[R] is set and PKT_INST_HDR[PM] is skip-to-L2 or skip-to-IP)

  PKT_INST_HDR directly specifies the packet-scheduling information, and PIP/IPD generate the packet decode information from RAWSCH packets.

  RAWFULL and RAWSCH packets require an eight-byte PKT_INST_HDR. Other PKT_INST_HDRs may be either two or four bytes, depending on whether PKT_INST_HDR[RS] is used.

The following descriptions refer to Figure 7–2:

PKT_INST_HDR[R] If PKT_INST_HDR[R] = 1 and PKT_INST_HDR[PM] = 0 (i.e. if the packet is a RAWFULL packet), then, PIP/IPD uses PKT_INST_HDR[QOS,GRP,TT,TAG] for scheduling information (WORD 1), and PIP_RAW_WORD[WORD] for decode information (WORD 2) in the work-queue entry.

If PKT_INST_HDR[R] = 1 and PKT_INST_HDR[PM] ≠ 0 (i.e. if the packet is a RAWSCH packet), then PIP/IPD uses PKT_INST_HDR[QOS,GRP,TT,TAG] for scheduling information (WORD 1), and parses the packet for decode information (WORD 2) in the work-queue entry.

If PKT_INST_HDR[R] = 1, then PKT_INST_HDRs are eight bytes.

PKT_INST_HDR[PM] The mode used to parse the packet:

0 = uninterpreted
1 = skip-to-L2 mode
2 = skip-to-IP mode
3 = reserved

PKT_INST_HDR[SL] The number of bytes in the PKT_INST_HDR and Skip II fields (refer to Figure 7–1) in the packet.

The Skip II field could have no bytes (i.e. it may not exist), but the PKT_INST_HDR must exist and has minimum size constraints. The following bullets describe the minimum PKT_INST_HDR size and, thus, the minimum PKT_INST_HDR[SL] values for a packet:

- When PKT_INST_HDR[R] = 1, the PKT_INST_HDR is eight bytes, so PKT_INST_HDR[SL] should be eight or more.
- When PKT_INST_HDR[R] = 0, the PKT_INST_HDR may be as small as two bytes, so PKT_INST_HDR[SL] should be two or more. If PKT_INST_HDR[RS] is needed, PKT_INST_HDR[SL] should be four or more bytes rather than two. (The PKT_INST_HDR[RS] description below explains when PKT_INST_HDR[SL] might be used.)

The total SKIP length (sum of Skip I, PKT_INST_HDR, and Skip II) is discussed in Section 7.2.8.

PKT_INST_HDR[QOS, GRP,TT,TAG] When PKT_INST_HDR[R] = 1, PKT_INST_HDR[QOS,GRP,TT,TAG] are the selected scheduling parameters that become WORD1[QOS,GRP,TT,TAG] of the work-queue entry. Refer to Section 7.5.

PKT_INST_HDR[RS] When PKT_INST_HDR[RS] = 1, it enables the packet to be buffered solely in the work-queue entry, and not in L2/DRAM. Note the following points about PKT_INST_HDR[RS]:

- PKT_INST_HDR[RS] is not used if any of the following are true:
  - PIP_PRT_CFG*n*[DYN_RS] = 1 for the given port, or
  - PIP_GBL_CFG[IGNRS] = 1 and the port is not a PCI port, or
  - the packet is not dynamic short. (i.e. It doesn't fit entirely in the work-queue entry. Refer to Section 7.5 for more details regarding dynamic-short packets.)
- If PIP_GBL_CFG[IGNRS] = 1 and the port is not a PCI port, PKT_INST_HDR[RS] is effectively forced to 0.
- If PIP_PRT_CFG*n*[DYN_RS] = 1, the packet is always enabled to be buffered solely in the work-queue entry, regardless of the PKT_INST_HDR[RS] value.

7.2.2 PCI Instruction-to-Packet Conversion

Figure 7–3 shows the hardware translation from a PCI instruction into a packet, including the creation of a PKT_INST_HDR from a PCI_INST_HDR. Whenever a PKT_INST_HDR is prepended to a PCI input packet, the packet size and buffering are correspondingly increased to include the PKT_INST_HDR. A packet instruction header is present on a PCI packet when PCI_INST_HDR[R] is set, or when NPI_PORT(32/33)_INSTR_HDR[USE_IHDR] is set for the port the packet arrived on.

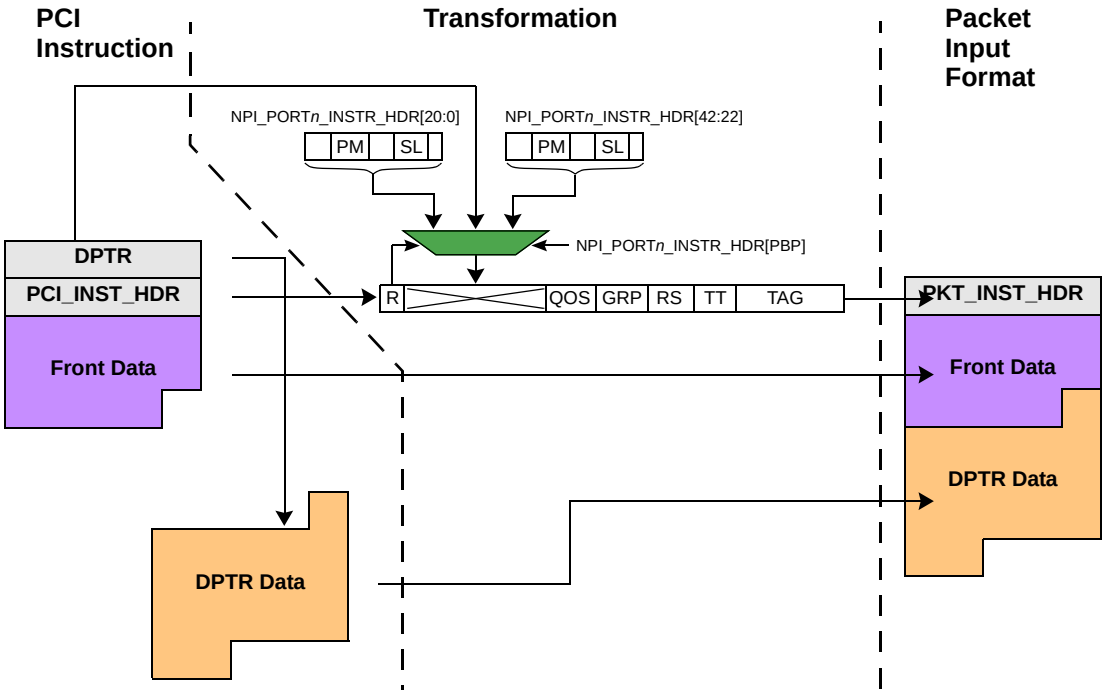


Figure 7–3 PCI Instruction to Packet Transformation

The PKT_INST_HDR comes directly from the PCI_INST_HDR in the PCI instruction, with the exception of bits <62:42>. PKT_INST_HDR[PM,SL] are the only bits within <62:42> that are not reserved.

When NPI_PORT(32/33)_INSTR_HDR[PBP] is set:

- NPI_PORT(32/33)_INSTR_HDR[USE_IHDR] must be set so that CN50XX always creates a PKT_INST_HDR.
- PKT_INST_HDR[PM,SL] come from the individual PCI instruction.

When NPI_PORT(32/33)_INSTR_HDR[PBP] is clear:

- When PCI_INST_HDR[R] is set, PKT_INST_HDR[PM,SL] come from NPI_PORT(32/33)_INSTR_HDR[RPARMODE,RSKIP_LEN].
- When PCI_INST_HDR[R] is clear and NPI_PORT(32/33)_INSTR_HDR[USE_IHDR] is set for the port, PKT_INST_HDR[PM,SL] come from NPI_PORT(32/33)_INSTR_HDR[PAR_MODE,SKIP_LEN].

Refer to Section 9.3 for more detailed description of a PCI instruction.

7.2.3 Parse Mode and Skip Length Selection

The parse mode and skip lengths are selected by a combination of per-port configuration and packet-by-packet information included in the optional PKT_INST_HDR (see Figure 7–1).

For parse mode:

- When a PKT_INST_HDR is not included in a packet, per-port configuration (i.e. PIP_PRT_CFG(0..2,32,33)[MODE]) selects the parse mode used for the packet.
- When a PKT_INST_HDR is included in a packet, PKT_INST_HDR[R,PM] select the parse mode used for the packet, and determine whether the packet is RAWFULL or RAWSCHED.

For skip lengths:

- Per-port configuration (i.e. PIP_PRT_CFG*n*[SKIP]) selects the number of bytes in the Skip I field, except when a PKT_INST_HDR is present in a packet arriving via a PCI port, in which case the Skip I field is not present.
- Whenever a PKT_INST_HDR is included with a packet, PKT_INST_HDR[SL] indicates the combined number of bytes in both the PKT_INST_HDR and Skip II fields.

Note that, as described in Section 7.2.2, PKT_INST_HDR[PM,SL] (when present) are derived from per-port configuration rather than packet-by-packet information for the PCI ports (ports 32–33) if NPI_PORT(32/33)_INSTR_HDR[PBP] is clear.

7.2.4 PIP/IPD L2 Parsing and Is_IP Determination

In the Skip-to-L2 mode, the PIP/IPD hardware parses the L2 HDR field to determine its length and whether an IP packet follows. Figure 7–4 depicts the supported L2 HDR types, which are Ethernet II and IEEE 802.3 with zero, one, or two additional VLAN fields. Regardless of the classification, the packet is considered to be IP only when the last two bytes of a valid L2 HDR field (i.e. the TYPE field) equals 0x0800 (IPv4) or 0x86DD (IPv6).

In the skip-to-IP case, there is no L2 HDR field, but the PIP/IPD hardware acts much the same as the case when the hardware considers the packet to be IP in the Skip-to-L2 mode. Note that the skip field could be anything, including an L2 header field that does not need to be interpreted by the PIP/IPD hardware.

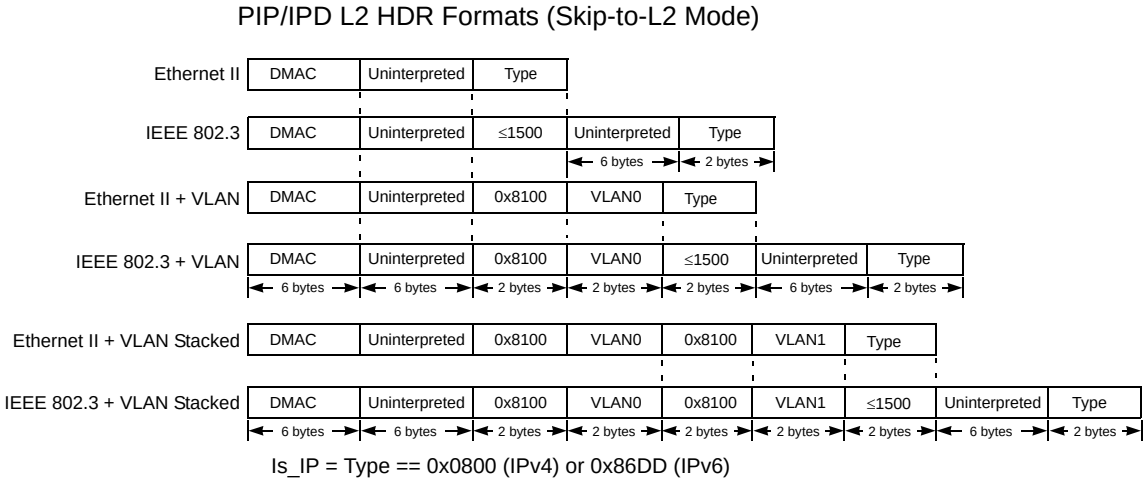


Figure 7–4 Supported L2 HDR Types in Skip-to-L2 Mode

7.2.5 Pre-IP Parsing Summary

The following pseudo-code describes the Pre-IP parsing in detail:

```
ethertype_val = 0;
LAST_SKIP = PIP_PRT_CFG[port][SKIP];
// port is the input port the packet arrived on (0..2)
if(port >= 32) { // if a PCI port
  if(PCI_INST_HDR[R] or NPI_PORT(32..33)_INSTR_HDR[USE_IHDR]) {
    PKT_INST_HDR = PCI_INST_HDR;
    pkt_inst_hdr_pres = true;
    SKIP = 0;
    if(PCI_INST_HDR[R])
      PKT_INST_HDR<62:42> = NPI_PORT[port]_INSTR_HDR<42:22>; // RPARMODE,RSKIP_LEN
    else
      PKT_INST_HDR<62:42> = NPI_PORT[port]_INSTR_HDR<20:0>;  // PAR_MODE, SKIP_LEN
    if(NPI_PORT[port]_INSTR_HDR[PBP])
      PKT_INST_HDR[PM,SL] = from individual PCI instruction;
    // CN50XX PCI logic includes PKT_INST_HDR as first 8 bytes
  }
  else {
    skip SKIP bytes;
    pkt_inst_hdr_pres = false;
  }
}
else { // if a RGMII, GMII, MII port
  skip SKIP bytes;
  if(PIP_PRT_CFG(0..2)[INST_HDR]) {
    pkt_inst_hdr_pres = true;
    PKT_INST_HDR = next 8 bytes; // 2, 4, or 8 bytes may be used
  }
```

```
  else
    pkt_inst_hdr_pres = false;
}
if(pkt_inst_hdr_pres) {
  MODE = PKT_INST_HDR[PM];
  SKIPSL = PKT_INST_HDR[SL]; // note that PKT_INST_HDR[SL] should account for
                             // PKT_INST_HDR size (minimum two to eight bytes)
  ISRAWFULL = PKT_INST_HDR[R] && (MODE is uninterpreted);
  ISRAWSCH = PKT_INST_HDR[R] && (MODE is skip-to-L2 or skip-to-IP);
  skip SKIPSL bytes;
  SKIP = SKIP + SKIPSL;
}
else {
  MODE = PIP_PRT_CFG[port][MODE];
  ISRAWFULL = false;
  ISRAWSCH = false;
}
L2_Size = 0;
VV = VS = false;
VLAN0 = VLAN1 = 0;
if(MODE is skip-to-L2) {
  skip 14 bytes;
  L2_Size = L2_Size + 14;
  type = prior two bytes;
  if(type == 0x8100) { // VLAN
    VV = true;
    VLAN0 = next two bytes;
    skip 4 bytes;
    L2_Size = L2_Size + 4;
    ype = prior two bytes;
    if(type == 0x8100) { // STACKED VLAN
      VLAN1 = next two bytes;
      VS = true;
      skip 4 bytes;
      L2_Size = L2_Size + 4;
      type = prior two bytes;
    }
  }
  limit = PIP_GBL_CFG[MAX_L2] ? 1535 : 1500;
  if(type <= limit) {
    skip 8 bytes;
    L2_Size = L2_Size + 8;
    type = prior two bytes;
  }
  Is_IP = (type == 0x0800) || (type == 0x86DD);
}
else if(MODE == skip-to-IP)
  Is_IP = true;
else
  Is_IP = false;
```

7.2.6 Packet Input CRC

PIP/IPD can remove CRC from a packet, but never checks it. The packet interfaces check CRC of inbound packets.

Note that the PIP/IPD hardware can never remove CRC for a packet that arrives via the PCI interface (ports 32–33).

For packets arriving by the packet interfaces (ports 0-2), PIP/IPD can optionally either remove or retain the CRC field for later processing. When IPD_SUB_PORT_FCS[PORT_BIT<port>] is set for the port a packet arrives on, PIP/IPD removes the CRC before buffering the packet.
IPD_SUB_PORT_FCS[PORT_BIT<port>] should only be set when both CRC is present and should be removed.

7.2.7 Packet Length Checks

PIP/IDP is capable of two classes of length checks. The first class of checks ensures that the packet is within a legal packet size. The second class is able to validate the L2 length field when an L2 header is present.

These exceptions can be enabled by setting the appropriate bit in PIP_PRT_CFGX for each port in which the checker is desired. If the bit is set, the exception is logged in PIP_INT_REG. In addition, packets sent into CN50XX arrive with a receive error (WQE WORD2[RE] = 1) and the opcode (WORD2[Opcode]) is set to the exception in the work queue entry.

Table 7–1 lists the length exceptions, their causes, and how they are handled.

Table 7–1 Receive Errors/Exceptions

Exception	Cause	Notification
MAXERR	A packet was received with length > PIP_FRM_CHK0/1[MAXLEN] bytes. For tagged frames, MAXLEN increases by four bytes for each VLAN found up to a maximum of two VLANs, or MAX + 8 bytes.	PIP_PRT_CFG*n*[MAXERR_EN] enables the check for port *n*. If enabled, PIP_INT_REG[MAXERR] is set to 1 and WQE WORD2[OPCODE] is set to 0x3 (if packet has bad FCS) or 0x4 (if packet has good FCS).
MINERR	A packet was received with length < PIP_FRM_CHK0/1[MINLEN] bytes.	PIP_PRT_CFG*n*[MINERR_EN] enables the check for port *n*. If enabled, PIP_INT_REG[MINERR] is set to 1 and WQE WORD2[OPCODE] is set to 0x6 (if packet has bad FCS) or 0x8 (if packet has good FCS).
LENERR	A packet's received length does not match the length field extracted from the L2 header within the packet. The check is only valid for packets PARSE_MODE = skip-to-L2 with a valid length field (64–1500 bytes). Refer to Figure 7–5 for the length-check algorithm.	PIP_PRT_CFG*n*[LENERR_EN] enables the check for port *n*. When PIP_PRT_CFG*n*[VLAN_LEN] = 1, the check is surpressed for VLAN packets. When PIP_PRT_CFG*n*[PAD_LEN] = 1, the check is disabled for packets with padding in the data. If enabled, PIP_INT_REG[LENERR] is set to 1 and WQE WORD2[OPCODE] is set to 0xA.

```
#define DMAC   6
#define SMAC   6
#define LENGTH 2
#define FCS    4
#define ETHERNET_HEADER_SIZE (DMAC+SMAC+LENGTH+FCS)
// parameters
//   l2_length   16 bit field as extracted from the ethernet packet
//   pkt_size    total size for the received packet including any FCS bytes
//   SKIP        from Pre-IP Parsing (the sim of Skip I + PKT_INST_HDR + Skip II)
//   vv          VLAN Valid
//   vs          VLAN Stack
// return value
//   true        L2 Length field is ok and HW will not compute a LENERR
//   false       HW will compuete a LENERR
// notes
//   algorithm assumes that FCS is in the packet.  if this is not the case,
//   the check should be disabled in PIP_PRT_CFG[LENERR_EN].
bool l2_length_ok(uint16 l2_length, uint16 pkt_size, uint16 SKIP, bool vv, bool vs) {
  if (PKT_PARSE_MODE != skip-to-L2)
    return true;
  if (l2_length > PIP_GBL_CFG[MAX_L2] ? 1535 : 1500;
    return true;
  uint16 apply_pad_then_vlan;
  {
    apply_pad_then_vlan  = (l2_length < 46) ? 46 : l2_length;
    apply_pad_then_vlan += 4*vv + 4*vs;
    apply_pad_then_vlan += ETHERNET_HEADER_SIZE;
    apply_pad_then_vlan += SKIP;
  }
  uint16 apply_vlan_then_pad;
  {
    apply_vlan_then_pad  = l2_length + 4*vv + 4*vs;
    apply_vlan_then_pad  = (apply_vlan_then_pad < 46) ? 46 : apply_vlan_then_pad;
    apply_vlan_then_pad += ETHERNET_HEADER_SIZE;
    apply_vlan_then_pad += SKIP;
  }
  bool result = false;
  if ((pkt_size == apply_pad_then_vlan) || (pkt_size == apply_vlan_then_pad))
    result = true;
  if (VLAN_LEN && vv)
    result = true;
  if (PAD_LEN && ((l2_length+ETHERNET_HEADER_SIZE+SKIP < pkt_size) || (l2_length < 46 && SKIP != 0))
    result = true;
  return (result);
}
```

Figure 7–5 Length-Check Algorithm

7.2.8 Legal SKIP Values

The total SKIP (total number of bytes in the Skip I, PKT_INST_HDR, and Skip II fields shown in Figure 7–1) must not be too large.

SKIP should always be less than the number of bytes in the packet. Otherwise WORD2[RE] will be set in the work-queue entry and PIP_INT_REG[SKPRUNT] will be set. WORD2[Opcode] will also be 12 or 17 if the packet was not already tagged with a receive error that has a smaller WORD2[Opcode] code. (The packet may commonly already be tagged with a WORD2[Opcode] = 6 or 8 (i.e. MINERR) receive error, so WORD2[Opcode] = 12 or 17 may not be often seen.) The number of bytes in the packet is before the optional CRC strip.

When the maximums specified in Table 7–2 are adhered to, PIP/IPD hardware is fully functional, with no caveats.

Table 7–2 SKIP Maximum Values (No Caveats Needed)

Packet Type	Maximum SKIP (No Caveats Needed)
Uninterpreted, RAWFULL, or RAWSCH	127
Skip-to-L2, Is_IP=0	130 – L2_Size
Skip-to-L2, Is_IP=1, IPv4	104 – L2_Size
Skip-to-L2, Is_IP=1, IPv6	84 – L2_Size
Skip-to-IP, IPv4	104
Skip-to-IP, IPv6	84

PIP/IPD hardware can also function with some caveats at larger SKIP values.

- For example, PIP/IPD hardware supports packets and SKIP values in the ranges specified in Table 7–3 with modest caveats.

Table 7–3 SKIP Value Ranges (QOS L4 Watcher Caveat)

Packet Type	SKIP Range (QOS L4 Watcher Caveat)
Skip-to-L2, Is_IP=1, IPv4	(105 – L2_Size) to (116 – L2_Size)
Skip-to-L2, Is_IP=1, IPv6	(85 – L2_Size) to (98 – L2_Size)
Skip-to-IP, IPv4	105 to 116
Skip-to-IP, IPv6	85 to 98

CN50XX is fully functional for these IPv4/IPV6 packets in these SKIP ranges when L4 QOS watchers are not enabled.

When L4 QOS watchers are enabled for these packets with a SKIP in this range, the hardware may disable the L4 watchers when it generates the QOS value used for admission control. This means that the QOS that the hardware uses for admission control may differ from the QOS used for the work-queue entry. (Sections 7.4.2 and 7.5 define the QOS value used for the work-queue entry, and Section 7.7 defines the QOS generated for admission control.)

L4 QOS watchers are enabled for a packet when there exists an **i** such that PIP_PRT_CFG*n*[QOS_WAT<i>] (for the port the packet arrives on) is set, while PIP_QOS_WATCH[i][TYPE] is also TCP or UDP.

- For example, PIP/IPD hardware supports packets and SKIP values in the ranges shown in Table 7–4 with larger, but still modest caveats.

Table 7–4 SKIP Value Ranges (QOS DiffServ/Watcher Caveat)

Packet Type	SKIP Range (QOS DiffServ/Watcher Caveat)
Skip-to-L2, Is_IP=1, IPv6	(99 – L2_Size) to (112 – L2_Size)
Skip-to-IP, IPv6	99 to 112

CN50XX is fully functional for these IPv6 packets in these SKIP ranges when neither QOS watchers nor QOS DiffServ are enabled.

When either is enabled for these IPv6 packets in these SKIP ranges, the hardware may disable QOS watchers and/or DiffServ when it generates the QOS value used for admission control, so the QOS used for admission control may differ from the QOS used for the work-queue entry.

QOS watchers are enabled for a packet when PIP_PRT_CFG*n*[QOS_WAT] (for the port the packet arrives on) is not 0, and QOS DiffServ is enabled for a packet when PIP_PRT_CFG*n*[QOS_DIFF] (for the port the packet arrives on) is set.

- Generally speaking, the PIP/IPD hardware does not support SKIP values larger than the values specified in Table 7–5.

Table 7–5 SKIP Maximum Values (All Caveats)

Packet Type	Maximum SKIP (All Caveats)
Uninterpreted, RAWFULL, or RAWSCH	127
Skip-to-L2, Is_IP=0	130 – L2_Size
Skip-to-L2, Is_IP=1, IPv4	116 – L2_Size
Skip-to-L2, Is_IP=1, IPv6	112 – L2_Size
Skip-to-IP, IPv4	116
Skip-to-IP, IPv6	112

7.3 Packet Buffering

PIP/IPD normally writes a copy of the entire packet into L2/DRAM buffers, as well as a portion or all the packet (depending on the packet size and configuration) into WORD4–WORD15 of the work-queue entry.

There are several reasons why a packet may not be buffered in ordinary L2/DRAM buffers in CN50XX:

- All bytes of some short packets can completely fit in WORD4–WORD15 of the work-queue entry for some configurations, so the ordinary L2/DRAM packet buffering is redundant and avoided. Refer to Section 7.5 for more details on this dynamic short case and WORD4–WORD15 the work-queue entry.
- Packets may selectively be dropped by PIP/IPD. Refer to Section 7.7. In this case, the packet is buffered neither in L2/DRAM, nor in the work-queue entry.
- Packets may be dropped due to buffer overflow outside of PIP/IPD. In this case also, the packet is buffered neither in L2/DRAM nor in the work-queue entry. Note that these external-buffer overflows can also result in partial packets (i.e. WORD2[RE] is set and WORD2[OPCODE] is set to 1 (partial error) in this case. See Section 7.5.), which are buffered in L2/DRAM.

The remainder of this section assumes that the packet is buffered in L2/DRAM.

The PIP/IPD hardware both allocates buffers for a packet and writes packet data into the buffers allocated for a packet. PIP/IPD gets the packet buffers it needs from free pool 0, which is a hardware free pool managed by the FPA hardware unit (see Chapter 6.) PIP/IPD prequeues a number of pointers to packet buffers that it receives from FPA. The number of packet-buffer pointers currently held by IPD is

IPD_PTR_COUNT[PKT_PCNT] + IPD_PTR_COUNT[PFIF_CNT] +
IPD_PTR_COUNT[PKTV_CNT] + IPD_PRC_PORT_PTR_FIFO_CTL[MAX_PKT]

IPD holds no pointers if it has not yet been enabled.

Figure 7–6 depicts the way that the PIP/IPD hardware writes a large packet into L2/DRAM. This example is a case where the packet is too large to fit in a single buffer (in fact, it is too large to fit into three buffers), so the PIP/IPD hardware must allocate multiple buffers, write the packet data into them, and link them together with pointers. Ultimately, the PIP/IPD hardware places a pointer to the first buffer holding the packet data in the work-queue entry that it creates to signal the arrival of a packet. This work-queue entry also indicates the number of buffers that were used (WORD2[Bufs]) and the total packet length in bytes (Len). Refer to Section 7.5 for more description of the work-queue entry and WORD2.

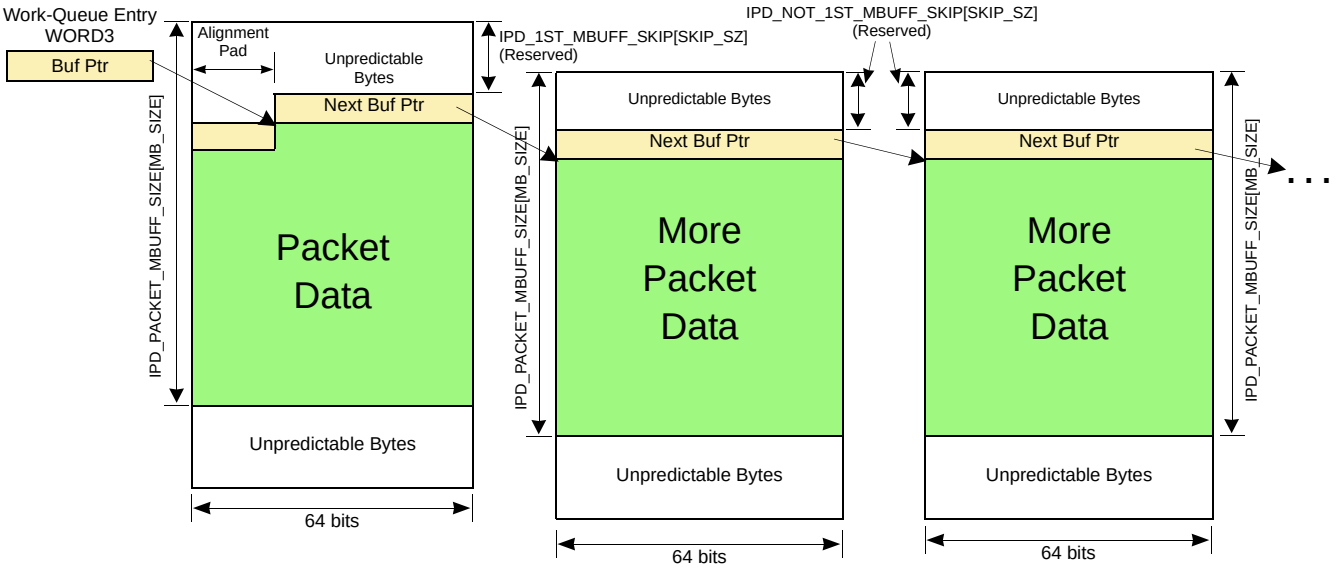


Figure 7–6 PIP/IPD Hardware Allocating Multiple Buffers

The PIP/IPD stores packets into buffers contiguously, but leaves space at the top and bottom of a buffer if desired. Software can configure the number of 64-bit words at the from the front of a buffer that are guaranteed not to contain packet data. There are two configuration values for this reserved distance at the top of a buffer:

- one for the first buffer (IPD_1ST_MBUFF_SKIP[SKIP_SZ]) used by the packet.
- the other for all subsequent buffers (IPD_NOT_1ST_MBUFF_SKIP[SKIP_SZ]) used by the packet.

These reserved field sizes can be as small as zero and as large as 32 64-bit words. Software can also configure the buffer size (IPD_PACKET_MBUFF_SIZE[MB_SIZE]), again in 64-bit words, defined as the distance from the start of the buffer to the first word that the PIP/IPD hardware cannot write. IPD_PACKET_MBUFF_SIZE[MB_SIZE] must always be at least 18 64-bit words larger than IPD_1ST_MBUFF_SKIP[SKIP_SZ], and at least 16 64-bit words larger than IPD_NOT_1ST_MBUFF_SKIP[SKIP_SZ]. Provided there is enough packet data, the PIP/IPD hardware always fills the buffer with the maximum

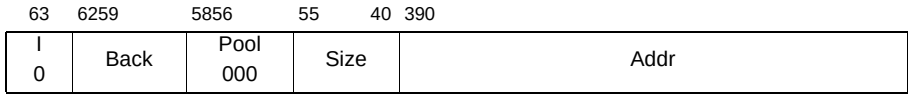
amount of packet data allowed by these configuration values (and the alignment pad in the case of the first buffer). The bytes in the buffer that do not contain packet data are unpredictable.

The PIP/IPD always writes packet data in complete (128 byte) cache blocks, including when it writes the first and last data. This means that if a buffer ends within the same cache block as the last buffer write allowed by IPD_PACKET_MBUFF_SIZE[MB_SIZE], but not at the end of a cache block, the memory immediately following the end of the buffer will be corrupted.

PIP/IPD has several modes to write packet data to L2/DRAM, selected by IPD_CTL_STATUS[OPC_MODE]:

- Write all packet data into the L2 cache.
- Write all packet data, but don't allocate space for it in the L2 cache.
- Write the first cache block of packet data into the L2 cache, and write any remaining packet data without allocating space for it in the L2 cache.
- Write the first two cache blocks of packet data into the L2 cache.
- Write any remaining packet data without allocating space for it in the L2 cache.

The PIP/IPD hardware links the packet segments in Figure 7–6 together with buffer pointers. It writes the first buffer pointer in the work-queue entry created for the packet, and writes all subsequent buffer pointers exactly eight bytes before the first byte of packet data. Figure 7–7 shows the format of each 64-bit buffer pointer. Note that the buffer pointer field in the last buffer contains unpredictable data that should never be used by core software. All other buffer pointers are valid ones. The PIP/IPD hardware allocates space in every buffer for a buffer pointer prior to the packet data in the buffer, though the buffer pointer field in the last buffer is invalid.



Addr	Byte address of start-of-packet in the buffer.
Size	Size of the buffer in bytes.
Pool	Always 0.
Back	Distance from Addr to beginning of buffer (loaded from the configuration register).
I	Always 0.

Figure 7–7 Format Of 64-bit Buffer Pointer

The **Addr** field in a valid-buffer pointer is always the physical L2/DRAM address of the first byte of packet data in the buffer. The Addr field in the buffer pointer in the work-queue entry may not be aligned on a 64-bit boundary, exactly like the start of the packet data in the first buffer, but the buffer pointers in all subsequent buffers are naturally aligned on a 64-bit boundary.

Except for the last valid-buffer pointer, the **Size** field in Figure 7–7 contains exactly the number of bytes of packet data held in the buffer. The Size field in the last valid-buffer pointer indicates the distance from Addr to the end of the buffer, just like the Size field in all other valid-buffer pointers. However, the Size field in the last valid-buffer pointer is larger than the number of bytes of packet data in the last buffer unless the packet ends exactly on the buffer boundary. When there is only one buffer, the last valid-buffer pointer is the one in the work-queue entry. Otherwise, the last valid-buffer pointer is the one in the second-to-last buffer.

CN38XX/CN36XX pass-2 versions have an erratum in which the Size field is too large by 8. The current CN50XX version is compatible with this specification (in which the Size field is correct) only when IPD_CTL_STATUS[LEN_M8] = 1. When IPD_CTL_STATUS[LEN_M8] = 0 (default value), the Size field is too large by 8, as it is in pass-2 CN38XX/CN36XX parts.

The value in the **Back** field in Figure 7–7 can indicate the distance from Addr to the beginning of the buffer (in cache blocks). PIP/IPD hardware creates it from one of two configuration variables:

- IPD_1st_NEXT_PTR_BACK[BACK]
- IPD_2nd_NEXT_PTR_BACK[BACK]):

one value for the first buffer pointer (i.e. the buffer pointer in the work-queue entry), and another for all other buffer pointers (i.e. the buffer pointers in the buffers). Software can precalculate these Back field values to match the reserved field sizes in the two cases. Sections 8.4, 8.7, 9.5.2, and 9.5.6 describe usages of the Back field, which generally specifies the number of 128-byte cache blocks.

The alignment-pad field in the first buffer in Figure 7–6 can be between 0 and 7 bytes. It places the packet data at a convenient alignment for core software processing. For example, Figure 7–8 shows the alignment for IPv4 and IPv6 packets. (The PIP hardware aligns a packet that it considers to be IP, but that has neither 4 nor 6 in the IP version field, as if it were an IPv4 packet.) In the non-IP cases, configuration variables (PIP_GBL_CFG[NIP_SHF] and PIP_GBL_CFG[RAW_SHF]) select the alignment pad value.

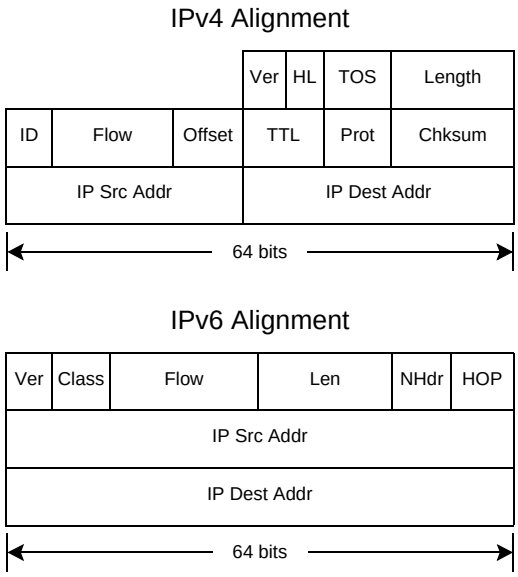


Figure 7–8 Packet Alignment for IPv4 and IPv6 Packets

The following pseudo code describes the number of alignment pad bytes added by the PIP/IPD hardware:

```
if (packet had an L1/L2 receive error) {
  // WORD2[RE] is set in this case
  if (pkt_len <= 128)
    alignment pad = PIP_GBL_CFG[NIP_SHF];
  else
    alignment pad = unpredictable; // but in the range 0..7
}
else if (ISRAWFULL)
  alignment pad = PIP_GBL_CFG[RAW_SHF];
else if (Is_IP) {
  // WORD2[NI] is clear in this case
  if (IP.ver == 6)
     // WORD2[V6] is set
    alignment pad = (8 - (SKIP + L2_size)) & 0x7;
  else
     // WORD2[V6] is clear
    alignment pad = (4 - (SKIP + L2_size)) & 0x7;
}
else
  // WORD2<9> is set in this case
  alignment pad = PIP_GBL_CFG[NIP_SHF];
```

This code uses the following definitions from the pseudo-code in Section 7.2.5:

- **L2_size** is the number of L2 HDR bytes. L2_size is zero when in skip-to-IP mode. Figure 7–6 shows the L2_size values for the skip-to-L2 mode
- **SKIP** is the total number of bytes in any Skip I, PKT_INST_HDR, and Skip II fields, if present.
- **Is_IP** is set for an IP packet.
- **ISRAWFULL** is set for a RAWFULL packet.

Note that **PIP_GBL_CFG[NIP_SHF]** and **PIP_GBL_CFG[RAW_SHF]** are common for all packets from all input ports.

Figures 7–6 and 7–8 depict the packet-data format when PIP/IPD is in big-endian mode. The unit is in big-endian mode for the packet data in the buffers when IPD_CTL_STATUS[PKT_LEND] is clear. In big-endian mode, the unit places sequential packet bytes from most-significant to least-significant within a 64-bit word in an L2/DRAM buffer, and stores the next buf pointers in big-endian format.

When IPD_CTL_STATUS[PKT_LEND] is set, the unit is in little-endian mode, and it instead places sequential packet bytes from least significant to most significant within a 64-bit word in an L2/DRAM buffer, and stores the next pointers in little-endian format. Note that the number of alignment-pad bytes is identical in both big and little endian modes, but the most-significant bytes are the pad bytes in big-endian mode, while the least-significant bytes are the pad bytes in little-endian mode. Note also that IPD_CTL_STATUS[PKT_LEND] does not affect the format of the packet data in the work-queue entry.

Note that PIP/IPD can support a maximum of 255 buffers for a packet and a maximum packet size of 65535 bytes.

7.4 Packet Scheduling

Unless it drops an inbound packet, PIP/IPD creates a work-queue entry for each packet (refer to Sections 7.6 and 7.7 for PIP/IPD drop conditions). The work-queue entry contains the following fields that are important for packet scheduling:

- QOS: Selects one of the POW input queues (one of eight).
- Grp: Selects the group attached to the work (one of sixteen).
- TT: Tag type: NULL, ORDERED, or ATOMIC.
- Tag: 32-bit tag value.

The remainder of this section qualitatively describes how these fields are generated by PIP/IPD. Section 7.5 describes the fields in more detail. Refer to Chapter 5 for more information on work-queue entries and how these fields affect work scheduling.

7.4.1 RAWFULL and RAWSCHED Packets

For RAWFULL and RAWSCHED packets, the QOS, Grp, TT and Tag generated by PIP/IPD are directly specified in fields in the packet instruction header: PKT_INST_HDR[QOS,GRP,TT,TAG].

7.4.2 QOS

The QOS value can be calculated by PIP/IPD in a variety of ways for non-RAWFULL and non-RAWSCHED packets:

- The default QOS value for a packet may be used. This default value can be configured separately for each input port.
- The VLAN priority field can override the default QOS when VLAN is present. This can be enabled on a per-port basis, and there is an eight-entry table to convert the three-bit VLAN priority field into a QOS value. VLAN stacking is supported, and either VLAN priority field may be used when VLAN stacking is used.
- The IP DiffServ field can override the QOS value for IP packets. This can be enabled on a per-port basis, and there is a 64-entry table to convert the DiffServ field (i.e. IP.TOS/CLASS<7:2>) into a QOS value.
- One of the four “QOS/Grp Watchers” can override the QOS value for IP packets that match a protocol/next_header value or TCP/UDP port. Each QOS watcher is individually enabled on a per-port basis.

7.4.3 Grp

The Grp value can be calculated by PIP/IPD in a number of ways for non-RAWFULL and non-RAWSCHED packets:

- The default Grp for the packet may be used. This default value can be configured separately for each port.
- For IP packets, low-order tag bits can override the Grp value. This can be enabled on the per-port basis, and the tag width and base Grp value are also programmable on a per-port basis.
- One of four “Qos/Grp Watchers” can override the Grp value for IP packets that match a protocol/next_header value or TCP/UDP port. Each Grp watcher is individually enabled on a per-port basis.

7.4.4 TT

For IP packets that are neither RAWFULL nor RAWSCHED, there are four per-port configured TT values used by PIP/IPD:

- An IPv4 TCP TT value
- An IPv4 non-TCP value
- An IPv6 TCP TT value
- An IPv6 non-TCP value

For non-IP packets that are neither RAWFULL nor RAWSCHED, there is a per-port configured TT value used by PIP/IPD.

7.4.5 Tag

There are two major tag-generation algorithms that are used for packets that are neither RAWFULL nor RAWSCHED: tuple and mask.

In many cases, the goal of the PIP/IPD tag-generation algorithms is to generate different tags for different flows, so CN50XX can schedule packets for different flows freely to different cores. Tuple-tag generation can create a hash of up to a 7-tuple. Tuple-tag generation is well-suited to IP TCP 5-tuple hash generation, for example. The mask tag is useful for non-IP fixed headers, and also for including additional information from user-defined headers.

The tuple and hash algorithms can be combined in the following ways based on a per-port configured tag mode:

- Always use the tuple tag.
- Always use the mask tag.
- Use the tuple tag for IP packets, and the mask tag for non-IP packets.
- Exclusive-OR (XOR) the tuple and the mask tags.

For an IP packet, the tuple tag is an XOR of a source and destination hash value. This XOR can give the same result for both directions of a TCP flow, for example. The tuple-tag generation scheme also includes a secret configured value that makes it difficult or impossible for potential denial-of-service attackers to predict tag/flow collisions.

The PIP/IPD source tuple-hash value is a CRC of any combination of these two fields, each selected by per-port configuration:

- IP source address
- IP TCP/UDP source port

PIP/IPD zeroes the source tuple-hash value for an initiating TCP SYN packet if selected on a per-port basis.

The PIP/IPD destination tuple-hash value is a CRC of any combination of these fields, each selected by per-port configuration:

- VLAN ID
  - Either or both in the stacked VLAN case
- IP Destination Address
- IP Protocol/next header value
- IP TCP/UDP destination port

The mask tag is a CRC of selected bytes from the first 128 bytes in the packet. The bytes are selected by four 128-bit masks. Each bit in each mask represents one of the first 128-bytes in the packet. (Note specifically that mask-tag generation is not affected by any SKIP value.) Each port can be configured to use any one of the four 128-bit masks.

The resultant tuple/mask tag is up to 16 bits. If desired, fewer bits can be used, generating fewer unique tag values. If desired, the port number can be included in the tag generated by PIP/IPD.

7.5 Work-Queue Entry

The PIP/IPD hardware allocates and creates a work-queue entry for each packet that arrives and that it does not decide to drop. (Sections 7.6 and 7.7 cover the PIP/IPD drop conditions.) The hardware allocates the work-queue entry from a configured hardware free pool (IPD_WQE_FPA_QUEUE[WQE_QUE]). PIP/IPD prequeues a number of pointers to work-queue entries that it receives from FPA. The number of work-queue-entry buffer pointers currently held by IPD is:

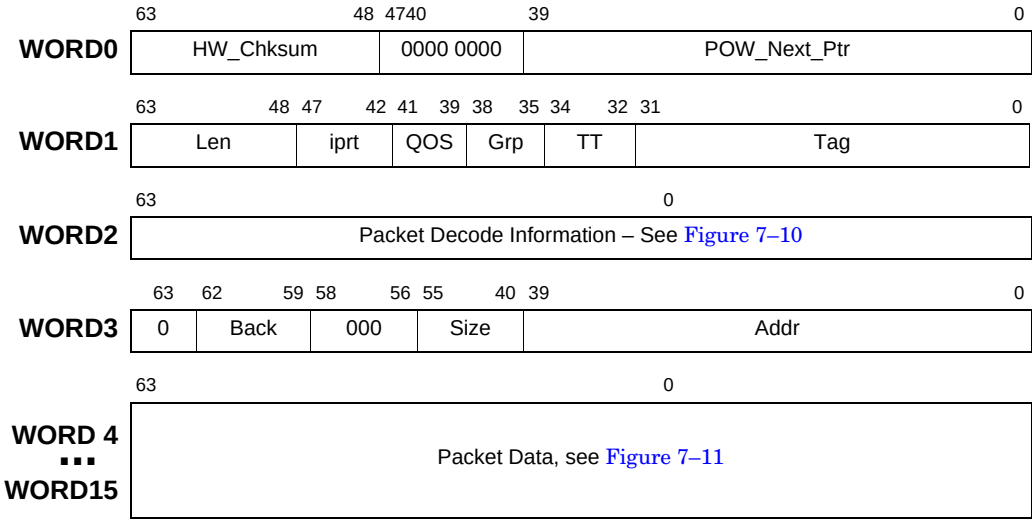
IPD_PTR_COUNT[WQE_PCNT] + IPD_PTR_COUNT[WQEV_CNT].

IPD holds no pointers if it has not yet been enabled.

Figure 7–9 depicts the work-queue entry created by the PIP/IPD hardware for input packets. The work-queue entry is 128 bytes, or 16 64-bit words. WORD0 and WORD1 contain scheduling information and match the format in Figure 5–6, with some additional information added into fields that are not used by the POW unit hardware. WORD2 contains decode information that is described in Figure 7–10. WORD3 contains a pointer to the packet data in the first buffer.

Words 4–15 contain packet data, in different formats as defined in Figure 7–11. When the unit considers the packet to be an IP packet, it places the IP at an offset defined by PIP_IP_OFFSET[OFFSET]. With this flexibility, the particular configuration defines the position of the IP packet in the work-queue entry, and the amount of (pre-IP) packet header data in the work-queue entry.

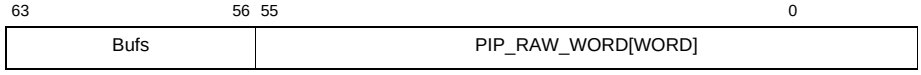
Note that Figures 7–9, 7–10, and 7–11 contain a simple description of the fields in the figure. More-detailed descriptions of the fields are provided in the text following the figures.



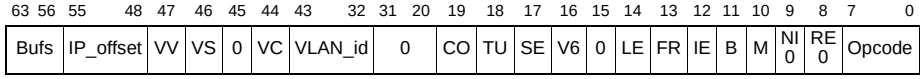
POW_Next_Ptr	Reserved for POW hardware.
HW_Chksum	Hardware-calculated checksum of packet bytes.
Len	Total packet length in bytes.
iprt	Input port number.
QOS	POW input queue used for the work.
Grp	Core group number used for the work.
TT	Tag type used for the work (Atomic, Ordered, or Null)
Tag	Tag used for the work.
Addr	Byte address of start-of-packet in the first buffer.
Size	Size of the first buffer in bytes.
Back	Distance from Addr to beginning of the first buffer (loaded from the configuratio register).

Figure 7–9 PIP/IPD Hardware Work-Queue Entry

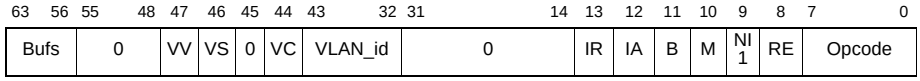
WORD2 (RAWFULL and (No L2/L1 Error))



WORD2 (Is_IP and (No L2/L1 Error))



WORD2 (All Other Cases)



Bufs Number of buffers used to store the packet.

IP_Offset Number of bytes from start-of-packet to the start of IP.

VV Set to 1 when VLAN is found in L2 HDR.

VS Set to 1 when stacked VLAN is present.

VC When VV = 1, the VLAN CFI bit; otherwise 0.

VLAN_id When VV = 1, the VLAN ID field; otherwise 0.

CO Set to 1 when IP packet is IPCOMP.

TU Set to 1 when IP packet is TCP or UDP.

SE Set to 1 when IP may packet require IPSEC decryption.

V6 IP version 6.

LE IP packet has TCP/UDP error.

FR IP packet is a fragment.

IE IP packet error.

B Packet's DMAC is broadcast.

M Packet's DMAC is multicast.

NI Not an IP packet.

RE Packet has L2/L1 error.

IR Packet's Type indicates RARP.

IA Packet's Type indicates ARP.

Figure 7–10 Work-Queue Entry Format; Word 2 Cases

Words 4–15 are unpredictable when WORD2[RE] = 1 and the packet is not RAWFULL.

Figure 7–11 Work-Queue Entry Format; Word 4-15 Cases

Figure 7–11 shows the packet data format in Words 4–15 when the PIP/IPD places the packet data into the work-queue entry in big-endian format, as is the case when IPD_CTL_STATUS[WQE_LEND] is clear. When IPD_CTL_STATUS[WQE_LEND] is set, the unit instead inserts the packet data bytes in Words 4–15 of the work-queue entry in little-endian format, from least-significant to most-significant within each word rather than most-significant to least-significant. In other words, the unit byte-swaps Words 4–15 when IPD_CTL_STATUS[WQE_LEND] is set.

PIP/IPD analyzes the packet length, work-queue-entry alignment configuration, and dynamic alignment to determine if the packet will fit into Words 4–15 of the work-queue-entry or if it requires buffering in L2/DRAM. A packet that meets such criteria is said to be “dynamic short”. Basically, dynamic short means that all packet bytes reside in Words 4–12 of the work-queue entry, so the ordinary L2/DRAM copy of the packet might be considered redundant. In order for a packet to be dynamic short in the IP case, all of the header prior to the IP packet must fit into PIP_IP_OFFSET[OFFSET] words, and all of the IP packet and pad must fit into the remaining Words 4–15 of the work-queue entry. Refer to Figure 7–1 for the definition of pad.

Note that any packet that has an L2/L1 receive error is never dynamic short. Note also that the packet includes any inbound CRC bytes that are not stripped (see Section 7.2.6). Note specifically that for an IPv4 packet, the first four bytes in Word 4 of the work-queue entry can never buffer packet data, so the effective work-queue buffer size is 92 bytes rather than the possible 96 bytes for IPv4, as Figure 7–11 shows.

When PIP/IPD classifies a packet as dynamic short, it will only (redundantly) buffer the packet in the L2/DRAM buffer if it is not enabled to avoid the ordinary L2/DRAM copy. When PIP/IPD does not write the ordinary copy, WORD2[Bufs] is set to 0 to indicate that no buffers have been allocated for the packet (outside the work-queue entry, that is).

PIP/IPD is enabled to avoid the ordinary L2/DRAM copy of dynamic short packets whenever one of the following occurs:

- PIP_PRT_CFG*n*[DYN_RS] = 1, or
- the packet has a PKT_INST_HDR and both of the following are true:
  - PKT_INST_HDR[RS] =1, and
  - either PIP_GBL_CFG[IGNRS] = 0, or the port is a PCI port.

Note that the PIP/IPD hardware ignores both PIP_PRT_CFG*n*[DYN_RS] and PKT_INST_HDR[RS] for packets that the hardware does not classify dynamic short, and always creates ordinary L2/DRAM copies of these packets. Thus, it is legal for PKT_INST_HDR[RS] to be set for large packets. For example, it can be set for all packets, and PIP/IPD will buffer minimally.

The remainder of this section is a description of each field in WORD0 – WORD3 of the work-queue entry. Section 7.2.5 describes the Pre-IP parsing that occurs prior to the pseudo-code fragments included below, and defines Is_IP, ISRAWFULL, and ISRAWSCH.

WORD0[HW_Chksum] Refer to Figure 7–9. A hardware-generated 16-bit one's complement sum of the packet data.

WORD0[HW_Chksum] is the 1's complement sum calculated over the packet from [SKIP + 20B] until the end of packet (including any optional CRC bytes). The packet is padded with a zero octet at the end (if necessary) to make it a multiple of two octets.

Software should not strip the FCS (i.e. IPD_SUB_PORT_FCS[PORT_BIT<n>] should be clear, refer to Section 7.2.6) from ports for which WORD0[HW_Chksum] may be used. This is because the CRC bytes are included in the WORD0[HW_Chksum] value, and software will likely need to reference the calculated CRC value to effectively use the WORD0[HW_Chksum] value.

The starting checksum value is 0. If the packet is less than the start of the checksum, then the checksum will simply be 0.

WORD0[POW_Next_Ptr] Refer to Figure 7–9. Reserved for POW Unit. See Chapter 5.

WORD1[Len] Refer to Figure 7–9. The total number of bytes of packet data, from 1 to 65535. The total packet length includes all the packet bytes shown in the formats in Figure 7–1, except that the CRC field is discarded when IPD_SUB_PORT_FCS[PORT_BIT<n>] = 1. (If IPD_SUB_PORT_FCS[PORT_BIT<n>] = 0, both the Len value and the stored packet (both the work-queue-entry copy and the ordinary copy) include the CRC bytes. See Section 7.2.6 for more details on CRC processing.)

WORD1[Len] exactly matches the number of packet bytes written to the ordinary L2/DRAM copy of the packet, except for dynamic short packets when PIP/IPD avoids the ordinary L2/DRAM copy.

WORD1[iprt] Refer to Figure 7–9. The input port that the packet arrived on, as described in 7.1 on page 266.

WORD1[QOS] Refer to Figure 7–9. The POW unit input queue used for the work. For RAWFULL and RAWSCH packets, the group is PKT_INST_HDR[QOS]. Otherwise, PIP/IPD selects this (differently for each port) based on a number of parameters, including:

- a default QOS value for each port (PIP_PRT_CFG*n*[QOS]).
- an eight-entry table (PIP_QOS_VLAN*n*[VLAN.priority][QOS]) to convert a VLAN priority (from the most-significant three bits of the VLAN field in Figure 7–4) to a QOS value.
- a VLAN enable bit for each port (PIP_PRT_CFG*n*[QOS_VLAN]) that causes the calculated VLAN QOS value to be used when the packet has a VLAN field (as shown in Figure 7–4) in the L2 HDR.
- a VLAN selection bit, used for all ports, that selects which VLAN priority to use when VLAN stacking is detected. (In the pseudo-code below, VLAN0.priority is the first in network order, and VLAN1.priority is the second in network order.)
- a 64-entry table (PIP_QOS_DIFF*n*[IP.TOS/CLASS<7:2>][QOS]) to convert a diffserv DSCP field (the most-significant six bits of IPv4 TOS / IPv6 CLASS) to a QOS value.
- a diffserv-enable bit for each port (PIP_PRT_CFG*n*[QOS_DIFF]) that causes the calculated diffserv QOS value to be used when the packet is IP.
- four QOS watchers (PIP_QOS_WATCH*n*[i]), including a QOS value, that can match IP protocol/next_header values or TCP/UDP destination port numbers.

- an enable bit for each watcher for each port (PIP_PRT_CFG*n*[QOS_WAT<i>]) that causes the QOS watcher QOS value to be used when the QOS watcher matches for the packet.

The following pseudo code details the PIP/IPD work-queue QOS selection for a given packet:

```
QOS = PIP_PRT_CFG[port][QOS];
if (there is an L1/L2 receive error) {
  // WORD2[RE] will be set in this case, as will WORD2[NI]
}
else if(ISRAWFULL || ISRAWSCH) {
 QOS = PKT_INST_HDR[QOS];
}
else {
 if(VV && PIP_PRT_CFG[port][QOS_VLAN]) {
   VLAN = (VS && PIP_GBL_CFG[VS_QOS]) ? VLAN1 : VLAN0;
   QOS = PIP_QOS_VLAN[VLAN<15:13>][QOS]; // VLAN<15:13> is priority
 }
 if(Is_IP && !WORD2[IE] && !WORD2[LE]) {
   // WORD2[NI] is always clear in this case
   if(PIP_PRT_CFG[port][QOS_DIFF] && !(VV && PIP_CRT_CFG[port][QOS_VLAN] &&
      PIP_CRT_CFG[port][QOS_VOD]))
     QOS = PIP_QOS_DIFF[IP.TOS/CLASS<7:2>][QOS]; // the 6 msbs of TOS/CLASS
   for (i=0; i<4; i++) {
     if (PIP_PRT_CFG[port][QOS_WAT<i>] and PIP_QOS_WATCH[i][TYPE]!=0) {
        mask16  = ~PIP_QOS_WATCH[i][MASK] & 0xffff;
        mask8   = ~PIP_QOS_WATCH[i][MASK] &   0xff;
        match16 = PIP_QOS_WATCH[i][MATCH] & mask16;
        match8  = PIP_QOS_WATCH[i][MATCH] & mask8;
        if(PIP_QOS_WATCH[i][TYPE] is protocol/next_header and
              match8 equals (IP protocol/next header field)&mask8) {
           QOS = PIP_QOS_WATCH[i][WATCHER];
           break;
        }
        else if(PIP_QOS_WATCH[i][TYPE] is TCP and
              IP protocol/next header field is TCP and
              match16 equals (TCP destination port field)&mask16 and
              !WORD2[FR] and
              (WORD2[V6] || (ipv4.HL==5))) {
           QOS = PIP_QOS_WATCH[i][WATCHER];
           break;
        }
        else if(PIP_QOS_WATCH[i][TYPE] is UDP and
              IP protocol/next header field is UDP and
              match16 equals (UDP destination port field)&mask16 and
              WORD2[FR] and
              (WORD2[V6] || (ipv4.HL==5))) {
           QOS = PIP_QOS_WATCH[i][WATCHER];
           break;
        }
     }
   }
 }
```

Note that the QOS watchers will resolve in watcher order if more than one watcher matches the packet with different PIP_QOS_WATCH*n*[WATCHER] values. Note also that for IPv6, the IP next-header field is the initial next-header field.

WORD1[Grp] Refer to Figure 7–9. The core group number used for the work. For RAWFULL and RAWSCH packets, the group is PKT_INST_HDR[GRP], else this group value comes from PIP_PRT_TAG*n*[GRP], a per-port configuration variable.

- There is a default GRP value for each port (PIP_PRT_TAG[port][Grp] below).

- For IP packets, there is an enable bit per port (PIP_PRT_TAG[port][GRPTAG]) that causes PIP/IPD to set WORD1[Grp] to the least-significant bits of WORD1[Tag] (translated).
- There are four GRP watchers (PIP_QOS_WATCH*n*[i] below), including a GRP value, that can match IP protocol/next_header values or TCP/UDP destination port numbers.
- There are eight GRP watchers (PIP_QOS_WATCHn[i] below), including a GRP value, that can match IP protocol/next_header values, TCP/UDP destination port numbers, or ethertype.
- There is an enable bit for each watcher for each port (PIP_PRT_CFG*n*[GRP_WAT<i>]) that causes the GRP watcher GRP value to be used when the GRP watcher matches for the packet.

```
Grp = PIP_PRT_TAG[port][GRP];
if (there is an L1/L2 receive error) {
  // WORD2[RE] will be set in this case, as will WORD2[NI]
}
else if(ISRAWFULL || ISRAWSCH) {
  Grp = PKT_INST_HDR[GRP];
}
else if(Is_IP) {
  // WORD2[NI] is always clear in this case
  if(PIP_PRT_TAG[port][GRPTAG]) {
    Grp = ((WORD1[Tag] & ~PIP_PRT_TAG[GRPTAGMASK])
        + PIP_PRT_TAG[GRPTAGBASE]) & 0xF;
  }
  if(!WORD2[IE] && !WORD2[LE]) {
    for (i=0; i<4; i++) {
      if (PIP_PRT_CFG[port][GRP_WAT<i>] and PIP_QOS_WATCH[i][TYPE]!=0) {
         mask16 = ~PIP_QOS_WATCH[i][MASK] & 0xffff;
         mask8 = ~PIP_QOS_WATCH[i][MASK] & 0xff;
         match16 = PIP_QOS_WATCH[i][MATCH] & mask16;
         match8 = PIP_QOS_WATCH[i][MATCH] & mask8;
         if(PIP_QOS_WATCH[i][TYPE] is protocol/next_header and
           match8 equals (IP protocol/next header field)&mask8) {
            Grp = PIP_QOS_WATCH[i][GRP];
            break;
         }
         else if(PIP_QOS_WATCH[i][TYPE] is TCP and
              IP protocol/next header field is TCP and
              match16 equals (TCP destination port field)&mask16 and
              !WORD2[FR] and
              (WORD2[V6] || (ipv4.HL==5))) {
            Grp = PIP_QOS_WATCH[i][GRP];
            break;
         }
         else if(PIP_QOS_WATCH[i][TYPE] is UDP and
              IP protocol/next header field is UDP and
              match16 equals (UDP destination port field)&mask16 and
              !WORD2[FR] and
              (WORD2[V6] || (ipv4.HL==5))) {
            Grp = PIP_QOS_WATCH[i][GRP];
            break;
         }
      }
    }
  }
}
```

Note that the GRP watchers resolve in watcher order if more than one watcher matches the packet with different PIP_QOS_WATCH*n*[GRP] values. Note also that for IPv6, the IP next header field is the initial next header field.

WORD1[TT] Refer to Figure 7–9. The initial tag type, (NULL, ORDERED, or ATOMIC), for the work-queue entry.

For both RAWFULL and RAWSCH packets, TT is PKT_INST_HDR[TT].

For other packets, there are per-port configuration variables (PIP_PRT_TAG*n* below) that control the tag type. The tag type can be:

- different for IP and non-IP packets
- different for IPv4 and IPv6 packets
- different for TCP and non-TCP packets

The following pseudo code details the hardware behavior:

```
// zero-extend the two-bit fields to make the three-bit TT field
TT = PIP_PRT_TAG[port][NON_TAG]
if(there is a L2/L1 receive error) {
  // WORD2[RE] will be set, as will WORD2[NI]
}
else if(ISRAWFULL || ISRAWSCH)
  TT = PKT_INST_HDR[TT]
else if(Is_IP) {
  if(WORD2[V6])
    proto_nh = IPv6.next_header; // the initial next header
  else
    proto_nh = IPv4.protocol;
  // WORD2[NI] will always be clear here
  if(!WORD2[IE] and (proto_nh==6)) {
    if(WORD2[V6])
      TT = PIP_PRT_TAG[port][TCP6_TAG]
    else
      TT = PIP_PRT_TAG[port][TCP4_TAG]
  }
  else {
    if(WORD2[V6])
      TT = PIP_PRT_TAG[port][IP6_TAG]
    else
      TT = PIP_PRT_TAG[port][IP4_TAG]
  }
}
```

WORD1[Tag] Refer to Figure 7–9. The initial tag for the work-queue entry.

For RAWFULL and RAWSCH packets, Tag is PKT_INST_HDR[TAG].

For other packets:

- Tag<31:24> is always 0x0.
- Tag<23:16> is either the port number that the packet arrived on or all 1s.
- Tag<15:0> is a hash of specific fields within the packet.

PIP_PRT_TAG*n*[TAG_MODE] picks from among various combinations of the two different tag generation algorithms.

The first tag generation algorithm is the tuple hash. The tuple hash optionally includes the IP source and destination addresses, the IP protocol/next_header value, the TCP/UDP source and destination ports, and VLAN ID. There are per-port configuration variables (PIP_PRT_TAG) that control these options. The IPv4 and IPv6 configuration variables are separate. The tuple hash calculation also includes a secret value. The tuple hash results and, most importantly, potential collisions between different tuple hash results change with different secret values.

The second tag generation algorithm is the mask hash. The mask hash can optionally include any user-defined range of bytes within the first 128B of a packet. This is accomplished with four 128-bit masks (PIP_TAG_INC*). Each bit in each mask represents one of the first 128 bytes in the packet. (Note specifically that the mask hash is NOT affected by any SKIP values - each 128-bit mask refers strictly to the first 128 bytes in the packet.) Each port can be configured to use any of the 4 masks with PIP_PRT_CFG*n*[TAG_INC]. PIP_PRT_TAG*n*[TAG_MODE] picks the tuple hash, the mask hash, or a combination of the two hashes.

The following C-like pseudo code details precisely how PIP/IPD creates the Tag field:

```
Tag = 0;
if(there is a L2/L1 receive error) {
  // WORD2[RE] will be set, as will WORD2[NI]
  inc_port = true;
}
else if(ISRAWFULL || ISRAWSCH) {
  inc_port = false;
  Tag = PKT_INST_HDR[TAG];
}
else {
  inc_port = true;
  if(PIP_PRT_TAG[port][TAG_MODE] == 0)        // always tuple tag
    Tag<15:0> = hw_tuple_tag();
  else if(PIP_PRT_TAG[port][TAG_MODE] == 1)   // always mask tag
    Tag<15:0> = hw_mask_tag();
  else if(PIP_PRT_TAG[port][TAG_MODE] == 2) { // Is_IP ? tuple tag : mask tag
    if(Is_IP) // i.e. is WORD2[9] clear
       Tag<15:0> = hw_tuple_tag();
    else
       Tag<15:0> = hw_mask_tag();
  }
  else                                        // mask XOR tuple
    Tag<15:0> = hw_mask_tag() ^ hw_tuple_tag();
}
if(inc_port) {
  if(PIP_PRT_TAG[port][INC_PORT]) Tag<23:16> = port;
  else Tag<23:16> = 0xFF;
}

uint32 hw_tuple_tag() {
  Tag = 0;
  if(Is_IP) {
    // WORD2[NI] will always be clear here
    if(WORD2[V6])
      Tag = hw_ipv6_hash(port, WORD2, IP source address, IP destination address,
               initial IP next_header, tcp/udp source port, tcp/udp destination port,
                TCP_flag_SYN, TCP_flag_ACK, VLAN0, VLAN1);
    else
      Tag = hw_ipv4_hash(port, WORD2, IP source address, IP destination address,
                IP protocol, tcp/udp source port, tcp/udp destination port,
                TCP_flag_SYN, TCP_flag_ACK, VLAN0, VLAN1);
  }
  return(Tag);
}

uint16 crc16(uint16 iv, uint64 p) {
  int i;
  int j;
  uint   b;
  uint16 poly = 0x1021;

  for (i=7; i>=0; i--) {
    b = (p >> (i*8)) & 0xff;
    for (j=7; j>=0; j--) {
      iv = ((iv << 1) ^ ((((iv >> 15)&1) ^ ((b >> j)&1)) ? poly : 0))
           & 0xffff;
    }
  }
  return iv;
}
```

```
uint16 hw_ipv4_hash(uint8 port, uint64 WORD2, uint32 IPsrc, uint32 IPdest,
                    uint8 protocol, uint16 tcp_udp_sprt, uint16 tcp_udp_dprt,
                  bool TCP_flag_SYN, TCP_flag_ACK,
                  uint16 VLAN0, uint16 VLAN1)
{
  uint16 src_crc  = 0xffff;
  uint16 dst_crc  = 0xffff;
  uint16 prot_crc = 0xffff;
  uint16 result   = 0;

  uint16 src_secret16 = (PIP_TAG_SECRET[SRC]);
  uint32 src_secret32 = (((uint32)src_secret16) << 16) | src_secret16;

  uint16 dst_secret16 = (PIP_TAG_SECRET[DST]);
  uint32 dst_secret32 = (((uint32)dst_secret16) << 16) | dst_secret16;

  // WORD2[NI] will always be clear here
  if(!WORD2[IE]) {
    if(PIP_PRT_TAG[port][IP4_SRC])
      src_crc = crc16(src_crc, ((uint64)IPsrc+src_secret32)<<32);
    if(PIP_PRT_TAG[port][IP4_DST])
      dst_crc = crc16(dst_crc, ((uint64)IPdest+dst_secret32)<<32);
    if (   PIP_PRT_TAG[port][IP4_PCTL]
        || (WORD2[VV] && !WORD2[VS] && PIP_PRT_TAG[port][INC_VLAN])
        || (WORD2[VS] &&               PIP_PRT_TAG[port][INC_VS]  )) {
      uint64 pctl_vlan_dat = 0;
      if(PIP_PRT_TAG[port][IP4_PCTL])
        pctl_vlan_dat |= (((uint64)protocol)<<56);
      if (   (WORD2[VV] && !WORD2[VS] && PIP_PRT_TAG[port][INC_VLAN] )
          || (WORD2[VS] &&              (PIP_PRT_TAG[port][INC_VS]&1))) {
        vlan0 = 0x8000 | VLAN0<11:0>;
        pctl_vlan_dat |= (((uint64)vlan0)<<32);
      }
      if (WORD2[VS] && (PIP_PRT_TAG[port][INC_VS]&2)) {
        vlan1 = 0x8000 | VLAN1<11:0>;
        pctl_vlan_dat |= (((uint64)vlan1)<<16);
      }
      prot_crc = crc16(prot_crc, pctl_vlan_dat);
    }
    if(WORD2[TU] && !WORD2[FR] && !WORD2[LE] && (IPv4.HL==5)) {
      if(PIP_PRT_TAG[port][IP4_SPRT])
        src_crc = crc16(src_crc, ((uint64)tcp_udp_sprt+src_secret16)<<48);
      if(PIP_PRT_TAG[port][IP4_DPRT])
        dst_crc = crc16(dst_crc, ((uint64)tcp_udp_dprt+dst_secret16)<<48);
      if(PIP_GBL_CFG[TAG_SYN] && protocol==TCP
              && TCP_flag_SYN && !TCP_flag_ACK)
        src_crc = 0;
    }
    result = (src_crc ^ dst_crc ^ prot_crc) & ~PIP_TAG_MASK[MASK];
  }
  return(result);
}

uint32 hw_ipv6_hash(uint8 port, uint64 WORD2, uint64 IPsrc[2], uint64 IPdest[2],
                    uint8 next_header, uint16 tcp_udp_sprt, uint16 tcp_udp_dprt,
                    bool TCP_flag_SYN, TCP_flag_ACK,
                    uint16 VLAN0, uint16 VLAN1)
{
  uint16 src_crc  = 0xffff;
  uint16 dst_crc  = 0xffff;
  uint16 prot_crc = 0xffff;
  uint16 result   = 0;

  uint16 src_secret16 = (PIP_TAG_SECRET[SRC]);
  uint32 src_secret32 = (((uint32)src_secret16) << 16) | src_secret16;

  uint16 dst_secret16 = (PIP_TAG_SECRET[DST]);
  uint32 dst_secret32 = ()(uint32)dst_secret16) << 16) | dst_secret16;

  // WORD2[NI] will always be clear here
  if(!WORD2[IE]) {
    if(PIP_PRT_TAG[port][IP6_SRC]) {
```

```
      src_crc = crc16(src_crc,    ((IPsrc[0]+src_secret32)&0xffffffff)
                                | (((IPsrc[0]>>32)+src_secret32)<<32));
      src_crc = crc16(src_crc,    ((IPsrc[1]+src_secret32)&0xffffffff)
                                | (((IPsrc[1]>>32)+src_secret32)<<32));
    }
    if(PIP_PRT_TAG[port][IP6_DST]) {
      dst_crc = crc16(dst_crc,    ((IPdest[0]+dst_secret32)&0xffffffff)
                                | (((IPdest[0]>>32)+dst_secret32)<<32));
      dst_crc = crc16(dst_crc,    ((IPdest[1]+dst_secret32)&0xffffffff)
                                | (((IPdest[1]>>32)+dst_secret32)<<32));
    }
    if (   PIP_PRT_TAG[port][IP6_NXTH]
        || (WORD2[VV] && !WORD2[VS] && PIP_PRT_TAG[port][INC_VLAN])
        || (WORD2[VS] &&               PIP_PRT_TAG[port][INC_VS] )) {
      uint64 nxth_vlan_dat = 0;
      if(PIP_PRT_TAG[port][IP6_NXTH])
        nxth_vlan_dat |= (((uint64)next_header)<<56);
      if (   (WORD2[VV] && !WORD2[VS] && PIP_PRT_TAG[port][INC_VLAN] )
          || (WORD2[VS] &&              (PIP_PRT_TAG[port][INC_VS]&1))) {
        vlan0 = 0x8000 | VLAN0<11:0>;
        nxth_vlan_dat |= (((uint64)vlan0)<<32);
      }
      if (WORD2[VS] && (PIP_PRT_TAG[port][INC_VS]&2)) {
        vlan1 = 0x8000 | VLAN1<11:0>;
        nxth_vlan_dat |= (((uint64)vlan1)<<16);
      }
      prot_crc = crc16(prot_crc, nxth_vlan_dat);
    }
    if(WORD2[TU] && !WORD2[FR] && !WORD2[LE]) {
      if(PIP_PRT_TAG[port][IP6_SPRT]) src_crc =
        crc16(src_crc, ((uint64)tcp_udp_sprt+src_secret16)<<48);
      if(PIP_PRT_TAG[port][IP6_DPRT]) dst_crc =
        crc16(dst_crc, ((uint64)tcp_udp_dprt+dst_secret16)<<48);
      if(PIP_GBL_CFG[TAG_SYN] && next_header==TCP
              && TCP_flags_SYN && !TCP_flags_ACK)
        crc_src = 0;
    }
    result = (src_crc ^ dst_crc ^ prot_crc) & ~PIP_TAG_MASK[MASK];
  }
  return(result);
}

// NOTE: pkt_dat is from the first byte - no SKIP is applied
uint16 hw_mask_tag(uint16 pkt_len, uint8* pkt_dat, uint8 port)
{
  uint16 crc    = 0xffff;
  uint16 result = 0;
  int i;
  int j;
  uint16 pkt_cnt = 0;
  for (i=0; i<16; i++) {
    uint8 tag_inc = PIP_TAG_INC[i + (PIP_PRT_CFG[port][TAG_INC]*16)];
    uint8 mod_inc = 0;
    if ((pktLen & 7) == 0) {
      if (pktLen > (i*8)) {
        mod_inc = 0xff;
      }
    }
    else {
      if ( (pktLen>>3) == i )
        mod_inc = (0xff << (8-(pktLen&7)));
      else if ( (pktLen>>3) > i)
        mod_inc = 0xff;
    }
    if((tag_inc & mod_inc) != 0) {
      uint64 dat = 0;
      for (j=7; j>=0; j--, pkt_cnt++)
        dat |= (((tag_inc&mod_inc)>>j)&1) ?
                  ((uint64)pkt_dat[pkt_cnt] << (j*8)) : 0;
      crc16(crc, dat);
    }
    else
      pkt_cnt += 8;
```

```
  }
  result = crc & ~PIP_TAG_MASK[MASK];

  return(result);
}
```

WORD3[Back, Size, Addr] Refer to Figure 7–9. Fields in the first buffer pointer. Figure 7–7 details these fields. Back, Size, and Addr are 0 when there is no ordinary DRAM copy of the packet, which happens when the packet is dynamic short and PIP/IPD is enabled to avoid the ordinary L2/DRAM copy.

WORD2[Bufs] Refer to Figure 7–10. The number of buffers used to store the packet data. Any value for this field is possible. The value 0 happens (only) for dynamic short packets when PIP/IPD is enabled to avoid the ordinary L2/DRAM copy, and it indicates that PIP/IPD created no ordinary L2/DRAM of the packet.

WORD2[IP_offset] Note that WORD2[IP_offset] is not present when there is an L2/L1 receive error, nor when the packet is RAWFULL, nor when the packet is not IP.

Refer to Figure 7–10. The number of bytes from the first byte of packet data to the first byte of the IP packet.

WORD2[VV] Note that WORD2[VV] is not present for a RAWFULL packet.

Refer to Figure 7–10. Indicates that one or more VLAN fields were found in the L2 HDR for the packet. This bit asserts only when in skip-to-L2 mode and the L2 HDR type is Ethernet II + VLAN (stacked) or IEEE 802.3 + VLAN (stacked).

Also see the VV definition in the pseudo-code in Section 7.2.5 and Figure 7–4. WORD2[VV] is unpredictable when WORD2[RE] = 1.

WORD2[VS] Note that WORD2[VS] is not present for a RAWFULL packet.

Refer to Figure 7–10. Indicates that multiple VLAN fields were found in the L2 HDR for the packet. This bit asserts only when in skip-to-L2 mode and the L2 HDR type is Ethernet II + VLAN Stacked or IEEE 802.3 + VLAN Stacked.

Also see the VS definition in the pseudo-code in Section 7.2.5 and Figure 7–4. WORD2[VS] is unpredictable when WORD2[RE] = 1 (and the packet is not RAWFULL).

WORD2[VC] Note that WORD2[VC] is not present for a RAWFULL packet.

Refer to the VLAN CFI bit (i.e. VLAN<12>) in Figure 7–10.

If WORD2[VS] = 1, PIP_GBL_CTL[VS_WQE] selects which of the two VLAN fields (VLAN0 or VLAN1, as described in Section 7.2.5 and Figure 7–4) the CFI comes from.

- if PIP_GBL_CTL[VS_WQE] = 0, VLAN0 is selected.
- if PIP_GBL_CTL[VS_WQE] = 1, VLAN1 is selected.

WORD2[VC] is 0 when WORD2[VV] = 0, and is unpredictable when WORD2[RE] = 1 (and the packet is not RAWFULL).

WORD2[VLAN_id] Note that WORD2[VLAN_id] is not present for a RAWFULL packet.

Refer to the VLAN ID field (i.e. VLAN<11:0>) in Figure 7–10.

If WORD2[VS] = 1, PIP_GBL_CTL[VS_WQE] selects which of the two VLAN fields (VLAN0 or VLAN1, as described in Section 7.2.5 and Figure 7–4) is present here.

- if PIP_GBL_CTL[VS_WQE] = 0, VLAN0 is selected.
- if PIP_GBL_CTL[VS_WQE] = 1, VLAN1 is selected.

This field is 0x0 when WORD2[VV] is clear, and is unpredictable when WORD2[RE] is set (and the packet is not RAWFULL).

WORD2[CO] Note that WORD2[CO] is not present when there is an L2/L1 receive error, nor when the packet is RAWFULL, nor when the packet is not IP.

Refer to Figure 7–10. Set when the IP packet is IPCOMP (i.e. when the IPv4 protocol value or the initial IPv6 next header equals 108). This bit is always clear when WORD2[IE] = 1, and when WORD2[V6] and WORD2[FR] are both set.

WORD2[TU] Note that WORD2[TU] is not present when there is an L2/L1 receive error, nor when the packet is RAWFULL, nor when the packet is not IP.

Refer to Figure 7–10. Set when the IP packet is TCP or UDP (i.e. when the IPv4 protocol value or the initial IPv6 next header equals 6 (TCP) or 17 (UDP)). This bit is always clear when WORD2[IE] = 1, and when WORD2[V6] and WORD2[FR] are both set.

WORD2[SE] Note that WORD2[SE] is not present when there is an L2/L1 receive error, nor when the packet is RAWFULL, nor when the packet is not IP.

Refer to Figure 7–10. Set when the IP packet may require IPSEC decryption. This bit gets set in a number of ways:

- The packet is IPSEC ESP (i.e. the IPv4 protocol value or the initial IPv6 next header equals 50).
- The packet is IPSEC AH (i.e. the IPv4 protocol value or the initial IPv6 next header equals 51).
- The packet is TCP (i.e. the IPv4 protocol value or the initial IPv6 next header equals 6) and the TCP destination port matches one of four possible programmed values and (WORD2[V6] || (IPv4.HL==5)).
- The packet is UDP (i.e. the IPv4 protocol value or the initial IPv6 next header equals 17) and the UDP destination port matches one of four possible programmed values and (WORD2[V6] || (IPv4.HL==5)).

There are four programmable destination port values (PIP_DEC_IPSEC[0..3]) shared by TCP and UDP. Each programmed port can match TCP and/or UDP. WORD2[SE] = 0 whenever any of WORD2[IE], WORD2[FR], or WORD2[LE] are set.

WORD2[V6] Note that WORD2[V6] is not present when there is an L2/L1 receive error, nor when the packet is RAWFULL, nor when the packet is not IP.

Refer to Figure 7–10. This bit is set when the IP version number field is 6.

WORD2[LE] Note that WORD2[LE] is not present when there is an L2/L1 receive error, nor when the packet is RAWFULL, nor when the packet is not IP.

Refer to Figure 7–10. This bit is set when WORD2[TU] is set and the PIP/IPD hardware found an error in the TCP/UDP header and/or data. When this bit is set, WORD2[Opcode] contains one of the following possible (non-zero) codes:

1 = Malformed L4:

IPv4/TCP

- ipv4_total_length < IP_header(20B) + (TCP_data_offset × 4)

  or

- TCP_data_offset < 5

IPv6/TCP

- ipv6_payload_length < TCP_data_offset × 4

  or

- TCP_data_offset < 5

IPv4/UDP

- ipv4_total_length < IP_header(20B) + UDP_header(8B)

IPv6/UDP

- ipv6_payload_length < UDP_header(8B)

2 = L4 checksum error: the L4 checksum value is bad.

NOTE: For a UDP packet that has a UDP length (specified in the UDP L4 header) that indicates a larger packet than the IP length indicates (specified in the IP L3 header), either a UDP L4 checksum error (WORD2[Opcode] = 2) or a UDP length error (WORD2[Opcode] = 3) is possible.

3 = UDP length error: The UDP length field would make the UDP data longer than what remains in the IP packet (as defined by the IP header length field).

IP/TCP

- Will not assert

IPv4/UDP

- ipv4_total_length < IP_header(20B) + UDP_length

IPv6/UDP

- ipv6_payload_length < UDP_length

NOTE: The hardware does not consider it an error when the UDP length field makes the UDP data shorter than the IP packet.

4 = Bad L4 Port: either the source or destination TCP/UDP port is 0.

8 = TCP FIN Only: the packet is TCP and only the FIN flag set.

9 = TCP No Flags: the packet is TCP and no flags are set.

10 = TCP FIN RST: the packet is TCP and both FIN and RST are set.

11 = TCP SYN URG: the packet is TCP and both SYN and URG are set.

12 = TCP SYN RST: the packet is TCP and both SYN and RST are set.

13 = TCP SYN FIN: the packet is TCP and both SYN and FIN are set.

NOTE: WORD2[LE] is never set when WORD2[IE] or WORD2[FR] are set, and is never set for IPv4 packets when (IPv4.HL ≠ 5).

WORD2[FR] Note that WORD2[FR] is not present when there is an L2/L1 receive error, nor when the packet is RAWFULL, nor when the packet is not IP.

Refer to Figure 7–10. Set when the packet is a fragment.

For IPv4, this bit is set when either the MF bit is set or the offset field is non-zero.

For IPv6, this bit is set when the initial next header value is fragmentation (i.e. 44). WORD2[FR] is never set when WORD2[IE] is set.

WORD2[IE] Note that WORD2[IE] is not present when there is an L2/L1 receive error, nor when the packet is RAWFULL, nor when the packet is not IP.

Refer to Figure 7–10. Set when the packet has an IP exceptional condition. When this bit is set, WORD2[Opcode] contains one of the following possible (non-zero) codes:

1 = Not IP: the IP version field is neither 4 nor 6.

2 = IPv4 header checksum error: the IPv4 header has a checksum violation. Note that the CN50XX hardware does not check IPv4-header checksums for IPv4 packets with options (i.e. HLEN > 5.

3 = IP malformed header: the packet is not long enough to contain the IP header.

IPv6 packet

- received packet length – crc < SKIP+L2_Size+IPv6_header(40 bytes)

not IPv6 packet (IPv4 packet)

- received packet length – crc < SKIP+L2_Size+IPv4_header(20 bytes)
  or
- ipv4_header_length < 5
  or
- ipv4_total_length < 20

Note that CN50XX's IPv4 malformed-header check assumes that the IPv4 header is 20 bytes. This means that for IPv4 packets with options (i.e. HLEN > 5), the hardware may not signal an IP malformed error even though the header size is larger than the IP packet size. Thus, as part of the IPv4 options handler, software should check that HLEN × 4 does not exceed the IP packet size.

4 = IP malformed: the packet is not long enough to contain the bytes indicated by the IP header. Pad is allowed.

NOTE: The IP length can legally indicate fewer bytes than are in the packet

IPv6 packet

- received packet length – crc < SKIP+L2_Size+IPv6_header(40)+ipv6_payload_length

not IPv6 packet (IPv4 packet)

- received packet length – crc < SKIP+L2_Size+IPv4_total_length

5 = IP TTL hop: the IPv4 TTL field or the IPv6 hop count field are zero.

6 = IP options:

For IPv4, the packet has options (i.e. the IP HLEN field is not 5. Note that the case where HLEN < 5 is flagged in the IP malformed-header check). Also note that the CN50XX hardware does not check header checksums of IPv4 packets with options and only partially performs an IP malformed-header check.

Upon receipt of an IPv4 packet with an IP-options exception, software should perform an IPv4-header checksum check, and check that HLEN × 4 does not exceed the IP length indicated in the IPv4 header.

For IPv6, the packet has early extension headers.
When PIP_GBL_CTL[IP6_EEXT<1>] = 1, this violation occurs when the (initial) next header field in the IPv6 header does not contain one of the following values:

- 6 (TCP)
- 17 (UDP)
- 44 (fragmentation)
- 58 (ICMP)
- 50 (ESP)
- 51 (AH)
- 108 (IPCMP)

When PIP_GBL_CTL[IP6_EEXT<1>] = 0 and PIP_GBL_CTL[IP6_EEXT<0>] = 1, this violation occurs when the (initial) next header field in the IPv6 header contains one of the following values:

- 0 (hop-by-hop)
- 60 (destination)
- 43 (routing)

WORD2[B] Note that WORD2[B] is not present when there is not an L2/L1 receive error and the packet is RAWFULL.

Refer to Figure 7–10. Set when the packet's destination MAC address field in the L2 HDR is the broadcast address (i.e. all 1s). This bit is unpredictable when WORD2[RE] is set, and always clear when not in skip-to-L2 mode.

WORD2[M] Note that WORD2[M] is not present when there is not an L2/L1 receive error and the packet is RAWFULL.

Refer to Figure 7–10. Set when the packet's destination MAC address field in the L2 HDR is a multicast address (i.e. the group bit is set, and at least one of the remaining bits is a zero). This bit is unpredictable when WORD2[RE] is set, and always clear when not in skip-to-L2 mode.

WORD2[NI] Note that WORD2[NI] is not present when the packet is RAWFULL.

Refer to Figure 7–10. Set when the packet is not an IP packet or has L2/L1 errors.

WORD2[RE] Note that WORD2[RE] is not present when there is not an L2/L1 receive error and the packet is RAWFULL.

Refer to Figure 7–10. Set when the packet had an L2/L1 receive error. Some of these errors are generated by PIP/IPD and others are generated by GMX. When WORD2[RE] is set, WORD2[Opcode] contains one of the following possible (non-zero) codes:

1 = partial error: a packet was partially received, but internal buffering/bandwidth was not adequate to receive the entire packet.

2 = jabber error: the RGMII/GMII/MII packet was too large and is truncated.

3 = overrun error: the RGMII/GMII/MII packet is longer than allowed and had an FCS error.

4 = oversize error: the RGMII/GMII/MII packet is longer than allowed.

5 = alignment error: the RGMII/GMII/MII packet is not an integer number of bytes and had an FCS error (100M and 10M only).

6 = fragment error: the RGMII/GMII/MII packet is shorter than allowed and had an FCS error.

7 = GMX FCS error: the RGMII/GMII/MII packet had an FCS error.

8 = undersize error: the RGMII/GMII/MII packet is shorter than allowed.

9 = extend error: the RGMII packet had an extend error (100M only for RGMII, half-duplex mode only for GMII/MII).

10 = length mismatch error: the RGMII/GMII/MII packet had a length that did not match the length field in the L2 HDR.

11 = RGMII/GMII/MII RX error: the RGMII/GMII/MII packet had one or more data reception errors (RXERR).

12 = RGMII/GMII/MII skip error: the RGMII/GMII/MII packet was not large enough to cover the skipped bytes

13 = RGMII nibble error: the RGMII packet had a stutter error (data not repeated - 10/100M only)

17 = PIP Skip error: a packet was not large enough to cover the skipped bytes.

18 = L2 header malformed: the packet is not long enough to contain the L2 header.

WORD2[Opcode] Note that WORD2[Opcode] is not present when there is not an L2/L1 receive error and the packet is RAWFULL.

Normally zero, but contains a (non-zero) exception opcode when any one of WORD2[RE], WORD2[IE], or WORD2[LE] are set.

WORD2[IR] Note that WORD2[IR] is not present when there is not an L2/L1 receive error and the packet is RAWFULL or IP.

Refer to Figure 7–10. Set when the packet's L2 HDR TYPE field indicates RARP (i.e. equals 0x0835). This bit is unpredictable when WORD2[RE] is set, and always clear when not in skip-to-L2 mode.

WORD2[IA] Note that WORD2[IA] is not present when there is not an L2/L1 receive error and the packet is RAWFULL or IP.

Refer to Figure 7–10. Set when the packet's L2 HDR TYPE field indicates ARP (i.e. equals 0x0806). This bit is unpredictable when WORD2[RE] is set, and always clear when not in skip-to-L2 mode.

7.6 Input Packet Data Unit (IPD) Quality of Service

There are many facets to the packet input quality of service capabilities of CN50XX in general. This section and the next focus on CN50XX's backpressure and packet drop capabilities. The PIP/IPD unit implements important components of CN50XX's backpressure and packet drop capabilities.

First, when there are no buffers available for PIP/IPD from the hardware pool 0 (i.e. when IPD_QUE0_FREE_PAGE_CNT and/or FPA_QUE(0..7)_AVAILABLE are almost zero) or from the FPA pool selected by IPD_WQE_FPA_QUEUE (i.e. the configured pool that PIP/IPD allocates work-queue entries from), the PIP/IPD hardware stops accepting input packet data from any of RGMII/GMII/MII or PCI sources, until buffers become available in the hardware free pool. This may cause GMX/SPX (i.e. the RGMII/GMII/MII interface logic) to drop packets indiscriminately and/or backpressure the ports on the interfaces, and will cause the PCI interface hardware to stop fetching instructions and input packet data for the four PCI input ports.

In many systems, it will be desirable to avoid this buffer exhaustion situation. There are multiple CN50XX mechanisms that can keep the input packets from exhausting the available buffer space.

One mechanism is per-port backpressure. This can prevent the packet from arriving at PIP/IPD, and, thus, may prevent CN50XX from dropping packets. The PIP/IPD hardware maintains a counter per port (IPD_PORT_BP_COUNTERS_PAIR*n*) that PIP/IPD may increment when PIP/IPD buffers packets. When a packet arrives and PIP/IPD creates a work-queue entry for the packet, PIP/IPD adds to the counter for a port according to the following table:

IPD_CTL_STATUS		
[NADDBUF]	[ADDPKT]	Add to Counter
0	0	WORD2[Bufs]
0	1	WORD2[Bufs] + 1
1	0	Undefined
1	1	1

Thus, the per-port counter can simply track FPA pool 0 buffers used, packets arrived, or the total of both. Note that when IPD_CTL_STATUS[ADDPKT] is clear, IPD_PORT_BP_COUNTERS_PAIR*n* counts only buffers and not packets, and PIP/IPD does not change IPD_PORT_BP_COUNTERS_PAIR*n* for dynamic short packets that are solely buffered in the work-queue entry.

When IPD_PORT_BP_COUNTERS_PAIR*n* exceeds the threshold for the port (IPD_PORT(0/1/2)_BP_PAGE_CNT[PAGE_CNT] is the threshold), the threshold check is active, and PIP/IPD can backpressure the port when backpressure is enabled for the port. A 0→1 transition on IPD_CTL_STATUS[PBP_EN] is necessary to enable per-port backpressure. Each port has an individual backpressure enable – IPD_PORT*n*_BP_PAGE_CNT[BP_ENB].

When backpressure is both enabled and active for a port, the hardware attempts to prevent packet arrival on the port. If the port is a PCI port, CN50XX prevents its PCI interface hardware from fetching packet data for the port. If the port is a RGMII port, CN50XX can send pause frames to prevent packets from arriving on the port.

In addition to or alternative to per-port backpressure, the PIP/IPD unit can also drop packets whenever the counter for the port (i.e. IPD_PORT_BP_COUNTERS_PAIR*n*) exceeds the threshold for the port. (IPD_PORT*n*_BP_PAGE_CNT[PAGE_CNT] is also the counter threshold for this packet drop mechanism.) Though the per-port backpressure and this per-port drop mechanism share a counter and a threshold, the per-port drop mechanism can be enabled independently of per-port backpressure. A 0→1 transition on IPD_CTL_STATUS[PBP_EN] is required to enable per-port packet drop, as it is for per-port backpressure, but there is a separate per-port enable for per-port packet drop (IPD_BP_PRT_RED_END[PRT_ENB*n*]).

Note that when PIP_PRT_CFG*n*[RAWDRP] is clear, RAWFULL and RAWSCH packets are never dropped by this per-port drop mechanism, but that when PIP_PRT_CFG*n*[RAWDRP] is set, RAWFULL and RAWSCH packets are treated as other packets.

For the per-port backpressure and/or per-port packet drop mechanisms (that use IPD_PORT_BP_COUNTERS_PAIR*n* and the IPD_PORT*n*_BP_PAGE_CNT[PAGE_CNT] threshold) to work correctly when enabled, the core software must eventually decrement IPD_PORT_BP_COUNTERS_PAIR*n* after a packet and/or its buffers should no longer be assigned to the port, and the port is free to receive more packets. When core software writes the IPD_SUB_PORT_BP_PAGE_CNT register with the appropriate values (2's complement of the amount to decrement), PIP/IPD decrements IPD_PORT_BP_COUNTERS_PAIR*n* for a port. The simplest scheme would be to decrement the counter for a port once the core software first gets the work that PIP/IPD creates for the packet, but this is not required and many other options are possible. Each decrement can be millions, so core software is free to decrement the port back pressure counter for many input packets using only a single CSR write.

The final mechanism that CN50XX provides to keep from exhausting the pool 0 buffers is IPD's per-QOS admission control. If desired, when the FPA pool 0 buffers get low (i.e. when IPD_QUE0_FREE_PAGE_CNT and/or FPA_QUE(0..7)_AVAILABLE get low), PIP/IPD can drop packets. The different QOS levels can be configured with different thresholds on the FPA pool 0 buffer count - packets with some QOS values can drop sooner than other packets with different QOS values. PIP/IPD hardware does not allocate buffers for, nor writes any data into, buffers for packets that it chooses to drop by either the per-QOS admission control or by the per-port packet drop mechanism. The details of the per-QOS admission control algorithm are described in the next section.

7.7 PIP/IPD Per-QOS Admission Control

As Section 7.5 describes, PIP/IPD uses input port, configuration variables, and packet parsing to select one of eight QOS levels. It then tags the work-queue entry for the packet with the QOS level (assuming the packet is not dropped) before sending the work to POW. PIP/IPD normally uses the exact same calculation to produce a QOS value to differentiate and prioritize packets for admission control, but the admission control QOS can be different than the value specified for the work-queue entry when the packet has one of the following:

- an L2/L1 receive error (WORD2[RE] = 1)
- an IP error (WORD2[IE] = 1)
- a TCP/UDP error (WORD2[LE] = 1)

The two QOS calculations are slightly different since PIP/IPD performs the admission-control check before the entire packet is received, and checksums and other errors cannot be checked before the hardware completes the admission control calculations. The admission control QOS logic assumes that these error conditions are not present when it calculates the QOS value.

The following pseudocode details the PIP/IPD admission control QOS selection for a given packet:

```
bool not_present_or_in_last_pip_tick(uint _skip, uint _pkt_len, uint _byte_num) {
  uint pkt_accumulate = ((_skip % 8) ? (8-(_skip % 8)) : 0);
  uint last_tick = (pkt_accumulate + ((_pkt_len % 8) ? (_pkt_len % 8) : 8));
  return (_pkt_len - last_tick) < _byte_num;
}

QOS = PIP_PRT_CFG[port][QOS];
if (there is an L1/L2 receive error and the packet is ≤ 136 bytes) {
  ;
}
else if(ISRAWFULL || ISRAWSCH) {
  QOS = PKT_INST_HDR[QOS];
}
else {
  if(VV && PIP_PRT_CFG[port][QOS_VLAN]) {
    VLAN = (VS && PIP_GBL_CFG[VS_QOS]) ? VLAN1 : VLAN0;
    QOS = PIP_QOS_VLAN[VLAN<15:13>][QOS]; // VLAN<15:13> is priority
  }
  if(Is_IP && ((IP.version==4) || (IP.version==6)) &&
     (IP.version is present in packet)) {
    if(PIP_PRT_CFG[port][QOS_DIFF])) {
      use_diffserv = !not_present_or_in_last_pip_tick(l2_skip, pkt_len,
                  IP.TOS/CLASS<7:2>);
      tos_class = use_diffserv ? IP.TOS/CLASS<7:2> : 0;
      QOS = PIP_QOS_DIFF[tos_class][QOS]; // the 6 msbs of TOS/CLASS }
  }
    for (i=0; i<4; i++) {
      if (PIP_PRT_CFG[port][QOS_WAT<i>] and PIP_QOS_WATCH[i][TYPE]!=0) {
         mask16 = ~PIP_QOS_WATCH[i][MASK] & 0xffff;
         mask8 = ~PIP_QOS_WATCH[i][MASK] & 0xff;
         match16 = PIP_QOS_WATCH[i][MATCH] & mask16;
         match8 = PIP_QOS_WATCH[i][MATCH] & mask8;
         protnh = IP protocol/next header field is present in packet ?
            IP protocol/next header field : 0;
         if (packet is IP malformed and
              not_present_or_in_last_pip_tick(l2_skip, pkt_len, PROT/NH) and
              PIP_QOS_WATCH[i][TYPE] is protocol/next_header) {
            QOS = UNPREDICTABLE;
            break;
         }
         else if(PIP_QOS_WATCH[i][TYPE] is protocol/next_header and
```

```
            match8 equals protnh&mask8) {
          QOS = PIP_QOS_WATCH[i][WATCHER];
          break;
        }
        else if(PIP_QOS_WATCH[i][TYPE] is TCP and protnh is TCP and
            match16 equals (TCP destination port field)&mask16 and
            ((packet is IPv6) or ((IPv4.HL==5) && !IPv4.MF && !IPv4.offset)) and
            TCP destination port field is present in packet) {
          QOS = PIP_QOS_WATCH[i][WATCHER];
          break;
        }
        else if(PIP_QOS_WATCH[i][TYPE] is UDP and protnh is UDP and
            match16 equals (UDP destination port field)&mask16 and
            ((packet is IPv6) or ((IPv4.HL==5) && !IPv4.MF && !IPv4.offset)) and
            UDP destination port field is present in packet) {
          QOS = PIP_QOS_WATCH[i][WATCHER];
          break;
        }
      }
    }
  }
}
```

Each of the eight QOS levels has:

- Independent pass and drop marks for RED-like probabilistic drop scheme
- Independent mixing to calculate a moving average of IPD_QUE0_FREE_PAGE_CNT (i.e. the number of buffers available in FPA pool 0)

Figure 7–12 shows how PIP/IPD calculates the packet-drop probability for a QOS level using the configured values IPD_QOS(0..7)_RED_MARKS[PASS] and IPD_QOS*_RED_MARKS[DROP]. (IPD_RED_QUE(0..7)_PARAM[PRB_CON] must also be set properly for correct operation.) The actual hardware calculates the drop probability for the eight QOS levels only periodically and at a granularity of 1/256. PIP/IPD recalculates the drop probability once every 8×(IPD_RED_PORT_ENABLE[PRB_DLY]+68) clock cycles, using the moving average of IPD_QUE0_FREE_PAGE_CNT for the QOS level at that time. IPD_RED_PORT_ENABLE[PRB_DLY] should typically be set to 0x0.

Upon packet arrival, whenever the current moving average for the QOS level is between the pass and drop marks, PIP/IPD compares the precalculated drop probability to a pseudo-random number to determine whether to drop a packet due to per-QOS admission control. The precalculated drop probability is not used by PIP/IPD whenever the current moving average is below the drop or above the pass mark.

PIP/IPD also recalculates the moving average of IPD_QUE0_FREE_PAGE_CNT (equivalently, FPA_QUE(0..7)_AVAILABLE) once every 8 × (IPD_RED_PORT_ENABLE[PRB_DLY]+10) cycles:

$$avg = \left((avg \times IPD\_RED\_QUE*\_PARAM[AVG\_CON]) + (IPD\_QUE0\_FREE\_PAGE\_CNT \times IPD\_RED\_QUE*\_PARAM[NEW\_CON]) \right) / 256$$

where:

avg = moving average of IPD_QUE0_FREE_PAGE_CNT for the QOS level

IPD_RED_QUE*_PARAM[AVG_CON] = AVG_CON config for the QOS level

IPD_RED_QUE*_PARAM[NEW_CON] = NEW_CON config for the QOS level

IPD_QUE0_FREE_PAGE_CNT = number of buffers in FPA pool 0.

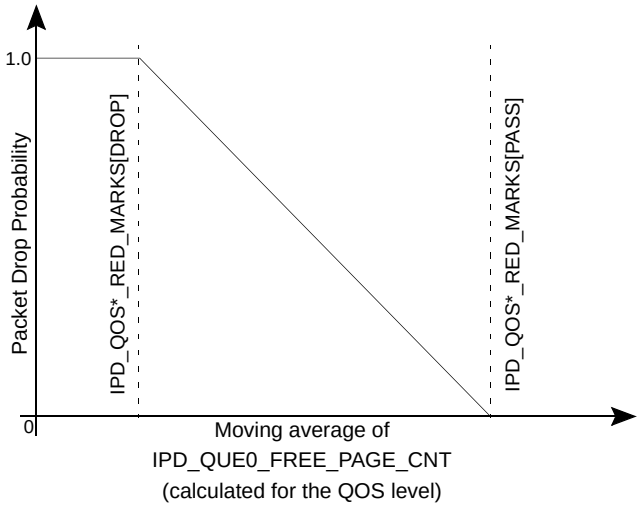


Figure 7–12 PIP/IPD Packet-Drop Probability

The moving average for a QOS level tracks changes to IPD_QUE0_FREE_PAGE_CNT more closely when AVG_CON for the QOS level is smaller, and less closely when AVG_CON is larger. (AVG_CON and NEW_CON must total to 256.) Note that the fractional part of **avg** is ignored, and this can lead to an inherent error of as much as 256/(256-AVG_CON) in the **avg** calculation.

Note that there are two methods to make the moving average track the actual average less closely:

- Larger IPD_RED_PORT_ENABLE[AVG_DLY] values cause the moving averages of ALL QOS levels to track less closely
- Larger AVG_CON values cause the moving averages of individual QOS levels to track less closely

The **avg** equation above assumes that IPD_RED_QUE(0..7)_PARAM[USE_PCNT] = 0.

If IPD_RED_QUE*_PARAM[USE_PCNT] = 1, **avg** for the selected QOS level is simply a periodic sample of IPD_QUE0_FREE_PAGE_CNT (assuming AVG_CON and NEW_CON total to 256).

RAWFULL and RAWSCH packets are never dropped due to per-QOS admission control when PIP_PRT_CFG*n*[RAWDRP] is clear. Otherwise, admission control is enabled on an individual port basis via IPD_RED_PORT_ENABLE[PRT_ENB<*n*>].

7.8 PIP Registers

All the PIP registers are 64 bits long (eight bytes) and are shown in Table 7–6.

Table 7–6 PIP Registers

Register	Address	CSR Type[1]	Detailed Description
PIP_BIST_STATUS	0x00011800A0000000	RSL	See page 307
PIP_INT_REG	0x00011800A0000008	RSL	See page 308
PIP_INT_EN	0x00011800A0000010	RSL	See page 309
PIP_STAT_CTL	0x00011800A0000018	RSL	See page 309
PIP_GBL_CTL	0x00011800A0000020	RSL	See page 310
PIP_GBL_CFG	0x00011800A0000028	RSL	See page 311
PIP_SFT_RST	0x00011800A0000030	RSL	See page 312
PIP_IP_OFFSET	0x00011800A0000060	RSL	See page 313
PIP_TAG_SECRET	0x00011800A0000068	RSL	See page 313
PIP_TAG_MASK	0x00011800A0000070	RSL	See page 314
PIP_TODO_ENTRY	0x00011800A0000078	RSL	See page 314
PIP_DEC_IPSEC0 ... PIP_DEC_IPSEC3	0x00011800A0000080 ... 0x00011800A0000098	RSL	See page 314
PIP_RAW_WORD	0x00011800A00000B0	RSL	See page 314
PIP_QOS_VLAN0 ... PIP_QOS_VLAN7	0x00011800A00000C0 ... 0x00011800A00000F8	RSL	See page 315
PIP_QOS_WATCH0 ... PIP_QOS_WATCH7	0x00011800A0000100 ... 0x00011800A0000138	RSL	See page 315
PIP_PRT_CFG0 ... PIP_PRT_CFG2	0x00011800A0000200 ... 0x00011800A0000210	RSL	See page 316
PIP_PRT_CFG32 PIP_PRT_CFG33	0x00011800A0000300 0x00011800A0000308	RSL	See page 316
PIP_PRT_TAG0 ... PIP_PRT_TAG2	0x00011800A0000400 ... 0x00011800A0000410	RSL	See page 317
PIP_PRT_TAG32 PIP_PRT_TAG33	0x00011800A0000500 0x00011800A0000508	RSL	See page 317
PIP_QOS_DIFF0 ... PIP_QOS_DIFF63	0x00011800A0000600 ... 0x00011800A00007F8	RSL	See page 318
PIP_STAT0_PRT0 PIP_STAT0_PRT1 PIP_STAT0_PRT2	0x00011800A0000800 0x00011800A0000850 0x00011800A00008A0	RSL	See page 319
PIP_STAT0_PRT32 PIP_STAT0_PRT33	0x00011800A0001200 0x00011800A0001250	RSL	See page 319
PIP_STAT1_PRT0 PIP_STAT1_PRT1 PIP_STAT1_PRT2	0x00011800A0000808 0x00011800A0000858 0x00011800A00008A8	RSL	See page 319
PIP_STAT1_PRT32 PIP_STAT1_PRT33	0x00011800A0001208 0x00011800A0001258	RSL	See page 319
PIP_STAT2_PRT0 PIP_STAT2_PRT1 PIP_STAT2_PRT2	0x00011800A0000810 0x00011800A0000860 0x00011800A00008B0	RSL	See page 319
PIP_STAT2_PRT32 PIP_STAT2_PRT33	0x00011800A0001210 0x00011800A0001260	RSL	See page 319
PIP_STAT3_PRT0 PIP_STAT3_PRT1 PIP_STAT3_PRT2	0x00011800A0000818 0x00011800A0000868 0x00011800A00008B8	RSL	See page 320
PIP_STAT3_PRT32 PIP_STAT3_PRT33	0x00011800A0001218 0x00011800A0001268	RSL	See page 320

Table 7–6 PIP Registers (Continued)

Register	Address	CSR Type[1]	Detailed Description
PIP_STAT4_PRT0 PIP_STAT4_PRT1 PIP_STAT4_PRT2	0x00011800A0000820 0x00011800A0000870 0x00011800A00008C0	RSL	See page 320
PIP_STAT4_PRT32 PIP_STAT4_PRT33	0x00011800A0001220 0x00011800A0001270	RSL	See page 320
PIP_STAT5_PRT0 PIP_STAT5_PRT1 PIP_STAT5_PRT2	0x00011800A0000828 0x00011800A0000878 0x00011800A00008C8	RSL	See page 320
PIP_STAT5_PRT32 PIP_STAT5_PRT33	0x00011800A0001228 0x00011800A0001278	RSL	See page 320
PIP_STAT6_PRT0 PIP_STAT6_PRT1 PIP_STAT6_PRT2	0x00011800A0000830 0x00011800A0000880 0x00011800A00008D0	RSL	See page 320
PIP_STAT6_PRT32 PIP_STAT6_PRT33	0x00011800A0001230 0x00011800A0001280	RSL	See page 320
PIP_STAT7_PRT0 PIP_STAT7_PRT1 PIP_STAT7_PRT2	0x00011800A0000838 0x00011800A0000888 0x00011800A00008D8	RSL	See page 320
PIP_STAT7_PRT32 PIP_STAT7_PRT33	0x00011800A0001238 0x00011800A0001288	RSL	See page 320
PIP_STAT8_PRT0 PIP_STAT8_PRT1 PIP_STAT8_PRT2	0x00011800A0000840 0x00011800A0000890 0x00011800A00008E0	RSL	See page 321
PIP_STAT8_PRT32 PIP_STAT8_PRT33	0x00011800A0001240 0x00011800A0001290	RSL	See page 321
PIP_STAT9_PRT0 PIP_STAT9_PRT1 PIP_STAT9_PRT2	0x00011800A0000848 0x00011800A0000898 0x00011800A00008E8	RSL	See page 321
PIP_STAT9_PRT32 PIP_STAT9_PRT33	0x00011800A0001248 0x00011800A0001298	RSL	See page 321
PIP_TAG_INC0 ... PIP_TAG_INC63	0x00011800A0001800 ... 0x00011800A00019F8	RSL	See page 318
PIP_STAT_INB_PKTS0 PIP_STAT_INB_PKTS1 PIP_STAT_INB_PKTS2	0x00011800A0001A00 0x00011800A0001A20 0x00011800A0001A40	RSL	See page 322
PIP_STAT_INB_PKTS32 PIP_STAT_INB_PKTS33	0x00011800A0001E00 0x00011800A0001E20	RSL	See page 322
PIP_STAT_INB_OCTS0 PIP_STAT_INB_OCTS1 PIP_STAT_INB_OCTS2	0x00011800A0001A08 0x00011800A0001A28 0x00011800A0001A48	RSL	See page 322
PIP_STAT_INB_OCTS32 PIP_STAT_INB_OCTS33	0x00011800A0001E08 0x00011800A0001E28	RSL	See page 322
PIP_STAT_INB_ERRS0 PIP_STAT_INB_ERRS1 PIP_STAT_INB_ERRS2	0x00011800A0001A10 0x00011800A0001A30 0x00011800A0001A50	RSL	See page 322
PIP_STAT_INB_ERRS32 PIP_STAT_INB_ERRS33	0x00011800A0001E10 0x00011800A0001E30	RSL	See page 322

1. RSL-type registers are accessed indirectly across the I/O Bus.

PIP BIST Results Register
PIP_BIST_STATUS

BIST status register. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:18>	—	RAZ	—	—	Reserved
<17:0>	BIST	RO	0x0	0x0	BIST results. Hardware sets a bit in BIST for memory that fails BIST.

PIP Interrupt Register
PIP_INT_REG

Any exception event that occurs is captured in the PIP_INT_REG. PIP_INT_REG will set the exception bit regardless of the value of PIP_INT_EN. PIP_INT_EN only controls if an interrupt is raised to software. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved.
<11>	LENERR	R/W1C	0	0	Frame was received with length error.
<10>	MAXERR	R/W1C	0	0	Frame was received with length > max_length.
<9>	MINERR	R/W1C	0	0	Frame was received with length < min_length.
<8>	BEPERR	R/W1C	0	0	Parity error in back-end memory.
<7>	FEPERR	R/W1C	0	0	Parity error in front-end memory.
<6>	TODOOVR	R/W1C	0	0	To-do list overflow.
<5>	SKPRUNT	R/W1C	0	0	Skip runt packets. Packet was engulfed by skipper.
<4>	BADTAG	R/W1C	0	0	A bad tag was sent from IPD.
<3>	PRTNXA	R/W1C	0	0	Nonexistent port.
<2>	BCKPRS	R/W1C	0	0	PIP asserted backpressure.
<1>	—	RAZ	—	—	Reserved.
<0>	PKTDRP	R/W1C	0	0	Packet dropped due to QOS.

Notes:

- TODOOVR: The PIP to-do list stores packets that have been received and require work-queue-entry generation.
- SKPRUNT: If a [packet size] < [amount programmed in the per-port skippers], there is nothing to parse and the entire packet is skipped over. This is probably not the desired effect, so there is an indication to software.
- BADTAG: A tag is considered bad when it is resumed by a new packet before it was released by PIP. PIP considers a tag released by one of two methods.
  - QOS dropped so that it is released over the **pip_ipd_release** bus.
  - Work-queue entry is validated by the **pip_ipd_done** signal.
- PRTNXA: CN50XX supports ports 0–2, 32, 33. If PIP receives a packet that is not in the range, the address processed is mapped into the valid address space (the mapping is currently unpredictable) and the bit is set.

  For packet ports (0–31), PRTNXA is asserted for packets received on ports 3–31 regardless of mode.

  For upper (PCI) ports (32–63), PRTNXA is asserted for packets received on ports 34–63.
- BCKPRS: PIP can assert backpressure to the receive logic when the to-do list exceeds a high-water mark. When this occurs, PIP can raise an interrupt to software.
- PKTDRP: PIP can drop packets based on QOS results received from IPD. If the QOS algorithm decides to drop a packet, PIP asserts an interrupt.

PIP Interrupt Enable Register
PIP_INT_EN

Determines if hardware should raise an interrupt to software when an exception event occurs.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved
<11>	LENERR	R/W	0	0	Frame was received with length error.
<10>	MAXERR	R/W	0	0	Frame was received with length > max_length.
<9>	MINERR	R/W	0	0	Frame was received with length < min_length.
<8>	BEPERR	R/W	0	0	Parity error in back end memory
<7>	FEPERR	R/W	0	0	Parity error in front end memory
<6>	TODOOVR	R/W	0	0	To-do list overflow
<5>	SKPRUNT	R/W	0	0	Packet was engulfed by skipper
<4>	BADTAG	R/W	0	0	A bad tag was sent from IPD
<3>	PRTNXA	R/W	0	0	Nonexistent port
<2>	BCKPRS	R/W	0	0	PIP asserted backpressure
<1>	—	RAZ	—	—	Reserved
<0>	PKTDRP	R/W	0	0	Packet dropped due to QOS

PIP Stat Control Register
PIP_STAT_CTL

Controls how the PIP statistics counters are handled. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
63:1>	—	RAZ	—	—	Reserved
<0>	RDCLR	R/W	1	1	Stat registers are read and clear: 0 = stat registers hold value when read 1 = stat registers are cleared when read

PIP Global Control Register
PIP_GBL_CTL

Global control information. These are the enable signals global-checker for IPv4/IPv6 and TCP/UDP parsing. The enable signals affect all ports. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:20>	—	RAZ	—	—	Reserved
<19:17>	SPARE	R/W	0x0	0x0	Spare bits.
<16>	IGNRS	R/W	0	0	Ignore RS bit. When set to 1, ignore PKT_INST_HDR[RS]. Applies only to ports 0–31.
<15>	VS_WQE	R/W	0	0	Indicates which VLAN CFI and ID to use when VLAN stacking: 0 = use 1st VLAN (network order), 1 = use 2nd VLAN (network order)
<14>	VS_QOS	R/W	0	0	Indicates which VLAN priority to use when VLAN stacking: 0 = use 1st VLAN (network order), 1 = use 2nd VLAN (network order)
<13>	L2_MAL	R/W	1	1	Enable L2 malformed packet check.
<12>	TCP_FLAG	R/W	1	1	Enable TCP flags checks.
<11>	L4_LEN	R/W	1	1	Enable TCP/UDP length check.
<10>	L4_CHK	R/W	1	1	Enable TCP/UDP checksum check.
<9>	L4_PRT	R/W	1	1	Enable TCP/UDP illegal port check.
<8>	L4_MAL	R/W	1	1	Enable TCP/UDP malformed packet check.
<7:6>	—	RAZ	—	—	Reserved.
<5:4>	IP6_EEXT	R/W	0x1	0x3	Enable IPv6 early extension headers.
<3>	IP4_OPTS	R/W	1	1	Enable IPv4 options check.
<2>	IP_HOP	R/W	1	1	Enable TTL (IPv4) / hop (IPv6) check.
<1>	IP_MAL	R/W	1	1	Enable malformed check.
<0>	IP_CHK	R/W	1	1	Enable IPv4 header checksum check.

The following text describes the conditions in which each checker asserts and flags an exception. When the checker is disabled, the exception is not flagged and the packet is parsed as best it can be parsed.

NOTE: By disabling conditions, packets can be parsed incorrectly (.i.e. IP_MAL and L4_MAL could cause bits to be seen in the wrong place. IP_CHK and L4_CHK mean that the packet was corrupted).

- TCP_FLAG: Indicates any of the following conditions:
  - {URG, ACK, PSH, RST, SYN, FIN}: tcp_flag
  - 6'b000001 = (FIN only)
  - 6'b000000 = (0)
  - 6'bxxx1x1 = (RST+FIN+*)
  - 6'b1xxx1x = (URG+SYN+*)
  - 6'bxxx11x = (RST+SYN+*)
  - 6'bxxxx11 = (SYN+FIN+*)
- L4_LEN: Indicates that the TCP or UDP length does not match the IP length.
- L4_CHK: Indicates that a packet classified as either TCP or UDP contains an L4 checksum failure.
- L4_PRT: Indicates that a TCP or UDP packet has an illegal port number – either the source or destination port is 0.
- L4_MAL: Indicates that a TCP or UDP packet is not long enough to cover the TCP or UDP header.

- IP6_EEXT: Indicates the presence of IPv6 early extension headers. These bits only apply to packets classified as IPv6. Bit 0 flags early extensions when next_header is any one of the following:
  - Hop-by-hop (0)
  - Destination (60)
  - Routing (43)

  Bit 1 flags early extentions when next_header is not any of the following:
  - TCP (6)
  - UDP (17)
  - Fragmentation (44)
  - ICMP (58)
  - IPSEC ESP (50)
  - IPSEC AH (51)
  - IPCOMP
- IP4_OPTS: Indicates the presence of IPv4 options. It is set when the length ≠ 5. This only applies to packets classified as IPv4.
- IP_HOP: Indicates that the IPv4 TTL field or IPv6 HOP field is 0.
- IP_MAL: Indicates that the packet was malformed. Malformed packets are defined as packets that are not long enough to cover the IP header or not long enough to cover the length in the IP header.
- IP_CHK: Indicates that an IPv4 packet contained an IPv4 header checksum violations. Only applies to packets classified as IPv4.

PIP Global Config Register
PIP_GBL_CFG

Global configuration information that applies to all ports. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:19>	—	RAZ	—	—	Reserved.
<18>	TAG_SYN	R/W	0	0	Do not include src_crc for TCP/SYN&!ACK packets 0 = include src_crc 1 = tag hash is dst_crc for TCP/SYN&!ACK packets
<17>	IP6_UDP	R/W	1	1	IPv6/UDP checksum is not optional 0 = Allow optional checksum code 1 = Do not allow optional checksum code
<16>	MAX_L2	R/W	0	0	Config bit to choose the largest L2 frame size Chooses the value of the L2 Type/Length field to classify the frame as length. 0 = 1500 / 0x5dc 1 = 1535 / 0x5ff
<15:11>	—	RAZ	—	—	Reserved.
<10:8>	RAW_SHF	R/W	0x0	0x0	RAW Packet shift amount Number of bytes to pad a packet.
<7:3>	—	RAZ	—	—	Reserved.
<2:0>	NIP_SHF	R/W	0x0	0x0	Non-IP shift amount. Number of bytes to pad a packet that has been classified as not IP.

Note:

- IP6_UDP: IPv4 allows an optional UDP checksum by sending the all-0s patterns. IPv6 outlaws this and the spec says to always check UDP checksum. This mode bit allows the user to treat IPv6 as IPv4, meaning that the all-0s pattern will cause a UDP checksum pass.

PIP Soft Reset Register
PIP_SFT_RST

Allows soft reset. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	RST	R/W	0	0	Reset. When set to 1 by software, PIP gets a short reset pulse (three cycles in duration).

This bit resets much of PIP's internal state. The following CSRs are not reset:

- PIP_BIST_STATUS
- PIP_STAT*n*_PRT*m*
- PIP_STAT_INB_PKTS*n*
- PIP_STAT_INB_OCTS*n*
- PIP_STAT_INB_ERRS*n*
- PIP_TAG_INC*n*

PIP IP Offset Into the Work-Queue-Entry Register
PIP_IP_OFFSET

Specifies the eight-byte offset to find the start of the IP header in the data portion of IP work-queue entries. OFFSET is restricted in that the entire IP and TCP/UDP header must be buffered by hardware. In general, OFFSET must be set in the 0–4 range. If the system restricts all IPv6 packets, the full range of 0–7 can be used if desired. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved
<2:0>	OFFSET	R/W	0x0	0x0	Byte offset. The number of eight-byte ticks to include in the work-queue entry prior to IP data. 0 = 0 bytes (IP data starts at WORD4 of work-queue entry) 1 = 8 bytes (IP data starts at WORD5 of work-queue entry) 2 = 16 bytes (IP data starts at WORD6 of work-queue entry) 3 = 24 bytes (IP data starts at WORD7 of work-queue entry) 4 = 32 bytes (IP data starts at WORD8 of work-queue entry) 5 = 40 bytes (IP data starts at WORD9 of work-queue entry) 6 = 48 bytes (IP data starts at WORD10 of work-queue entry) 7 = 56 bytes (IP data starts at WORD11 of work-queue entry) In normal configurations, OFFSET must be set in the 0–4 range to allow the entire IP and TCP/UDP headers to be buffered in hardware and calculate the L4 checksum for TCP/UDP packets. The maximum value of OFFSET is determined by the types of packets that can be sent to PIP as follows: **Packet Type — MAX OFFSET** IPv4/TCP/UDP — 7 IPv6/TCP/UDP — 5 IPv6/without L4 parsing — 6 If the L4 can be ignored, the maximum value of OFFSET for IPv6 packets can increase to 6. Here are the following programming restrictions for IPv6 packets and OFFSET = 6: PIP_GBL_CTL[TCP_FLAG] = 0 PIP_GBL_CTL[L4_LEN] = 0 PIP_GBL_CTL[L4_CHK] = 0 PIP_GBL_CTL[L4_PRT] = 0 PIP_GBL_CTL[L4_MAL] = 0 PIP_DEC_IPSEC[TCP] = 0 PIP_DEC_IPSEC[UDP] = 0 PIP_PRT_TAG[IP6_DPRT] = 0 PIP_PRT_TAG[IP6_SPRT] = 0 PIP_GBL_CTL[TCP6_TAG] = 0 PIP_PRT_TAG[TAG_SYN] = 0

PIP Initial-Value Register
PIP_TAG_SECRET

The source and destination initial values (IVs) in tag generation provide a mechanism for each CN50XX to be unique. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:16>	DST	R/W	0x0	0x0	Secret for the destination tuple tag CRC calculation.
<15:0>	SRC	R/W	0x0	0x0	Secret for the source tuple tag CRC calculation.

PIP Mask-Bit Register
PIP_TAG_MASK

Provides the mask-bit field in tag generation. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved.
<15:0>	MASK	R/W	0x0	0x0	When set, clears the individual bits of the lower 16 bits of the computed tag. Does not affect RAW or INST_HDR packets.

PIP To-Do List Entry Register
PIP_TODO_ENTRY

Head entry of the to-do list (debug only). Summary of the current packet that has completed and is waiting to be processed. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	VAL	RO	—	—	Entry is valid.
<62>	—	RAZ	—	—	Reserved.
<61:0>	ENTRY	RO	—	—	To-do list entry summary.

PIP UDP/TCP Ports to Watch for DEC IPSEC Registers
PIP_DEC_IPSEC(0..3)

PIP sets the dec_ipsec based on TCP or UDP destination port. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:18>	—	RAZ	—	—	Reserved
<17>	TCP	R/W	0	0	This DPRT should be used for TCP packets.
<16>	UDP	R/W	0	0	This DPRT should be used for UDP packets.
<15:0>	DPRT	R/W	0x0	0x0	UDP or TCP destination port to match on.

PIP RAW Word2 of the Work-Queue-Entry Register
PIP_RAW_WORD

The RAW Word2 to be inserted into the work-queue entry of RAW packets. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:56>	—	RAZ	—	—	Reserved.
<55:0>	WORD	R/W	0x0	0x0	Word2 of the workQ entry. The 8-bit bufs field is still set by hardware (IPD)

PIP QOS VLAN Tables Registers
PIP_QOS_VLAN(0..7)

If the PIP indentifies a packet to be VLAN tagged, the QOS can be set based on the VLAN user priority. These eight registers comprise the QOS values for all VLAN user priority values. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved
<2:0>	QOS	R/W	0x0	0x0	VLAN QOS value.

PIP QOS Watcher-Table Registers
PIP_QOS_WATCH(0..7)

Sets up the configuration CSRs for the four QOS watchers. Each watcher can be set to look for a specific protocol, Ethertype, or TCP/UDP destination port to override the default QOS value. Note that the watchers have overlapping criteria. See Table 7–6 for address

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved
<47:32>	MASK	R/W	0x0	0x0	Mask off a range of values.
<31:28>	—	RAZ	—	—	Reserved
<27:24>	GRP	R/W	0x0	0x0	The group number of the watcher.
<23>	—	RAZ	—	—	Reserved
<22:20>	WATCHER	R/W	0x0	0x0	The QOS level of the watcher.
<19>	—	RAZ	—	—	Reserved
<18:16>	TYPE	R/W	0x0	0x0	The field for the watcher match against: 0x0 = disable across all ports 0x1 = protocol (IPv4) or next_header (IPv6) 0x2 = TCP destination port 0x3 = UDP destination port 0x4 = Ethertype 0x5–7 = Reserved
<15:0>	MATCH	R/W	0x0	0x0	The value to watch for.

PIP Frame Length Check Registers
PIP_FRM_LEN_CHK0/1

PIP_FRM_LEN_CHK0 is used for packets on packet interface0, and PCI, and PKO loopback ports. PIP_FRM_LEN_CHK1 is used for PCI RAW packets.

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
63:32>	—	RAZ	—	—	Reserved
<31:16>	MAXLEN	R/W	0x600	0x60	Byte count for max-sized frame check. Failing packets set the MAXERR interrupt and are optionally sent with opcode==MAXERR. The effective MAXLEN used by hardware is PIP_FRM_LEN_CHK[MAXLEN] + 4 × VV + 4 × VS
<15:0>	MINLEN	R/W	0x40	0x40	Byte count for min-sized frame check. Failing packets set the MINERR interrupt and are optionally sent with opcode==MINERR.

PIP Per-Port Configuration Registers
PIP_PRT_CFG(0..2, 32/33)

Contains per-port configuration information. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:53>	—	RAZ	—	—	Reserved.
<52>	PAD_LEN	R/W	0	0	When set, disables the length check for packets with padding in the client data.
<51>	VLAN_LEN	R/W	0	0	When set, disables the length check for VLAN packets.
<50>	LENERR_EN	R/W	0	1	L2 length error check enable. Frame was received with length error.
<49>	MAXERR_EN	R/W	0	1	Max frame error check enable. Frame was received with length > max_length.
<48>	MINERR_EN	R/W	0	1	Min frame error check enable. Frame was received with length < min_length.
<47:44>	GRP_WAT_47	R/W	0x0	0x0	GRP Watcher enable. Provides an enable bit for each watcher for each of four ports. When set to 1, GRP number in PIP_QOS_WATCH*n*[GRP] is used when the GRP watcher matches for the packet. (Watchers 4–7)
<43:40>	QOS_WAT_47	R/W	0x0	0x0	QOS Watcher enable. Provides an enable bit for each of four watchers for each port. When set to 1, QOS level in PIP_QOS_WATCH*n*[WATCHER] is used when the QOS watcher matches for the packet.(Watchers 4–7)
<39:37>	—	RAZ	—	—	Reserved.
<36>	RAWDRP	R/W	0	0	Allow RAW packet drop. 0 = never drop packets that PIP indicates are RAW 1 = allow the IPD to drop RAW packets based on RED algorithm
<35:34>	TAG_INC	R/W	0x0	0x0	Tag include. Specifies which of the four PIP_TAG_INC*n* registers to use when calculating the mask tag hash 0 = registers 0–15 2 =registers 32–47 1 = registers 16–31 3 = registers 48–63
<33>	DYN_RS	R/W	0	0	Dynamic RS. Dynamically calculate RS based on packet size.
<32>	INST_HDR	R/W	0	0	INST header. When set, the eight-byte INST_HDR is present on all packets (except PCI ports 32–35).
<31:28>	GRP_WAT	R/W	0x0	0x0	GRP watcher enable. Provides an enable bit for each watcher for each of four ports. When set to 1, GRP number in PIP_QOS_WATCH*n*[GRP] is used when the GRP watcher matches for the packet. (Watchers 0–3)
<27>	—	RAZ	—	—	Reserved.
<26:24>	QOS	R/W	0x0	0x0	Default QOS level of the port
<23:20>	QOS_WAT	R/W	0x0	0x0	QOS watcher enable. Provides an enable bit for each of four watchers for each port. When set to 1, QOS level in PIP_QOS_WATCH*n*[WATCHER] is used when the QOS watcher matches for the packet. (Watchers 0–3)
<19>	—	RAZ	—	—	Reserved.
<18>	QOS_VOD	R/W	0	0	QOS VLAN over Diffserv. If VLAN exists, it is used, else, if IP exists, Diffserv is used, else, the per-port default is used. Watchers are still highest priority.
<17>	QOS_DIFF	R/W	0	0	QOS Diffserv
<16>	QOS_VLAN	R/W	0	0	QOS VLAN
<15:10>	—	RAZ	—	—	Reserved.
<9:8>	MODE	R/W	0x0	0x0	Parse mode 0 = no packet inspection 2 = IP parsing / skip-to-L3 1 = L2 parsing/skip-to-L2 3 = Illegal
<7>	—	RAZ	—	—	Reserved.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<6:0>	SKIP	R/W	0x0	0x0	Optional SKIP I amount for packets. Does not apply to packets on PCI ports when a PKT_INST_HDR is present. See Section 7.2.8, Legal SKIP Values for further details.

PIP Per-Port Tag Configuration Registers
PIP_PRT_TAG(0..2, 32/33)

Contains per-port tag configuration information. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:40>	—	RAZ	—	—	Reserved
<39:36>	GRPTAGBASE	R/W	0x0	0x0	When GRPTAG = 1, specifies the offset to use to compute the work-queue entry group from tag bits.
<35:32>	GRPTAGMASK	R/W	0x0	0x0	When GRPTAG = 1, specifies which of the least-significant bits of the work-queue entry Tag field to exclude from the computation.
<31>	GRPTAG	R/W	0	0	Use work-queue entry tag. Enables the use of the least-significant bits of the work-queue entry Tag field to determine the work-queue entry group. group = (WORD2[Tag<3:0>] AND $\overline{\text{GRPTAGMASK}}$) + GRPTAGBASE For more information on the work-queue entry, refer to 7.5 on page 284.
<30>	GRPTAG_MSKIP	R/W	0	0	When set, GRPTAG is used regardless if the packet IS_IP.
<29:28>	TAG_MODE	R/W	0x0	0x0	Specifies the tag algorithm to use 0 = always use tuple tag algorithm 2 = use tuple if IP, else use mask 1 = always use mask tag algorithm 3 = tuple XOR mask
<27:26>	INC_VS	R/W	0x0	0x0	Specifies the VLAN ID (VID) to be included in the tuple tag generation when VLAN stacking is detected 0 = do not include VID 2 = include VID (VLAN1) in hash 1 = include VID (VLAN0) in hash 3 = include VID ({VLAN0,VLAN1)} in hash
<25>	INC_VLAN	R/W	0	0	Specifies whether the VID is included in the tuple tag generation when VLAN stacking is not detected: 0 = do not include VID, 1 = include VID in hash
<24>	INC_PRT	R/W	0	0	Indicates whether the port is included in tag.
<23>	IP6_DPRT	R/W	0	0	Indicates whether the TCP/UDP dst port is included in tuple tag for IPv6 packets.
<22>	IP4_DPRT	R/W	0	0	Indicates whether the TCP/UDP dst port is included in tuple tag for IPv4 packets.
<21>	IP6_SPRT	R/W	0	0	Indicates whether the TCP/UDP src port is included in tuple tag for IPv6 packets
<20>	IP4_SPRT	R/W	0	0	Indicates whether the TCP/UDP src port is included in tuple tag for IPv4 packets.
<19>	IP6_NXTH	R/W	0	0	Indicates whether IPv6 includes next header in tuple tag hash
<18>	IP4_PCTL	R/W	0	0	Indicates whether IPv4 includes protocol in tuple tag hash
<17>	IP6_DST	R/W	0	0	Indicates whether IPv6 includes dst address in tuple tag hash
<16>	IP4_DST	R/W	0	0	Indicates whether IPv4 includes dst address in tuple tag hash
<15>	IP6_SRC	R/W	0	0	Indicates whether IPv6 includes src address in tuple tag hash
<14>	IP4_SRC	R/W	0	0	Indicates whether IPv4 includes src address in tuple tag hash

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<13:12>	TCP6_TAG	R/W	0x0	0x0	Sets the tag_type of a TCP packet (IPv6) 0 = ordered tags 2 = null tags 1 = atomic tags 3 = reserved
<11:10>	TCP4_TAG	R/W	0x0	0x0	Sets the tag_type of a TCP packet (IPv4) 0 = ordered tags 2 = null tags 1 = atomic tags 3 = reserved
<9:8>	IP6_TAG	R/W	0x0	0x0	Sets whether IPv6 packet tag type 0 = ordered tags 2 = null tags 1 = atomic tags 3 = reserved
<7:6>	IP4_TAG	R/W	0x0	0x0	Sets whether IPv4 packet tag type 0 = ordered tags 2 = null tags 1 = atomic tags 3 = reserved
<5:4>	NON_TAG	R/W	0x0	0x0	Sets whether non-IP packet tag type 0 = ordered tags 2 = null tags 1 = atomic tags 3 = reserved
<3:0>	GRP	R/W	0x0	0x0	Core group number. Specifies the core group (0-15) to use for the work.

PIP QOS Diffserv Tables Registers

PIP_QOS_DIFF(0..63)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved
<2:0>	QOS	R/W	0x0	0x0	Diffserv QOS level. Specifies the Diffserv QOS level.

PIP Include Registers

PIP_TAG_INC(0..63)

Specifies which bytes to include in the new tag hash algorithm. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved.
<7:0>	EN	R/W	0x0	0x0	Indicates which bytes to include in the mask tag algorithm. It is broken into four 16-entry masks to cover 128 bytes. The PIP_PRT_CFG*n*[TAG_INC] field selects which of the four: if PIP_PRT_CFG[TAG_INC] = 00, use registers 0–15 if PIP_PRT_CFG[TAG_INC] = 01, use registers 16–31 if PIP_PRT_CFG[TAG_INC] = 10, use registers 32–47 if PIP_PRT_CFG[TAG_INC] = 11, use registers 48–63 Bit [7] corresponds to the MSB and bit [0] corresponds to the LSB of the 8-byte word.

7.8.1 PIP Statistics Counters

These counters work with RGMII ports: 0–2/PCI ports: 32–33.

Special Statistics Counter Behavior

1. Read and write operations must arbitrate for the statistics resources along with the packet engines that are incrementing the counters. In order to not drop packet information, the packet hardware is always a higher priority and the CSR requests are only satisfied when there are idle cycles. This can potentially cause long delays if the system becomes full.
2. Stat counters can be cleared in two ways.
   a. If PIP_STAT_CTL[RDCLR] is set, all read accesses clear the register.
   b. Any write to a stats register also resets the register to zero.

   Note that the clearing operations must obey rule #1 above.
3. All counters are wrapping. Software must ensure they are read periodically.

PIP Port Status 0 Registers
PIP_STAT0_PRT(0..2, 32/33)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	DRP_PKTS	R/W	0x0	—	Inbound packets dropped by the IPD QOS widget per port
<31:0>	DRP_OCTS	R/W	0x0	—	Inbound octets dropped by the IPD QOS widget per port

PIP Port Status 1 Registers
PIP_STAT1_PRT(0..2, 32/33)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved
<47:0>	OCTS	R/W	0x0	—	Number of octets received by PIP (good and bad)

PIP Port Status 2 Registers
PIP_STAT2_PRT(0..2, 32/33)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
63:32>	PKTS	R/W	0x0	—	Number of packets processed by PIP.
<31:0>	RAW	R/W	0x0	—	RAWFULL and RAWSCH packets without an L1/L2 error received by PIP per port.

PIP Port Status 3 Registers
PIP_STAT3_PRT(0..2, 32/33)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	BCST	R/W	0x0	—	Number of identified L2 broadcast packets. Does not include multicast packets. Only includes packets whose parse mode is SKIP_TO_L2.
<31:0>	MCST	R/W	0x0	—	Number of identified L2 multicast packets. Does not include broadcast packets. Only includes packets whose parse mode is SKIP_TO_L2.

PIP Port Status 4 Registers
PIP_STAT4_PRT(0..2, 32/33)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	H65to127	R/W	0x0	—	Number of 65-to-127-byte packets.
<31:0>	H64	R/W	0x0	—	Number of 64-byte packets.

PIP Port Status 5 Registers
PIP_STAT5_PRT(0..2, 32/33)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	H256to511	R/W	0x0	—	Number of 256-to-511-byte packets.
<31:0>	H128to255	R/W	0x0	—	Number of 128-to-255-byte packets.

PIP Port Status 6 Registers
PIP_STAT6_PRT(0..2, 32/33)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	H1024to1518	R/W	0x0	—	Number of 1024-to-1518-byte packets
<31:0>	H512to1023	R/W	0x0	—	Number of 512-to-1023-byte packets

PIP Port Status 7 Registers
PIP_STAT7_PRT(0..2, 32/33)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	FCS	R/W	0x0	—	Number of packets with FCS or Align opcode errors
<31:0>	H1519	R/W	0x0	—	Number of 1519-to-max packets

NOTE: FCS is not checked on the PCI ports 32–35.

PIP Port Status 8 Registers
PIP_STAT8_PRT(0..2, 32/33)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	FRAG	R/W	0x0	—	Number of packets with length < minimum and FCS error
<31:0>	UNDERSZ	R/W	0x0	—	Number of packets with length < minimum

NOTE: FCS is not checked on the PCI ports 32–35.

PIP Port Status 9 Registers
PIP_STAT9_PRT(0..2, 32/33)

See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	JABBER	R/W	0x0	—	Number of packets with length > maximum and FCS error
<31:0>	OVERSZ	R/W	0x0	—	Number of packets with length > maximum

NOTE: FCS is not checked on the PCI ports 32–35.

7.8.2 PIP Inbound Statistics Registers

The inbound statistics registers collect all data sent to PIP from all the packet interfaces, providing the raw counts of everything that comes into the block. The counts reflect all error packets and packets dropped by the PKI RED engine.

These counts are intended for system debug, but could convey useful information in production systems.

PIP Statistic Inbound Packets Registers
PIP_STAT_INB_PKTS(0..2, 32/33)

Inbound packets received by PIP per port. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31:0>	PKTS	R/W	0x0	—	Number of packets without errors received by PIP.

PIP Statistic Inbound Octets Registers
PIP_STAT_INB_OCTS(0..2, 32/33)

Inbound octets received by PIP per port. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved
<47:0>	OCTS	R/W	0x0	—	Total number of octets from all packets received by PIP.

PIP Statistic Inbound Error Registers
PIP_STAT_INB_ERRS(0..2, 32/33)

Inbound error packets received by PIP per port. See Table 7–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	ERRS	R/W	0x0	—	Number of packets with errors received by PIP.

7.9 IPD Registers

The IPD registers are shown in See Table 7–7.

Table 7–7 IPD Registers

Register	Address	CSR Type[1]	Detailed Description
IPD_1ST_MBUFF_SKIP	0x00014F0000000000	NCB	See page 324
IPD_NOT_1ST_MBUFF_SKIP	0x00014F0000000008	NCB	See page 324
IPD_PACKET_MBUFF_SIZE	0x00014F0000000010	NCB	See page 324
IPD_CTL_STATUS	0x00014F0000000018	NCB	See page 325
IPD_WQE_FPA_QUEUE	0x00014F0000000020	NCB	See page 326
IPD_PORT0_BP_PAGE_CNT ... IPD_PORT2_BP_PAGE_CNT	0x00014F0000000028 ... 0x00014F0000000038	NCB	See page 326
IPD_PORT32_BP_PAGE_CNT IPD_PORT33_BP_PAGE_CNT	0x00014F0000000128 0x00014F0000000130	NCB	See page 326
IPD_SUB_PORT_BP_PAGE_CNT	0x00014F0000000148	NCB	See page 326
IPD_1st_NEXT_PTR_BACK	0x00014F0000000150	NCB	See page 327
IPD_2nd_NEXT_PTR_BACK	0x00014F0000000158	NCB	See page 327
IPD_INT_ENB	0x00014F0000000160	NCB	See page 327
IPD_INT_SUM	0x00014F0000000168	NCB	See page 328
IPD_SUB_PORT_FCS	0x00014F0000000170	NCB	See page 328
IPD_QOS0_RED_MARKS ... IPD_QOS7_RED_MARKS	0x00014F0000000178 ... 0x00014F00000001B0	NCB	See page 328
IPD_PORT_BP_COUNTERS_PAIR0 ... IPD_PORT_BP_COUNTERS_PAIR2	0x00014F00000001B8 ... 0x00014F00000001C8	NCB	See page 329
IPD_PORT_BP_COUNTERS_PAIR32 IPD_PORT_BP_COUNTERS_PAIR33	0x00014F00000002B8 0x00014F00000002C0	NCB	See page 329
IPD_RED_PORT_ENABLE	0x00014F00000002D8	NCB	See page 329
IPD_RED_QUE0_PARAM ... IPD_RED_QUE7_PARAM	0x00014F00000002E0 ... 0x00014F0000000318	NCB	See page 330
IPD_PTR_COUNT	0x00014F0000000320	NCB	See page 330
IPD_BP_PRT_RED_END	0x00014F0000000328	NCB	See page 331
IPD_QUE0_FREE_PAGE_CNT	0x00014F0000000330	NCB	See page 331
IPD_CLK_COUNT	0x00014F0000000338	NCB	See page 331
IPD_PWP_PTR_FIFO_CTL	0x00014F0000000340	NCB	See page 331
IPD_PRC_HOLD_PTR_FIFO_CTL	0x00014F0000000348	NCB	See page 332
IPD_PRC_PORT_PTR_FIFO_CTL	0x00014F0000000350	NCB	See page 332
IPD_PKT_PTR_VALID	0x00014F0000000358	NCB	See page 332
IPD_WQE_PTR_VALID	0x00014F0000000360	NCB	See page 333
IPD_BIST_STATUS	0x00014F00000007F8	NCB	See page 333

1. NCB-type registers are accessed directly across the I/O Bus.

IPD First Memory-Buffer Word-Skip Size Register
IPD_1ST_MBUFF_SKIP

The number of words that the IPD will skip when writing the first memory buffer (MBUF). See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	—	—	Reserved
<5:0>	SKIP_SZ	R/W	0	0	The number of eight-byte words from the top of the first MBUF that the IPD stores the next pointer. Legal values for this field are 0 to 32, but the SKIP_SZ+18 ≤ IPD_PACKET_MBUFF_SIZE[MB_SIZE].

IPD Not First MBUF Word-Skip Size Register
IPD_NOT_1ST_MBUFF_SKIP

The number of words that the IPD will skip when writing any MBUF that is not the first. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	—	—	Reserved
<5:0>	SKIP_SZ	R/W	0	0	The number of eight-byte words from the top of any MBUF that is not the first MBUF that the IPD writes the next-pointer. Legal values are 0 to IPD_PACKET_MBUFF_SIZE[MB_SIZE] – 16 (to a max of 32).

IPD PACKET MBUF Size In Words Register
IPD_PACKET_MBUFF_SIZE

The number of words in a MBUF used for packet data store. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved
<11:0>	MB_SIZE	R/W	0x20	0x20	The number of eight-byte words in an MBUF. This must be a number in the range of 32 to 2048. This is also the size of the FPA's queue 0 free page.

IPD Control Status Register
IPD_CTL_STATUS

The number of words in a MBUF used for packet data store. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:15>	—	RAZ	—	—	Reserved.
<14>	NO_WPTR	R/W	0	0	When set, the WQE pointers are not used and the WQE is located at the front of the packet.
<13>	PQ_APKT	R/W	0	0	When set, IPD_PORT_QOS_(0..31,128..159,256..319)_CNT is incremented by one for each work queue entry that is sent to POW.
<12>	PQ_NABUF	R/W	0	0	When set, IPD_PORT_QOS_(0..31,128..159,256..319)_CNT is not incremented when IPD allocates a buffer for a packet.
<11>	IPD_FULL	R/W	0	0	IPD full. When this bit is set to 1, the IPD drives the IPD_BUFF_FULL line to the IOB arbiter, telling it to not give grants to I/O-bus devices sending packet data; when it is clear to 0, the IPD acts normally.
<10>	PKT_OFF	R/W	0	0	Packet buffer off. When this bit is set to 1, the IPD does not buffer the received packet data; when it is clear to 0, the IPD works normally, buffering the received packet data.
<9>	LEN_M8	R/W	0	1	Data length minus 8. When this bit is set to 1, 8 is subtracted from the data-length field in the header written to the POW and the top of an MBUFF. The CN50XX generates a length that includes the length of the data plus 8 for the header field. By setting this bit, the 8 for the Instr field is not included in the Length field of the header. **NOTE:** When this bit is set, the CN50XX is compliant with the IPD specification.
<8>	RESET	R/W	0	0	Reset. When set, causes a reset of the IPD, except RSL.
<7>	ADDPKT	R/W	0	0	Additional packet. When set to 1, IPD_PORT_BP_COUNTERS_PAIR*n*[CNT_VAL] is incremented by one for every work-queue entry that is sent to POW.
<6>	NADDBUF	R/W	0	0	When set to 1, IPD_PORT_BP_COUNTERS_PAIR*n*[CNT_VAL] is not incremented when IPD allocates a buffer for a packet on the port.
<5>	PKT_LEND	R/W	0	0	Packet little-endian. Changes PKT to little-endian write operations to L2C.
<4>	WQE_LEND	R/W	0	0	Work-queue entry little-endian. Changes WQE to little-endian write operations to L2C.
<3>	PBP_EN	R/W	0	0	Port backpressure enable. When set, enables the sending of port-level backpressure to the OCTEON Plus input ports. Once enabled, the sending of port-level backpressure can not be disabled by changing the value of this bit. GMX_INF_MODE[EN] must be set to 1 for each packet interface that requires port backpressure prior to setting PBP_EN to 1.
<2:1>	OPC_MODE	R/W	0x0	0x0	Select the style of write to the L2C. 0 = all packet data and next-buffer pointers are written through to memory. 1 = all packet data and next-buffer pointers are written into the cache. 2 = the first aligned cache block holding the packet data and initial next-buffer pointer is written to the L2 cache. All remaining cache blocks are not written to the L2 cache. 3 = the first two aligned cache blocks holding the packet data and initial next-buffer pointer is written to the L2 cache. All remaining cache blocks are not written to the L2 cache.
<0>	IPD_EN	R/W	0	0	IPD enable. When set to 1, enables the operation of the IPD. When clear to 0, the IPD appears to the IOB arbiter to be applying backpressure, which causes the IOB arbiter to not send grants to I/O-bus devices requesting to send packet data to the IPD.

IPD Work-Queue Entry FPA Page Size Register
IPD_WQE_FPA_QUEUE

Specifies from which FPA queue (0-7) to fetch page pointers for work-queue entries. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved
<2:0>	WQE_QUE	R/W	0	0	Specifies the FPA queue from which to fetch page-pointers for work-queue entries.

IPD Port Backpressure Page-Count Registers
IPD_PORT(0..2, 32/33)_BP_PAGE_CNT

Specifies the maximum number of pages the port may use before backpressure (BP) is applied to the port. BP is applied once the maximum is exceeded. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:18>	—	RAZ	—	—	Reserved
<17>	BP_ENB	R/W	0	0	Backpressure enable: 1 = BP will be applied, 0 = BP will not be applied to port.
<16:0>	PAGE_CNT	R/W	0x0	0x0	The maximum number of page pointers assigned to the port, which, when exceeded, will cause BP to be applied to the port. This value is in 256 page-pointer increments, (i.e. 0x0 = 0 page pointers, 0x1 = 256 page pointers, 0x2 = 512 page pointers, etc.)

IPD Subtract Port Backpressure Page-Count Register
IPD_SUB_PORT_BP_PAGE_CNT

Supplies a 2's-complement value that is added to the indicated port's page-count register, with the net result of the page count lowered. The value should be the 2's complement of the value that needs to be subtracted. Users would add 2's complement values to the IPD_PORT*n*_BP_PAGE_CNT register to return MBUFs (i.e. lower the count) to the counter in order to avoid port-level backpressure being applied to the port. Backpressure is applied when the MBUF-used count of a port exceeds the value in the IPD_PORT*n*_BP_PAGE_CNT. See Table 7–7 for address.

NOTE: This register can't be written from the PCI via a window write.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:31>	—	RAZ	—	—	Reserved
<30:25>	PORT	R/W	0x0	0x0	The port whose page-count register receives the PAGE_CNT field.
<24:0>	PAGE_CNT	R/W	0x0	0x0	The number of page pointers to add to the port counter pointed to by the PORT field.

IPD First Next Pointer Back Values Register
IPD_1ST_NEXT_PTR_BACK

Contains the BACK field used to create the next pointer header for the first MBUF. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	—	—	Reserved
<3:0>	BACK	R/W	0x0	0	Used to find head of buffer from the next pointer header.

IPD Second Next Pointer Back Value Register
IPD_2ND_NEXT_PTR_BACK

Contains the BACK field for use in creating the next pointer header for the first MBUF. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	—	—	Reserved
<3:0>	BACK	R/W	0x0	0	Used to find head of buffer from the next pointer header.

IPD Interrupt Enable Register
IPD_INT_ENB

Used to enable the various interrupting conditions of IPD. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved
<11>	PQ_SUB	R/W	0	0	Enables an interrupt when a port-QOS does a subtract to the count that causes the counter to wrap.
<10>	PQ_ADD	R/W	0	0	Enables an interrupt when a port-QOS does an add to the count that causes the counter to wrap.
<9>	BC_OVR	R/W	0	0	Enables interrupt when the byte-count to the IOB overflows.
<8>	D_COLL	R/W	0	0	Enables interrupt when the packet/WQE data to IOB collides.
<7>	C_COLL	R/W	0	0	Enables interrupt when the packet/WQE commands to IOB collide.
<6>	CC_OVR	R/W	0	0	Enables interrupt when the command credits to the IOB overflow.
<5>	DC_OVR	R/W	0	0	Enables interrupt when the data credits to the IOB overflow.
<4>	BP_SUB	R/W	0	0	Enables interrupts when a backpressure subtract has an illegal value.
<3>	PRC_PAR3	R/W	0	0	Enables parity error interrupts for bits [127:96] of the PBM memory.
<2>	PRC_PAR2	R/W	0	0	Enables parity error interrupts for bits [95:64] of the PBM memory.
<1>	PRC_PAR1	R/W	0	0	Enables parity error interrupts for bits [63:32] of the PBM memory.
<0>	PRC_PAR0	R/W	0	0	Enables parity error interrupts for bits [31:0] of the PBM memory.

IPD Interrupt Summary Register
IPD_INT_SUM

Fields are set when an interrupt condition occurs, write 1 to clear. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved
<11>	PQ_SUB	R/W1C	0	0	Set when a port-QOS does a subtract to the count that causes the counter to wrap.
<10>	PQ_ADD	R/W1C	0	0	Set when a port-QOS does an add to the count that causes the counter to wrap.
<9>	BC_OVR	R/W1C	0	0	Set when the byte-count to the IOB overflows.
<8>	D_COLL	R/W1C	0	0	Set when the packet/WQE data to IOB collides.
<7>	C_COLL	R/W1C	0	0	Set when the packet/WQE commands to IOB collide.
<6>	CC_OVR	R/W1C	0	0	Set when the command credits to the IOB overflow.
<5>	DC_OVR	R/W1C	0	0	Set when the data credits to the IOB overflow.
<4>	BP_SUB	R/W1C	0	0	Set when a backpressure subtract is done with a supplied illegal value.
<3>	PRC_PAR3	R/W1C	0	0	Set when a parity error is detected for bits [127:96] of the PBM memory.
<2>	PRC_PAR2	R/W1C	0	0	Set when a parity error is detected for bits [95:64] of the PBM memory.
<1>	PRC_PAR1	R/W1C	0	0	Set when a parity error is detected for bits [63:32] of the PBM memory.
<0>	PRC_PAR0	R/W1C	0	0	Set when a parity error is detected for bits [31:0] of the PBM memory.

IPD Subtract Ports FCS Register
IPD_SUB_PORT_FCS

When a bit is set, the port corresponding to the set bit position subtracts four bytes from the end of the packet. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:40>	—	RAZ	—	—	Reserved
<39:36>	PORT_BIT2	R/W	0xF	0xF	When a bit is set, the port corresponding to the bit position set subtracts the FCS for packets on that port.
<35:32>	—	RAZ	—	—	Reserved
<31:0>	PORT_BIT	R/W	0xFFFFFFFF	0xFFFFFFFF	When a bit is set, the port corresponding to the bit position set subtracts the FCS for packets on that port.

IPD QOS (0..7) Marks Red High/Low Registers
IPD_QOS(0..7)_RED_MARKS

Set the pass-drop marks for QOS level. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	DROP	R/W	0x0	—	Packet will be dropped if the average queue size is equal to or less than this value.
<31:0>	PASS	R/W	0x0	0	Packet will be passed if the average queue size is larger than this value.

MBUF Counters Port Registers
IPD_PORT_BP_COUNTERS_PAIR(0..2, 32/33)

Ports used to generate backpressure per-port. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:25>	—	RAZ	—	—	Reserved
<24:0>	CNT_VAL	RO	0x0	—	Number of MBUFs being used by data on this port.

IPD RED Port Enable Register
IPD_RED_PORT_ENABLE

Set the pass-drop marks for QOS level. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:50>	PRB_DLY	R/W	0x0	0x0	Pass-drop probability delay. The number of core-clock cycles to wait ([PRB_DLY + 68] × 8) before calculating the new packet-drop probability for each QOS level.
<49:36>	AVG_DLY	R/W	0x0	—	Average-queue-size delay. The number of core-clock cycles to wait ([AVG_DLY + 10] × 8) before calculating the moving average for each QOS level. Larger AVG_DLY values cause the moving averages of all QOS levels to track changes in the actual free space more slowly. Smaller IPD_RED_QUE*n*_PARAM[NEW_CON] (and larger IPD_RED_QUE*n*_PARAM[AVG_CON]) values can have a similar effect, but only affect an individual QOS level, rather than all.
<35:0>	PRT_ENB	R/W	0x0	0x0	Port enable. Any bit that is set enables the corresponding port's ability to have packets dropped by RED probability.

IPD RED Queue(0..7) Parameters Registers
IPD_RED_QUE(0..7)_PARAM

The parameter values that control the passing and dropping of packets by the RED engine for QOS level *n*. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:49>	—	RAZ	—	—	Reserved.
<48>	USE_PCNT	R/W	0	0	Use packet/page count. When this bit is set, RED uses the actual packet-page count in place of the average for RED calculations.
<47:40>	NEW_CON	R/W	0x0	0x0	This value is used control how much of the present actual queue size is used to calculate the new average queue size. The value is a number from 0 to 256, which represents NEW_CON/256 of the actual queue size that is used in the calculation. The number in this field plus the value of AVG_CON must be equal to 256. Larger IPD_RED_PORT_ENABLE[AVG_DLY] values cause the moving averages of all QOS levels to track changes in the actual free space more slowly. Smaller NEW_CON (and larger AVG_CON) values can have a similar effect, but only affect an individual QOS level, rather than all.
<39:32>	AVG_CON	R/W	0x0	0x0	This value is used control how much of the present average queue size is used to calculate the new average queue size. The value is a number from 0 to 256, which represents AVG_CON/256 of the average queue size that will be used in the calculation. The number in this field plus the value of NEW_CON must be equal to 256. Larger IPD_RED_PORT_ENABLE[AVG_DLY] values cause the moving averages of all QOS levels to track changes in the actual free space more slowly. Smaller NEW_CON (and larger AVG_CON) values can have a similar effect, but only affect an individual QOS level, rather than all.
<31:0>	PRB_CON	R/W	0x0	0x0	Used in computing the probability of a packet being passed or drop by the WRED engine. The field is calculated to be $(255 \times 2^{24})/(\text{PASS-DROP})$ where PASS and DROP are the field from the IPD_QOS*n*_RED_MARKS register.

IPD Page Pointer Count Register
IPD_PTR_COUNT

Shows the number of work-queue entries and packet page pointers stored in the IPD. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:19>	—	RAZ	—	—	Reserved
<18>	PKTV_CNT	RO	0	0	Packet pointer valid.
<17>	WQEV_PCNT	RO	0	0	Work-queue-entry pointer valid. This value is 1 when a work-queue entry is being held for use by the IPD. The value of this field should be added to the value of WQE_PCNT for a total count of the work-queue-entry page pointers being held by IPD. When IPD_CTL_STATUS[NO_WPTR] is set, this field represents a packet pointer, not a WQE pointer.
<16:14>	PFIF_CNT	RO	0x0	0x0	PFIF count. (PKT_PCNT + PFIF_CNT + 16) is the number of packet page pointers in IPD.
<13:7>	PKT_PCNT	RO	0x0	0x0	Packet count. (PKT_PCNT + PFIF_CNT + 16) is the number of packet page pointers in IPD.
<6:0>	WQE_PCNT	RO	0x0	0x0	Number of page pointers for work-queue-entry storage that are buffered in the IPD.

IPD Backpressure Port RED-Enable Register
IPD_BP_PRT_RED_END

When IPD applies backpressure to a port and the corresponding enable bit in this register is set, the RED unit will drop packets for that port. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:40>	—	RAZ	—	—	Reserved
<39:0>	PRT_ENB	R/W	0x0	0x0	Port enable. The ports corresponding to set bit positions in this field allow RED to drop back when port-level backpressure is applied to the port. The applying of port-level backpressure for this RED dropping does not take into consideration the value of IPD_PORT*n*_BP_PAGE_CNT[BP_ENB].

IPD Queue0 Free Page Count Register
IPD_QUE0_FREE_PAGE_CNT

The number of free-page pointers that are available for use in the FPA for Queue 0. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31:0>	Q0_PCNT	RO	0x0	0x0	Queue 0 page-pointer count. The number of Queue 0 page pointers available.

IPD Clock Count Register
IPD_CLK_COUNT

This register counts the number of core-clock cycles since the deassertion of reset. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	CLK_CNT	RO	0x0	0x0	Clock count. This counter is cleared to 0x0 when reset is applied and increments on every rising edge of the core clock.

IPD PWP Pointer FIFO Control Register
IPD_PWP_PTR_FIFO_CTL

This register allows reading of the page pointers stored in the IPD PWP FIFO. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:61>	—	RAZ	—	—	Reserved.
<60:54>	MAX_CNTS	RO	0x8	0x8	Maximum count. Maximum number of packet pointers or WQE pointers that could be in the FIFO. When IPD_CTL_STATUS[NO_WPTR] is set, this field only represents the maximum number of packet pointers; WQE-Pointers are not used in this mode.
<53:46>	WRADDR	RO	—	—	WQE read address. The current FIFO work-queue-entry read address.
<45:38>	PRADDR	RO	—	—	Packet read address. The current FIFO packet read address.
<37:9>	PTR	RO	—	—	The output of the PWP_FIFO.
<8>	CENA	R/W	1	1	Low-active chip enable to the read port of the PWP_FIFO. This field also controls the multiplexer select that steers RADDR to the PWP_FIFO. **WARNING:** Setting this field to 0 allows reading of the memories through the PTR field, but causes unpredictable operation of the IPD under normal operation.
<7:0>	RADDR	R/W	0x0	0x0	Sets the address to read from in the PWP_FIFO. Addresses 0 through 63 contain packet pointers while addresses 64 through 127 contain WQE pointers.

IPD PRC Holding Pointer FIFO Control Register
IPD_PRC_HOLD_PTR_FIFO_CTL

This register allows reading of the page pointers stored in the IPD PRC holding FIFO. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:39>	—	RAZ	—	—	Reserved.
<38:36>	MAX_PKT	RO	0x5	0x5	Maximum packet count. Maximum number of packet pointers that could be in the FIFO.
<35:33>	PRADDR	RO	—	—	Packet read address. The current packet pointer read address.
<32:4>	PTR	RO	—	—	The output of the PRC holding FIFO.
<3>	CENA	R/W	1	1	Low-active chip enable that controls the multiplexer select that steers RADDR to the holding FIFO. **WARNING:** Setting this field to 0 allows reading of the memories through the PTR field, but causes unpredictable operation of the IPD under normal operation.
<2:0>	RADDR	R/W	0x0	0x0	Sets the address to read from in the holding FIFO in the PRC. This FIFO holds packet pointers to be used for packet-data storage.

IPD PRC Port Pointer FIFO Control Register
IPD_PRC_PORT_PTR_FIFO_CTL

This register allows reading of the page pointers stored in the IPD PRC port FIFO. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:44>	—	RAZ	—	—	Reserved.
<43:37>	MAX_PKT	RO	0x10	0x10	Maximum packet. Maximum number of packet pointers that are in the FIFO.
<36:8>	PTR	RO	—	—	The output of the PRC port pointer FIFO.
<7>	CENA	R/W	1	1	Low-active chip enable to the read port of the PWP_FIFO. This field also controls the multiplexer select that steers RADDR to the PWP_FIFO. **WARNING:** Setting this field to 0 allows reading of the memories through the PTR field, but causes unpredictable operation of the IPD under normal operation.
<6:0>	RADDR	R/W	0x0	0x0	Sets the address to read from in the port FIFO in the PRC. This FIFO holds packet pointers to be used for packet-data storage.

IPD Packet Pointer Valid Register
IPD_PKT_PTR_VALID

This register contains the value of the fetched packet pointer. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:29>	—	RAZ	—	—	Reserved.
<28:0>	PTR	RO	—	—	Packet pointer value.

IPD Work-Queue Entry Pointer Valid Register
IPD_WQE_PTR_VALID

This register contains the value of the fetched work-queue-entry pointer. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:29>	—	RAZ	—	—	Reserved.
<28:0>	PTR	RO	—	—	Work-queue-entry pointer value. When IPD_CTL_STATUS[NO_WPTR] is set, this field represents a packet pointer, not a WQE pointer.

IPD BIST STATUS Register
IPD_BIST_STATUS

This register is the BIST status for IPD memories. See Table 7–7 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:18>	—	RAZ	—	—	Reserved
<17>	CSR_MEM	RO	0	0	CSR Register Memory Bist Status.
<16>	CSR_NCMD	RO	0	0	CSR IOB Commands Memory Bist Status.
<15>	PWQ_WQED	RO	0	0	PWQ PIP WQE DONE memory BIST status.
<14>	PWQ_WP1	RO	0	0	PWQ WQE PAGE1 PTR memory BIST status.
<13>	PWQ_POW	RO	0	0	PWQ POW MEM memory BIST status.
<12>	IPQ_PBE1	RO	0	0	IPQ PBE1 memory BIST status.
<11>	IPQ_PBE0	RO	0	0	IPQ PBE0 memory BIST status.
<10>	PBM3	RO	0	0	PBM3 memory BIST status.
<9>	PBM2	RO	0	0	PBM2 memory BIST status.
<8>	PBM1	RO	0	0	PBM1 memory BIST status.
<7>	PBM0	RO	0	0	PBM0 memory BIST status.
<6>	PBM_WORD	RO	0	0	PBM_WORD memory BIST status.
<5>	PWQ1	RO	0	0	PWQ1 memory BIST status.
<4>	PWQ0	RO	0	0	PWQ0 memory BIST status.
<3>	PRC_OFF	RO	0	0	PRC_OFF memory BIST status.
<2>	IPD_OLD	RO	0	0	IPD_OLD memory BIST status.
<1>	IPD_NEW	RO	0	0	IPD_NEW memory BIST status.
<0>	PWP	RO	0	0	PWP memory BIST status.

Chapter 8

Packet Output Processing Unit (PKO)

This chapter contains the following subjects:

Overview

This chapter discusses the CN50XX centralized packet-output processing (PKO) unit. It gathers packet data from L2/DRAM and sends it out on any/all of the RGMII or PCI interfaces. It can have a combined total of up to 5 output ports for sending packets between all these destinations. This effectively means that PKO supports a total of up to 5 simultaneous in-flight packets. The packets sent out to the different ports share some of the same PKO hardware resources, but logically the PKO unit treats the different in-flight packets independently.

The PKO unit supports up to 32 queues to buffer the packets to be sent out to the 5 available hardware ports. Figure 8–1 shows the conceptual architecture. Each port can have a variable number of queues attached to it, up to a maximum of 16, that must be contiguous.

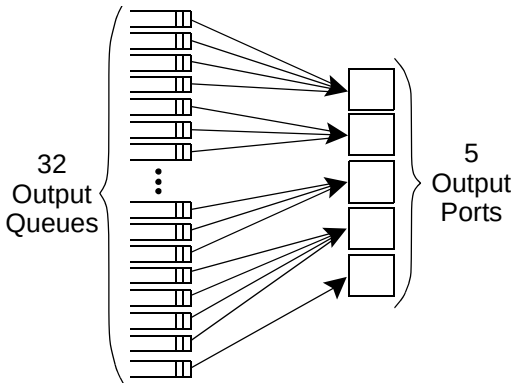


Figure 8–1 PKO Conceptual Architecture

Figure 8–2 shows the PKO unit internal architecture in more detail. The packet data and queue commands enter the unit from L2/DRAM via the PKOB bus. The PKO unit buffers the packet data and transfers it to the RGMII or PCI output ports via the POB bus.

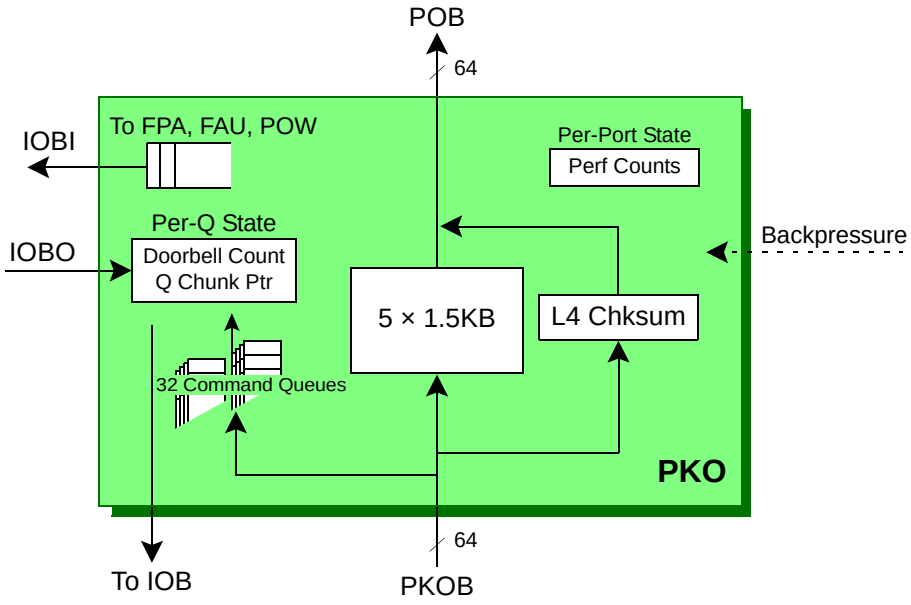


Figure 8–2 PKO Internal Architecture

Each output queue is a chunked list of commands in L2/DRAM. Each packet transfer is one command in an output queue. The core software writes a queue doorbell register inside the PKO unit after it adds commands to a queue. These doorbell writes arrive at the PKO unit via the IOBO bus. The PKO hardware reads commands from L2/DRAM and can cache four (or more) commands per queue inside the unit. The PKO unit performs a mixed weighted-fair/static-priority arbitration among queues to select packets to send out a port. The PKO unit also obeys per-port back-pressure indications (from the RGMII and PCI interfaces) when transferring packet data out the POB bus.

The PKO unit also contains hardware to calculate the L4 checksum for a flexibly-positioned IP TCP/UDP packet. When this checksum option is selected for a packet, the PKO unit can often buffer an entire packet in its internal store before inserting the checksum as it sends the packet out the POB bus. The internal store is 7.5 KB, giving 1.5 KB of internal store per port. The PKO hardware only reads the packet data from L2/DRAM once to send out a packet, unless it is directed to calculate the checksum for a TCP/UDP packet that is too large to fit in the internal buffering for a port.

The PKO unit has both a linked mode and a true-gather mode that can construct full packets from multiple packet segments in L2/DRAM, and can optionally (on a per-segment basis) free up the buffers containing the packet data and/or gather list after sending a packet. The PKO unit is bi-endian and supports L2 cache bypass on a packet-by-packet basis. The PKO unit frees queue chunks/buffers after it reads all the command words from the chunk. All these gather/packet/chunk buffer frees exit the PKO unit by the IOBI bus.

The queue commands can optionally direct PKO to perform up to two Fetch-and-Add (FAU) register decrements after sending a packet. They can also direct PKO to either submit a work queue entry or write an L2/DRAM byte to zero. All of these operations require IOBI bus transactions.

8.1 Output Ports

The PKO unit sends packets to up to 5 output ports numbered as follows:

- PKO ports 0–2 = packet interface 0 ports 0–2
- PKO ports 32–33 = PCI interface ports 0–1

The packet interface may not use all the available ports allocated to it. When an interface uses fewer ports, only the lower port numbers exist.

The PKO unit treats all ports identically.

8.2 Output Packet Format and TCP/UDP Checksum Insertion

Figure 8–3 shows the format requirements for packets entering and exiting the PKO unit.

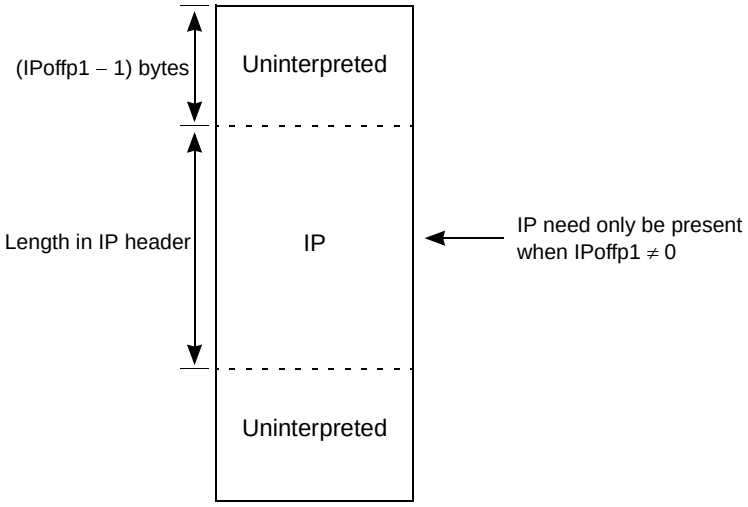


Figure 8–3 Format Requirements for Packets

If the command for a packet does not direct the PKO unit to calculate a TCP/UDP checksum, there is no format requirement. The PKO unit simply interprets the output packet data as a byte stream. The packet may actually contain an L2 header, an IP packet, and other fields, but the PKO unit does not interpret them.

If the command for a packet directs the PKO unit to calculate a TCP/UDP checksum, the command also contains a field (IPoffp1) that indicates the first byte of the IP packet. The following are the requirements to use the PKO unit TCP/UDP L4 checksum generation:

- The IP must be TCP or UDP (i.e. the protocol/next_header field must be either TCP (6) or UDP (17)) and must entirely reside in this packet. The IP can be followed by a trailer.
- If IPv4, the packet must not have options and must not be a fragment (i.e. HLEN must equal 5 and both MF and offset must be zero).
- If IPv6, the packet must not have any extension headers prior to the TCP/UDP header (i.e. the next_header field in the IPv6 header must be either TCP (6) or UDP (17)).
- In the UDP case, the PKO unit uses the UDP header length field to calculate the checksum, so the UDP length value must be legal (e.g. it must not indicate that the UDP data is longer than the IP length allows).

8.3 PKO Output Queue

Figure 8–4 shows the structure of each output queue. Each output queue is a linked-list of chunks, or buffers. Software allocates these chunks and the PKO hardware frees them. The PKO hardware reads words from the output queue (starting at the tail), and traverses the next chunk buffer pointer to the next chunk when it reaches the last word of a chunk. When the PKO hardware jumps chunks like this, it frees the earlier chunk/buffer to the FPA hardware-managed pool selected by the PKO_REG_CMD_BUF[POOL] CSR.

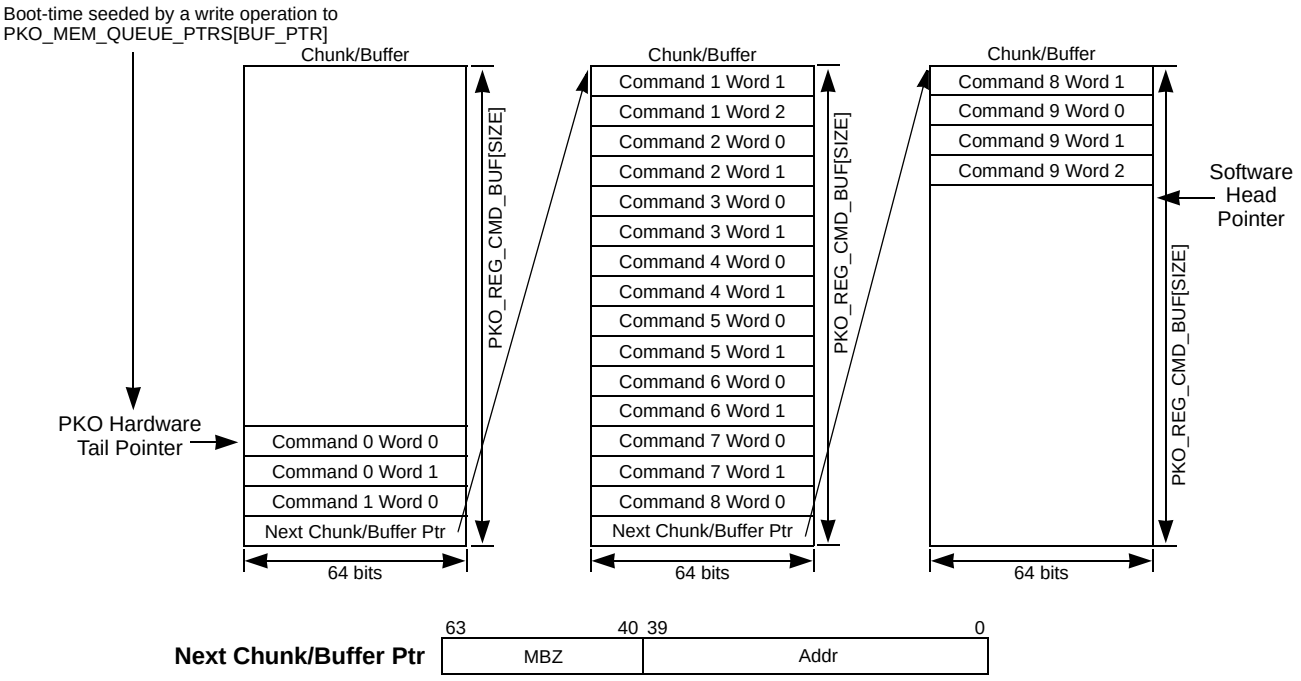


Figure 8–4 Structure of Output Queue

The PKO_REG_CMD_BUF[SIZE] CSR indicates the size of a chunk. Figure 8–4 shows an example where the chunk size is 16 64-bit words, the minimum legal value. The final word in a chunk is a next chunk buffer ptr, but all other words are valid command words. Each command has the information the PKO hardware needs to send out a single packet. This example shows ten commands, or ten packets, in the output queue. The commands are numbered from 0 to 9, with smaller numbers for the older commands. The packets for the older commands are sent first. Every command is either two or three words. Note that the words of an individual command may straddle a chunk boundary. This happens for commands 1 and 8 in Figure 8–4.

The PKO hardware maintains the tail pointer for the queue and the core software maintains the head pointer. To insert a packet into a queue, software must first write the command words for the packet into the queue, allocating chunks if necessary, and then write the PKO doorbell address for the queue with the number of words added to the queue. (“PKO Operations” on page 349 describes the store operations to the PKO unit that are a doorbell write.) Software must guarantee that L2/DRAM contains the number of words indicated in the doorbell write before the doorbell write reaches the PKO.

The distance between the head pointer and the tail pointer is both the size of the output queue and the outstanding doorbell count. The size of the output queue is limited only by the available memory and the 20-bit outstanding doorbell counter for a queue. Core software must guarantee that each queue is smaller than 2^{20} words.

Note that the PKO hardware may read the next chunk buffer ptr as soon as the doorbell count indicates that the next-to-last word in a chunk contains a valid command word. This implies that software must allocate the next chunk buffer, and set the next chunk buffer ptr in the prior chunk to point to it, as soon as it writes the next-to-last chunk word.

Software can “ring the doorbell” with any number of command words. Software can issue a doorbell write for each individual command word, or can issue exactly one doorbell write for each command, or can accumulate the words from multiple commands into a single doorbell write. The only requirements are that the number of valid command words in the queue must be at least as large as the doorbell count, and the next chunk buffer pointers interspersed among the words must also be set up properly. Section 8.8 describes the operations needed to ring the queue doorbells.

Figure 8–4 shows only a single output queue. The PKO hardware can manage 32 possible output queues. The command cache is 8words per queue.

At boot time, software must configure each queue with the original next chunk buffer pointer (i.e. the starting tail pointer) and must attach a queue to a port. This is done with writes to the PKO_MEM_QUEUE_PTRS CSR. This configuration can attach from one to eight queues to a given output port. All queues attached to a single port must have contiguous queue IDs, but otherwise there are no queue/port attachment constraints. Software must supply both the queue ID and its attached port ID with each doorbell write operation.

Figure 8–4 also shows the next chunk buffer pointer format. The primary component is the Addr field that selects a legal L2/DRAM byte location. Though Addr is a byte address, it must be naturally aligned on a 128-byte cache block boundary, so its least-significant seven bits must be 0x0.

8.4 PKO Commands

The PKO reads packets from L2/DRAM memory and passes them out a port whenever one of the queues attached to the port have valid commands. Each command can individually specify whether the PKO hardware should calculate and insert the IP TCP/UDP checksum.

Each command can also specify either linked or gather mode to construct the packet. In either case, the PKO hardware reads the packet segments from L2/DRAM, and can also optionally free the buffers containing the segments to one of the eight available hardware-managed free pools. (See “Free Pool Unit (FPA)” on page 253.) The free pool selection and the decision whether to free the buffer containing a segment can be on a segment-by-segment basis. A command can also specify whether to free buffers on a command-by-command basis.

Figure 8–5 shows an example of usage of the linked packet construction mode. This example shows a packet consisting of three segments. A segment is simply contiguous packet bytes which, except for the last segment, must be preceded by a 64-bit pointer to the next segment. In the example, the entire packet is the concatenation of all of segment 0, all of segment 1, and the first bytes of segment 2.

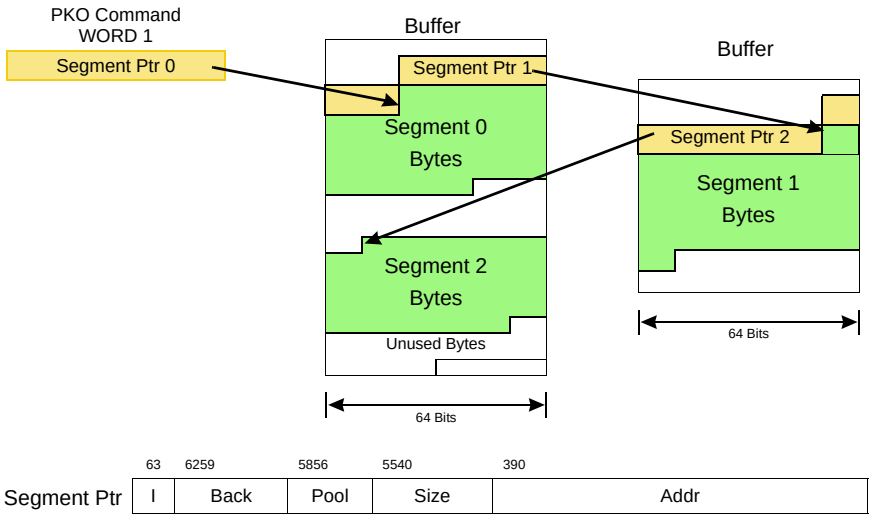


Figure 8–5 Usage of the Linked Packet Construction Mode

Figure 8–5 also shows the format of the 64-bit segment pointer. It has the byte address (Addr), the size in bytes (size), the hardware free pool (Pool), the distance from the start of the segment to the start of the buffer (Back), and free selection bit (I) for the next segment. This segment pointer format matches the format of WORD 1 in the command and is described below.

In linked mode, the segment pointer in WORD 1 of the command must be naturally-aligned on a 64-bit boundary. All other segment pointers precede the start of the segment data by exactly eight bytes and so are not aligned on a 64-bit boundary when the segment start is not aligned on a 64-bit boundary.

Figure 8–6 shows the same packet example as Figure 8–5, but it shows the packet segments connected with gather mode rather than linked mode.

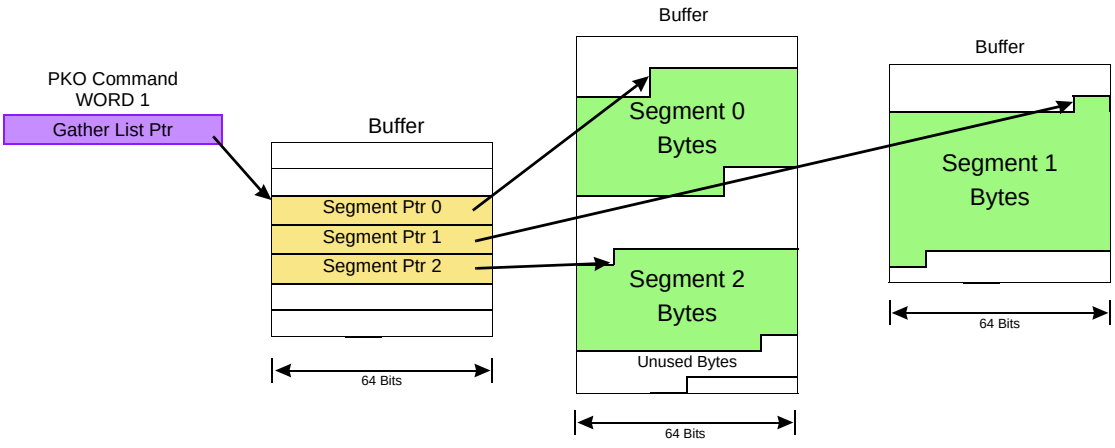


Figure 8–6 Packet Segments Connected with Gather Mode

Gather mode is different from linked mode in the following three ways:

- There need not be any segment pointers preceding any segment data.
- Instead, all segment pointers are contained in a gather list. The gather list (i.e. all segment pointers) must be aligned on a 64-bit boundary. The buffer containing a gather list can be freed just like the buffer containing any segment can be freed.
- WORD 1 of the command points indirectly to the gather list rather than directly to the first segment. The format of this gather list pointer is similar to a segment pointer.

Note the distinction between a segment and a buffer in Figures 8–5 and 8–6. Buffers that the PKO hardware frees must start on a 128 byte boundary and be at least 128 bytes. A segment can be as small as one byte and can start on any byte alignment. Multiple segments may reside in a single buffer. The first and last segment in Figure 8–5 share the same buffer, and the second segment resides in a different buffer. Segments can share buffers freely, but when they do, software must guarantee that the PKO hardware does not free a buffer before reading all segments contained in the buffer. When the command/buffer pointer directs the PKO hardware to free the buffer containing a segment, PKO may free the buffer immediately after it reads the segment. This implies that when multiple segments share a buffer, only the last segment in the list (in both linked and gather mode) can direct the PKO to free the buffer.

The last segment is special in both linked and gather modes. First, in linked mode, a segment pointer need not precede the last segment. The field does not need to exist, and the PKO hardware will never read it. Second, in both linked and gather mode, the last segment may be larger than needed to contain the packet. The total bytes field in the command determines the number of bytes in the packet. The PKO hardware uses this field to determine when to stop taking bytes from the last segment. This total bytes field must be larger than the sum of the sizes of the segments other than the last, and less than or equal to the sum of the sizes of all segments including the last. In other words, the last segment must hold at least one byte of the packet.

Figures 8–5 and 8–6 show PKO in the default big-endian mode. PKO also supports full little-endian mode on a packet-by-packet basis. In little-endian mode, the following apply:

- The output queue and commands, segment pointers, and gather-list formats are unchanged.
- The Addr field of a segment pointer is a little-endian byte pointer rather than a big-endian one.
- Segment bytes are fetched from L2/DRAM in little-endian format rather than big-endian format.
- Unaligned segment pointers (in linked-packet mode) must be in correct little-endian format, eight bytes prior to the segment bytes (i.e. the least-significant eight bytes of the pointer are first, the next least-significant bytes are next, etc.).

Each command may individually specify that the PKO hardware complete up to three additional operations after it sends the packet out the port:

- Decrement up to two Fetch-and-Add (FAU) registers. The particular register, the size of the decrement operation (64, 32, 16, or 8-bits), and the decrement amount (either one or the packet (i.e. WORD0[Total Bytes]) size in bytes) can be specified individually for each decrement. With these operations, software can maintain accurate queue sizes (bytes and/or packets) for each queue in selected FAU registers.
- Either:
  - write an L2/DRAM byte to zero, or

- submit a work queue entry

With the L2/DRAM write, software can poll until the PKO hardware sends a packet. The work queue entry is pre-constructed by software, so can cause the software to perform arbitrary tasks.

Figure 8–7 shows the format of commands in the PKO output queues. Every command is either two or three 64-bit words. The last word only exists when WORD0[R] is set.

PKO Command Format

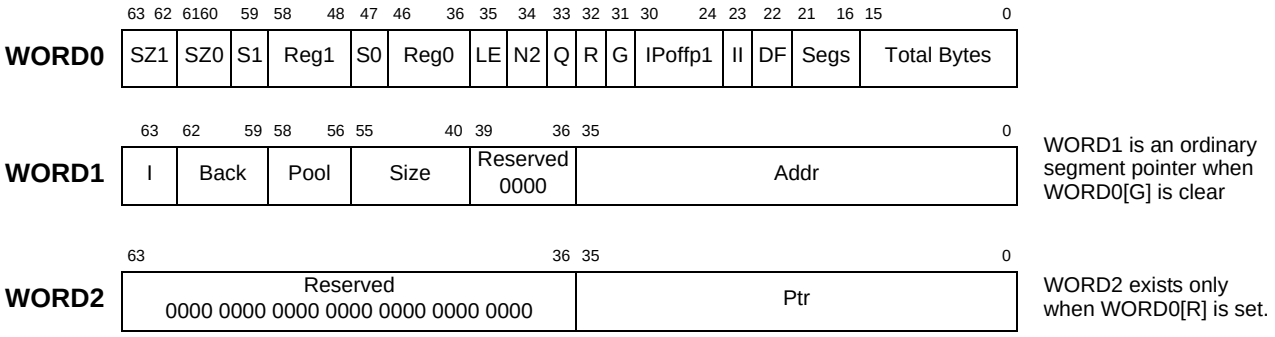


Figure 8–7 Format of Commands in the PKO Output Queues

The WORD1 format matches the segment pointer format shown in Figure 8–5, though WORD1 is a gather list pointer rather than a segment pointer when WORD0[G] is set. The description of the WORD1 fields describe all their uses.

The remainder of this section provides descriptions of each field in the command.

WORD0[SZ1] Size of the WORD0[Reg1] FAU subtract (0 = 8-bit, 1 = 16-bit, 2 = 32-bit, 3 = 64-bit) that can follow a packet send. Must be zero when WORD0[Reg1] is zero.

WORD0[SZ0] Size of the WORD0[Reg0] FAU subtract (0 = 8-bit, 1 = 16-bit, 2 = 32-bit, 3 = 64-bit) that can follow a packet send. Must be zero when WORD0[Reg0] is zero.

WORD0[S1] If set, the WORD0[Reg1] FAU subtract value is 1, else the WORD0[Reg1] subtract value is WORD0[Total Bytes]. Must be zero when WORD0[Reg1] is zero.

WORD0[Reg1] If non-zero, PKO subtracts from the WORD0[Reg1] FAU register after it sends the packet. If WORD0[Reg1] is zero, PKO does not subtract from WORD0[Reg1] after it sends the packet. WORD0[Reg1] is a byte address in the FAU register file, and must be naturally-aligned based on SZ1. (i.e. The WORD0[SZ1] least-significant bits of WORD0[Reg1] must be zero.)

When IOB_CTL_STATUS[PKO_ENB] is clear, WORD0[Reg1] is a big-endian pointer; when it is set, WORD0[Reg1] is a little-endian pointer

WORD0[S0] If set, the WORD0[Reg0] FAU subtract value is 1, else the WORD0[Reg0] subtract value is WORD0[Total Bytes]. Must be zero when WORD0[Reg0] is zero.

WORD0[Reg0] If non-zero, PKO subtracts from the WORD0[Reg0] FAU register after it sends the packet. If WORD0[Reg0] is zero, PKO does not subtract from WORD0[Reg0] after it sends the packet. WORD0[Reg0] is a byte address in the FAU register file, and must be naturally-aligned based on SZ0. (i.e. The WORD0[SZ0] least-significant bits of WORD0[Reg0] must be zero.)

When IOB_CTL_STATUS[PKO_ENB] is clear, WORD0[Reg0] is a big-endian pointer; when it is set, WORD0[Reg0] is a little-endian pointer

WORD0[LE] Determines PKO's endian mode. When clear (default), PKO operates in big-endian mode, as the diagrams in this section show. When set, all segment pointers and segment bytes are interpreted in little-endian order.

WORD0[N2] Determines PKO's L2 cache allocation. When clear, PKO allocates all load data into the L2 cache. When set, PKO does not allocate blocks containing segment bytes into the L2 cache.

NOTE: PKO always allocates command chunks and gather lists into the L2 cache.

Also, PKO always allocates packet data into the cache on the first packet read of a two-pass TCP/UDP checksum calculation. Refer to Section 8.7 for more performance details regarding this calculation.

WORD0[N2] affects performance, but otherwise does not affect OCTEON behavior. It may be advantageous to set WORD0[N2] if packet data will not be used after PKO sends the packet, which will allow the L2 cache to retain other more useful information.

WORD0[Q] If set and WORD0[R] is set, WORD2 contains a pointer to a work queue entry, and PKO will submit the work to POW after it sends the packet. If WORD0[Q] is clear and WORD0[R] is set, WORD2 contains an L2/DRAM byte pointer, and PKO will write the byte to zero after it sends the packet. WORD0[Q] must be zero when WORD0[R] is zero.

WORD0[R] If set, WORD2 exists and PKO will either submit a work queue entry or write a byte to zero after it sends the packet, depending on WORD0[Q]. If clear, WORD2 does not exist.

WORD0[G] Determines whether the PKO hardware operates in linked or gather mode for the packet. If set, PKO is in gather mode and WORD1 points to a gather list. If clear, PKO is in linked mode and WORD1 points to the first segment.

WORD0[IPoffp1] Determines whether the PKO hardware calculates the TCP/UDP checksum for a packet and specifies the location of the first byte of the IP packet. If WORD0[IPoffp1] is zero, PKO does not insert a TCP/UDP checksum into the packet. If WORD0[IPoffp1] is non-zero, PKO hardware will generate and insert the TCP or UDP checksum for an IP packet. The IP packet must be exactly WORD0[IPoffp1]–1 bytes from the beginning of the packet. Section 8.2 discusses requirements for and restrictions for TCP/UDP checksum generation.

WORD0[II] If set, ignore the I bit (effectively, force them all to zero) in all segment and gather pointers.

WORD0[DF] If set, by default buffers should not be freed for all segments (and any gather list, if present). If clear, by default buffers should be freed for any segments (or any gather list, if present). If WORD0[II] is set, WORD0[DF] is more than the default – it completely controls whether a buffer is freed by the PKO hardware. If WORD0[II] is clear, the I bit in the segment/gather pointer can invert the default behavior (i.e. If WORD0[II] is clear, the buffer containing the segment is freed when {WORD0[DF] $\oplus$ (segment/gather pointer I bit)} = 0.)

WORD0[Segs] The number of segments. If WORD0[G] is clear, WORD0[Segs] indicates the number of linked segments. If WORD0[G] is set, WORD0[Segs] is also the number of entries in the gather list.

WORD0[Total Bytes] The number of bytes in the packet. All packet bytes must reside in exactly WORD0[Segs] segments. Must not exceed 2^{16} – 8 bytes (65528 bytes).

I (WORD1 and all segment pointers) Invert bit of the segment/gather list pointer. Used to change the default freeing behavior for the buffer containing the segment or gather list. I exists both in WORD1 and in all segment pointers. When WORD0[II] is clear, I inverts the free behavior described by the WORD0[DF] bit. I is never used by the PKO hardware when WORD0[II] is set.

Back (WORD1 and all segment pointers) Back field of the segment/gather list pointer. When freeing the buffer containing this segment or gather list, the PKO hardware finds the start address of the buffer using the Back and Addr fields:

```
Buffer Start Address = ((Addr >> 7) - Back) << 7
```

NOTE: The Back value can specify a Buffer Start Address that is from 0-2047 bytes prior to Addr, and the Buffer Start Address must always be aligned on a 128 byte cache block boundary.

Pool (WORD1 and all segment pointers) Pool field of the segment/gather list pointer. The PKO hardware frees to this (FPA hardware-managed) free pool when it frees the buffer containing this segment or gather list.

Size (WORD1 and all segment pointers) Size field of the segment/gather list pointer.

- If WORD0[G] is clear or the pointer does not reside in WORD1 of the command, Size is the number of bytes in the segment pointed at by Addr.
- If WORD0[G] is set, WORD1[Size] is the number of segments and must exactly equal WORD0[Segs].

Note that for all segment pointers other than the last, Size is also the number of packet bytes pointed at by Addr, but that the last segment may contain fewer than Size packet bytes. Size must not exceed 2^{16} – 8 bytes (65528 bytes).

Addr (WORD1 and all segment pointers) Addr field of the segment/gather list pointer. Addr is always a physical memory byte pointer.

- If WORD0[G] is clear or the pointer does not reside in WORD1 of the command, Addr points to the start of the packet data for this segment and can be any byte alignment.
- If WORD0[G] is set, Addr points to the gather list – an array of segment pointers – and must be 64-bit aligned (i.e. the least-significant three bits must be clear).
- If WORD0[LE] is set, Addr is a little-endian byte pointer. Otherwise, Addr is a big-endian byte pointer.

WORD2[Ptr] A physical L2/DRAM byte pointer used after sending the packet. Ptr does not exist when WORD0[R] is clear. If WORD0[Q] and WORD0[R] are set, WORD2[Ptr] points to a work queue entry that PKO will add to a POW output work queue, and must be aligned on a 64-bit boundary. (See Section 7.5 in Chapter 7, for a description of the output work queues and work queue entries.) If WORD0[Q] is clear and WORD0[R] is set, WORD2[Ptr] points to a byte that will be written to zero, and may be any byte alignment. In the byte-pointer case, the pointer is a big-endian pointer when PKO_REG_FLAGS[STORE_BE] is set. Otherwise, the pointer is a little-endian pointer.

8.5 PKO Queue Arbitration Algorithm

When PKO needs to send a packet through a port, the PKO hardware uses a mixed-priority/weighted-fair algorithm to select from among the queues attached to the port (a variable number, up to eight queues).

Write operations to PKO_MEM_QUEUE_PTRS configure the mapping of PKO output queues to ports. All queues that map to a single port must reside in a contiguous range of queue identifiers (QIDs). Boot-time mapping of queue QID to port PID via PKO_MEM_QUEUE_PTRS[QID, PID, IDX,TAIL] also establishes the position of the queue in the range for the port.

- IDX must be the relative position of QID in the range (i.e. QID – min(QID)).
- TAIL must be set only for the last queue in the range for the port - i.e. only for the one with the largest QID.

This output queue to port configuration should be done only once at boot time before any output packets flow on the queues/ports.

Each queue has an associated QOS_MASK field. Software must configure QOS_MASK for each used queue at boot time via PKO_MEM_QUEUE_PTRS[QOS_MASK]. During normal operation, software can also optionally update the QOS_MASK fields for the queues via PKO_MEM_QUEUE_QOS[QOS_MASK]. QOS_MASK controls scheduling of the queue. When QOS_MASK is set to zero, the queue is disabled, and can no longer send packets through the port.

The priority algorithm allows a configurable number of queues to be designated (static) high priority. When present, these priority queues must be the lowest-index queues attached to the port. The lower the queue index, the (statically) higher the priority of the queue. PKO visits these priority queues in order prior to visiting the remaining weighted-fair queues. All priority queues are higher-priority than the weighted-fair queues. The QOS_MASK value for a priority queue must be either all 1s (0xFF) or all 0s.

All queues that are not priority queues are weighted-fair queues. The weighted-fair algorithm has eight rounds, where each weighted-fair queue can optionally participate in each round. With weighted-fair queues, QOS_MASK is a bit-mask that determines the rounds that a queue participates in. The fewer rounds that a queue participates in, i.e. the fewer bits set in QOS_MASK, the lower priority the queue is. Of course, all disabled queues (QOS_MASK is all 0s) never participate in any round, and any enabled queue must participate in at least one of the eight rounds.

To use static queues within a port:

- The static priority queues must be the first queues attached to the port (highest priority first).
- PKO_MEM_QUEUE_PTRS[STATIC_P] must be set for all queues attached to the port.
- PKO_MEM_QUEUE_PTRS[STATIC_Q] must be set for the queues attached to the port that are static priority queues.
- PKO_MEM_QUEUE_PTRS[S_TAIL] must be set for the last static priority queue attached to the port.
- PKO_MEM_QUEUE_PTRS[QOS_MASK] for a static queue must be either all 1s (to enable the queue) or all 0s (to disable the queue).

In the pseudo-code below:

- **current_round** and **queue_within_round** are the current weighted round-robin state of the port.
- **max_queue_idx** is the number of queues attached to the port minus one.
- **queue_base** is the first queue attached to the port.
- **num_static_priority** is the number of static priority queues attached to the port.
- **QOS_MASK** is PKO_MEM_QUEUE_PTRS[QOS_MASK] for the queue
- **doorbell_count** is the doorbell count for the queue (Sections 8.3 and 8.8 explain the doorbell).

The following pseudo-code shows the details of the algorithm:

```
struct {
   int current_round, queue_within_round;
   int queue_base;   int max_queue_idx;
   int num_static_priority;
} port_state[36];

struct {
   int QOS_MASK;
   int doorbell_count;
} queue_state[128];

// port_queue_arb() only runs when at least one queue attached
// to the port has a non-zero doorbell count and has a QOS_MASK
// that is non-zero.

// port_queue_arb() returns the queue to select from.
int port_queue_arb(int port) {
   bool found = false;
   int queue;
   int loops = 0;
   do {
     int rel_q;
     bool is_static = (loops < port_state[port].num_static_priority);
     if(is_static)
       rel_q = loops;
     else
       rel_q = port_state[port].queue_within_round;
     queue = port_state[port].queue_base + rel_q;
     bool participate = (queue_state[queue].QOS_MASK >> port_state[port].current_round) & 1;
     if(participate && queue_state[queue].doorbell_count)
       found = true;
     if(!is_static) {
       if(port_state[port].queue_within_round == port_state[port].max_queue_idx) {
          port_state[port].current_round = (port_state[port].current_round + 1) & 7;
          port_state[port].queue_within_round = port_state[port].num_static_priority;
       }
       else
          port_state[port].queue_within_round ++;
     }
     loops++;
   } while(!found);
   return(queue);
 }
```

For example, for a four-queue configuration with these requirements:

- one static priority queue (highest priority)
- one (generally, but not statically) higher-priority queue
- two low-priority queues

One reasonable configuration might be:

- queue 0: static priority, QOS_MASK = 0xFF
- queue 1: weighted-fair queue, QOS_MASK = 0xFF
- queue 2: weighted-fair queue, QOS_MASK = 0x80
- queue 3: weighted-fair queue, QOS_MASK = 0x08

Example 8–1 Four-Queue Configuration Example, After Queue 0 is Drained

		Rounds							
		0	1	2	3	4	5	6	7
Queues	1	1	3	4	5	6	8	9	10
	2	2	–	–	–	–	–	–	–
	3	–	–	–	–	7	–	–	–

If all four queues had packets to send with this configuration, first all queue 0 packets would be sent. Then, after eight rounds, eight packets would be sent through queue 1, and one packet would be sent through each of queue 2 and queue 3. Also, as configured, four packets would flow through queue 1 between interruptions in the flow from queue 2 or queue 3. Example 8–1 indicates the order of packet transition after queue 0 is drained.

8.6 PKO Don't-Write-Back (DWB) Calculation

A hardware-pool free command includes a don't-write-back (DWB) argument that specifies the maximum number of don't-write-back commands to execute on the coherent memory bus. When PKO_REG_FLAGS[ENA_DWB] is clear, these DWB counts will always be zero.

If PKO_REG_FLAGS[ENA_DWB] is set, the DWB count for output queue chunks is:

((PKO_REG_CMD_BUF[SIZE] × 8) + 127) >> 7

If PKO_REG_FLAGS[ENA_DWB] is set, the DWB count for the buffer containing a segment that is freed is:

(((**Addr** & 127) + **Size** + 127) >> 7) + **Back**

which may result in a coherent memory bus DWB command for all cache blocks from the start of the buffer up to the block that includes the last byte of the segment.

- **Addr** and **Back** here come directly from the segment pointer.
- **Size** is always the number of bytes in the segment or gather list that are used by PKO.
  - For all gather lists and all segments except the last one for a packet, **Size** is the number of bytes in the segment/gather list.
  - For the last segment of a packet, **Size** may be less than the number of bytes in the segment since the number of bytes used in the segment can be limited by the total number of bytes in the packet.

8.7 PKO Performance

The PKO can send 20 million or more small packets per second when many ports are used. When only a single port is used, PKO can send 8–12 million or more small packets per second through the single port. PKO performance is better when the output (command) queue, gather lists, and packet data reside in the L2 cache rather than in DRAM.

If the command for a packet directs the PKO unit to calculate a TCP/UDP checksum, and the entire packet fits inside the internal buffer space allocated to the port (i.e., the entire packet is less than 1.5 KB), then the PKO unit reads the packet once, buffers it internally and calculates the checksum. It then inserts the checksum as it reads the packet from its internal buffer and sends it out an output port. The PKO unit cannot send any part of the packet out the POB bus until it reads all packet data from L2/DRAM.

If the command for a packet directs the PKO unit to calculate a TCP/UDP checksum, but the entire packet does not fit inside the internal buffer space allocated to the port (i.e., the entire packet is greater than 1.5 KB), then the PKO unit reads the packet twice from L2/DRAM. It calculates the TCP/UDP checksum during the first read. The data from the second read flows through the internal buffering and immediately out the POB bus, and the checksum logic inserts the checksum as it is sent out. In this case, the PKO unit starts sending the packet out the POB bus before it finishes reading the packet the second time.

If the command for a packet does not direct the PKO unit to calculate a TCP/UDP checksum, the PKO unit only reads the packet data from L2/DRAM once and the data flows through the internal buffering to the POB bus. In this case, the PKO unit starts sending the packet out the POB bus before it finishes reading the packet.

PKO has two performance counters for each port: one that counts the number of packets sent out the port, and another that counts the number of bytes sent out the port. The packet counts can be read through PKO_MEM_COUNT0 (after setting up PKO_REG_READ_IDX), and the byte counts can be read through PKO_MEM_COUNT1 (again, after setting up PKO_REG_READ_IDX).

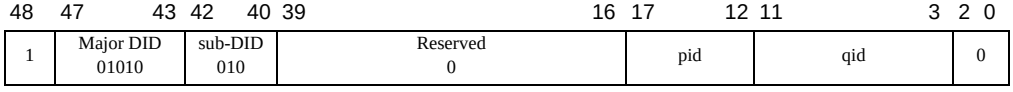
8.8 PKO Operations

This section shows the detailed bit formats and codes for the various PKO transactions.

NOTE: Only 64-bit stores are allowed (i.e. only SDs are allowed); no loads or IOBDMA operations.

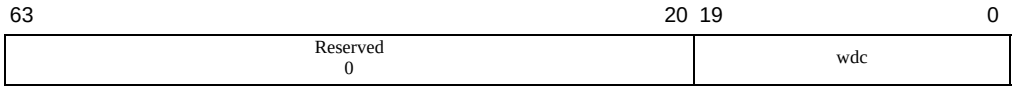
8.8.1 Store Operations

Doorbell Writes **Physical Address to Store to PKO**



- **qid** - queue identifier (0–31 legal)
- **pid** - port identifier (0–35 legal). The port identifier must match the port previously attached to the selected queue. (The port was attached to a queue by a previous write for the queue to the PKO_MEM_QUEUE_PTRS CSR.)

Store Data on a Store to PKO



- **wdc** - contains the number of words to add to the doorbell count

8.9 PKO Registers

The packet-output processing registers are listed in Table 8–1

Table 8–1 PKO Registers

Register	Address	CSR Type[1]	Detailed Description
PKO_REG_FLAGS	0x0001180050000000	RSL	See page 352
PKO_REG_READ_IDX	0x0001180050000008	RSL	See page 353
PKO_REG_CMD_BUF	0x0001180050000010	RSL	See page 352
PKO_REG_GMX_PORT_MODE	0x0001180050000018	RSL	See page 353
PKO_REG_QUEUE_MODE	0x0001180050000048	RSL	See page 353
PKO_REG_BIST_RESULT	0x0001180050000080	RSL	See page 353
PKO_REG_ERROR	0x0001180050000088	RSL	See page 354
PKO_REG_INT_MASK	0x0001180050000090	RSL	See page 354
PKO_REG_DEBUG0	0x0001180050000098	RSL	See page 354
PKO_REG_DEBUG1	0x00011800500000A0	RSL	See page 354
PKO_REG_DEBUG2	0x00011800500000A8	RSL	See page 354
PKO_REG_DEBUG3	0x00011800500000B0	RSL	See page 355
PKO_REG_QUEUE_PTRS1	0x0001180050000100	RSL	See page 355
PKO_MEM_QUEUE_PTRS	0x0001180050001000	RSL	See page 355
PKO_MEM_QUEUE_QOS	0x0001180050001008	RSL	See page 355
PKO_MEM_COUNT0	0x0001180050001080	RSL	See page 357
PKO_MEM_COUNT1	0x0001180050001088	RSL	See page 358
PKO_MEM_DEBUG0	0x0001180050001100	RSL	See page 358
PKO_MEM_DEBUG1	0x0001180050001108	RSL	See page 358
PKO_MEM_DEBUG2	0x0001180050001110	RSL	See page 359
PKO_MEM_DEBUG3	0x0001180050001118	RSL	See page 359
PKO_MEM_DEBUG4	0x0001180050001120	RSL	See page 359
PKO_MEM_DEBUG5	0x0001180050001128	RSL	See page 360
PKO_MEM_DEBUG6	0x0001180050001130	RSL	See page 360
PKO_MEM_DEBUG7	0x0001180050001138	RSL	See page 361
PKO_MEM_DEBUG8	0x0001180050001140	RSL	See page 361
PKO_MEM_DEBUG9	0x0001180050001148	RSL	See page 361
PKO_MEM_DEBUG10	0x0001180050001150	RSL	See page 362
PKO_MEM_DEBUG11	0x0001180050001158	RSL	See page 362
PKO_MEM_DEBUG12	0x0001180050001160	RSL	See page 362
PKO_MEM_DEBUG13	0x0001180050001168	RSL	See page 363

1. RSL-type registers are accessed indirectly across the I/O bus.

PKO Flags Register
PKO_REG_FLAGS

See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	X	0	Reserved. MBZ.
<3>	RESET	RAZ	0	0	Reset PKO. When set to 1, causes a one-cycle reset pulse to the entire PKO unit.
<2>	STORE_BE	R/W	0	0	Force to big endian. When set to 1, inverts bits[2:0] of all zero-byte write addresses.
<1>	ENA_DWB	R/W	0	0	Enable don't-write-backs. When set to 1, enables the use of Don't-Write-Backs during the buffer freeing operations.
<0>	ENA_PKO	R/W	0	0	Enable the PKO picker. When set to 1, enables the PKO picker and places the PKO in normal operation.

PKO Read Index Register
PKO_REG_READ_IDX

This register provides the read index during a CSR read operation to any of the CSRs that are physically stored as memories (i.e. the CSRs that begin with the prefix PKO_MEM_). IDX[7:0] is the read index and INC[7:0] is an increment that is added to IDX[7:0] after any CSR read. The intended use is to initially write this CSR such that IDX = 0x0 and INC = 0x1. Then, the entire contents of a CSR memory can be read with consecutive CSR read commands.

See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	X	0x0	Reserved. Must be zero.
<15:8>	INC	R/W	0x0	0x0	Increment to add to current index for next index.
<7:0>	IDX	R/W	0x0	0x0	Index to use for next memory CSR read.

PKO Command Buffer Parameters Register
PKO_REG_CMD_BUF

This register sets the command buffer parameters. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:23>	—	RAZ	X	0x0	Reserved. Must be zero.
<22:20>	POOL	R/W	0x0	0x0	Pool number. Specifies the free list used to free command-buffer segments
<19:13>	—	R/W	X	0x0	Reserved. Must be zero.
<12:0>	SIZE	R/W	0x0	0x0	Size. Specifies the number of uint64s per command-buffer segment

PKO GMX Port Mode Register
PKO_REG_GMX_PORT_MODE

See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	X	0x0	Must be zero.
<5:3>	MODE1	R/W	0x0	0x0	Must be zero.
<2:0>	MODE0	R/W	0x0	0x0	Must be zero.

PKO Queue Mode Register
PKO_REG_QUEUE_MODE

Sets the number of queues and amount of local storage per queue. The system has a total of 256 queues and (256 × 8) words of local command storage. This register sets the number of queues that are used. Increasing the value of MODE by 1 decreases the number of queues by a power of 2 and increases the local storage per queue by a power of 2.

MODEn	Queues	Storage/Queue
0	256	64 bytes (8 words)
1	128	128 bytes (16 words)
2	64	256 bytes (32 words)

See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	X	0x0	Must be zero.
<1:0>	MODE	R/W	0x0	0x0	PKO queue modes, must be 0, 1, or 2.

PKO BIST Result Register
PKO_REG_BIST_RESULT

This register provides access to the internal BIST results. Each bit is the BIST result of an individual memory (0 = pass and 1 = fail). See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:33>	—	RAZ	X	0x0	Must be zero.
<32>	CSR	RO	X	0	BIST result of the CSR memories.
<31>	IOB	RO	X	0	BIST result of the IOB memories.
<30>	—	RO	X	0	Reserved.
<29:27>	OUT_CTL	RO	X	0x0	BIST result of the OUT_CTL memories.
<26>	OUT_STA	RO	X	0	BIST result of the OUT_STA memories.
<25>	OUT_WIF	RO	X	0	BIST result of the OUT_WIF memories.
<24:22>	PRT_CHK	RO	X	0x0	BIST result of the PRT_CHK memories.
<21>	PRT_NXT	RO	X	0	BIST result of the PRT_NXT memories.
<20:15>	PRT_PSB	RO	X	0x0	BIST result of the PRT_PSB memories.
<14:13>	NCB_INB	RO	X	0x0	BIST result of the NCB_INB memories.
<12:11>	PRT_QCB	RO	X	0x0	BIST result of the PRT_QCB memories.
<10:8>	PRT_QSB	RO	X	0x0	BIST result of the PRT_QSB memories.
<7:4>	DAT_DAT	RO	X	0x0	BIST result of the DAT_DAT memories.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<3:0>	DAT_PTR	RO	X	0x0	BIST result of the DAT_PTR memories.

PKO Error Register
PKO_REG_ERROR

This register contains error flags for PKO. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	X	0x0	Must be zero.
<2>	CURRZERO	R/W1C	0	0	A packet-data pointer has size = 0.
<1>	DOORBELL	R/W1C	0	0	A doorbell count has overflowed.
<0>	PARITY	R/W1C	0	0	A read-parity error has occurred in the port data buffer.

PKO Interrupt-Mask Register
PKO_REG_INT_MASK

This register contains interrupt-enable mask bits for the PKO errors listed in PKO_REG_ERROR. When a mask bit is set, the corresponding interrupt is enabled. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	X	0x0	Reserved. Must be zero.
<2>	CURRZERO	R/W	0	0	Interrupt-enable bit for packet-data pointer = 0.
<1>	DOORBELL	R/W	0	0	Interrupt-enable bit for doorbell overflow errors.
<0>	PARITY	R/W	0	0	Interrupt-enable bit for read-parity errors.

PKO Debug Register
PKO_REG_DEBUG0

See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	ASSERTS	RO	0x0	0x0	Assertion checks.

PKO Debug Register 1
PKO_REG_DEBUG1

Internal debug register. See Table 8–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	ASSERTS	RO	0x0	0x0	Various assertion checks.

PKO Debug Register 2
PKO_REG_DEBUG2

Internal debug register. See Table 8–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	ASSERTS	RO	0x0	0x0	Various assertion checks.

PKO Debug Register 3
PKO_REG_DEBUG3

Internal debug register. See Table 8–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	ASSERTS	RO	0x0	0x0	Various assertion checks.

PKO Interrupt Mask Register
PKO_REG_QUEUE_PTRS1

This register is used with PKO_MEM_QUEUE_PTRS and PKO_MEM_QUEUE_QOS to allow access to queues 128–255, and to allow mapping of up to 16 queues per port.

NOTE: When programming queues 128–255, the programming sequence must first write PKO_REG_QUEUE_PTRS1 and then write PKO_MEM_QUEUE_PTRS or PKO_MEM_QUEUE_QOS for each queue. See the descriptions of PKO_MEM_QUEUE_PTRS and PKO_MEM_QUEUE_QOS for further explanation of queue programming.

See Table 8–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	X	0x0	Reserved. Must be zero.
<1>	IDX3	R/W	0	0	Bit[3] of Index (distance from head) in the queue array. See PKO_MEM_QUEUE_PTRS[IDX].
<0>	QID7	R/W	0	0	Bit[7] of the Queue ID. See PKO_MEM_QUEUE_PTRS[QID]

PKO Memory Queue Pointers Register
PKO_MEM_QUEUE_PTRS

NOTE: This register is used with PKO_MEM_QUEUE_PTRS to allow access to queues 128–255 and to allow mapping of up to 16 queues per port.

This register sets the queue-to-port mapping and the initial command-buffer pointer per queue. Each queue may map to at most one port, and no more than 16 queues may map to a port. The set of queues that is mapped to a port must be a contiguous array of queues.

- The queue designated in QID is mapped to the port designated in PID.
- The index of queue QID in port PID's queue list is IDX.
- The last queue in port PID's queue array must have its TAIL bit set.
- Unused queues must be mapped to port 63.

STATIC_P marks the port PID to which QID is mapped as having at least one queue with static priority. If any QID that maps to PID has static priority, then all QID that map to PID must have STATIC_P set. Queues marked as static priority must be contiguous and begin at IDX 0. The last queue that is marked as having static priority must have its S_TAIL bit set.

This CSR is a memory of 256 entries and consequently the PKO_REG_READ_IDX CSR must be written before any CSR read operations to this address can be performed. A read of any entry that has not been previously written is illegal and results in unpredictable CSR read data. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	S_TAIL	R/W	X	0	Set if this QID is the tail of the static queues
<62>	STATIC_P	R/W	X	0	Set if any QID in this PID has static priority
<61>	STATIC_Q	R/W	X	0	Set if this QID has static priority
<60:53>	QOS_MASK	R/W	X	0x0	Mask to control priority across eight QOS rounds.
<52:17>	BUF_PTR	R/W	X	0x0	Command buffer pointer, bits <23:17> must be 0x0.
<16>	TAIL	R/W	X	0	Set if this QID is the tail of the queue array.
<15:13>	IDX	WR0	X	0x0	Index[2:0] (distance from head) in the queue array.
<12:7>	PID	WR0	X	0x0	Port ID to which this queue is mapped.
<6:0>	QID	R/W	X	0x0	Queue ID[6:0].

PKO Memory QOS Register
PKO_MEM_QUEUE_QOS

This register sets the QOS mask per queue. The QOS_MASK field is logically and physically the same QOS_MASK field in PKO_MEM_QUEUE_PTRS. This CSR allows the QOS_MASK bits to be written during PKO operation without affecting any other queue state. The port to which queue QID is mapped is port PID. Note that the queue-to-port mapping must be the same as was programmed via PKO_MEM_QUEUE_PTRS.

This CSR is a memory of 256 entries and consequently the PKO_REG_READ_IDX CSR must be written before any CSR read operations to this address can be performed. A read of any entry that has not been previously written is illegal and results in unpredictable CSR read data.

See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:61>	—	RAZ	X	0x0	Must be zero.
<60:53>	QOS_MASK	R/W	X	0x0	Mask to control priority across eight QOS rounds.
<52:13>	—	RAZ	X	0x0	Must be zero.
<12:7>	PID	WR0	X	0x0	Port ID to which this queue is mapped.
<6:0>	QID	R/W	X	0x0	Queue ID.

PKO Memory Count0 Register
PKO_MEM_COUNT0

This register provides the total number of packets seen by PKO per port. Writing to this address clears the entry whose index is specified as COUNT[5:0].

This register is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. A read of any entry that has not been previously written is illegal and results in unpredictable CSR read data. If the counter reaches the maximum value, it wraps to 0x0 and starts over. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	COUNT	R/W1C	X	0x0	Total number of packets seen by PKO.

PKO Memory Count1 Register
PKO_MEM_COUNT1

This register provides the total number of bytes seen by PKO per port. Writing to this address clears the entry whose index is specified as COUNT[5:0].

This register is a memory of 36 entries and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. A read of any entry that has not been previously written is illegal and results in unpredictable CSR read data. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved.
<47:0>	COUNT	R/W1C	X	0x0	Total number of bytes seen by PKO.

PKO Memory Debug0 Register
PKO_MEM_DEBUG0

This register provides internal per-port state intended for debug use only. It is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	FAU	RO	X	0x0	Fetch and add command words.
<35:22>	CMD	RO	X	0x0	Command word.
<21:16>	SEGS	RO	X	0x0	Number of segments/gather size.
<15:0>	SIZE	RO	X	0x0	Packet length in bytes.

PKO Memory Debug1 Register
PKO_MEM_DEBUG1

This register provides internal per-port state intended for debug use only. It is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	I	RO	X	0	I value used for free operation.
<62:59>	BACK	RO	X	0x0	Back value used for free operation.
<58:56>	POOL	RO	X	0x0	Pool value used for free operation.
<55:40>	SIZE	RO	X	0x0	Size in bytes.
<39:0>	PTR	RO	X	0x0	Data pointer.

PKO Memory Debug2 Register
PKO_MEM_DEBUG2

This register provides internal per-port state intended for debug use only. It is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	I	RO	X	0	I value used for free operation.
<62:59>	BACK	RO	X	0x0	Back value used for free operation.
<58:56>	POOL	RO	X	0x0	Pool value used for free operation.
<55:40>	SIZE	RO	X	0x0	Size in bytes.
<39:0>	PTR	RO	X	0x0	Data pointer.

PKO Memory Debug3 Register
PKO_MEM_DEBUG3

This register provides internal per-port state intended for debug use only. It is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	DATA	RO	X	0x0	Work-queue data or Store0 pointer

PKO Memory Debug4 Register
PKO_MEM_DEBUG4

This register provides internal per-port state intended for debug use only. It is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:61>	CMND_SEGS	RO	X	0x0	Internal state.
<60:45>	CMND_SIZ	RO	X	0x0	Internal state.
<44:39>	CMND_OFF	RO	X	0x0	Internal state.
<38:36>	UID	RO	X	0x0	Internal state.
<35>	DREAD_SOP	RO	X	0	Internal state.
<34>	INIT_DWRITE	RO	X	0	Internal state.
<33>	CHK_ONCE	RO	X	0	Internal state.
<32>	CHK_MODE	RO	X	0	Internal state.
<31>	ACTIVE	RO	X	0	Internal state.
<30>	STATIC_P	RO	X	0	Internal state.
<29:27>	QOS	RO	X	0x0	Internal state.
<26:22>	QCB_RIDX	RO	X	0x0	Internal state.
<21:18>	QID_OFF_MAX	RO	X	0x0	Internal state.
<17:14>	QID_OFF	RO	X	0x0	Internal state.
<13:6>	QID_BASE	RO	X	0x0	Internal state.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<5>	WAIT	RO	X	0	Internal state.
<4:3>	MINOR	RO	X	0x0	Internal state.
<2:0>	MAJOR	RO	X	0x0	Internal state.

PKO Memory Debug5 Register
PKO_MEM_DEBUG5

This register provides internal per-port state intended for debug use only. It is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:35>	CURR_PTR	RO	X	0x0	Internal state.
<34:19>	CURR_SIZ	RO	X	0x0	Internal state.
<18:3>	CURR_OFF	RO	X	0x0	Internal state.
<2:0>	CURR_SEGS	RO	X	0x0	Internal state.

PKO Memory Debug6 Register
PKO_MEM_DEBUG6

This register provides internal per-port state intended for debug use only. It is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:11>	—	RAZ	X	0x0	Must be zero.
<10:0>	CURR_PTR	RO	X	0x0	Internal state.

PKO Memory Debug7 Register
PKO_MEM_DEBUG7

This register provides internal per-queue state intended for debug use only. It is a memory of 256 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:59>	QOS	RO	X	0x0	QOS mask to enable the queue when set.
<58>	TAIL	RO	X	0	This queue is the last (tail) in the queue array.
<57:45>	BUF_SIZ	RO	X	0x0	Command buffer remaining size in words.
<44:12>	BUF_PTR	RO	X	0x0	Command word pointer.
<11:6>	QCB_WIDX	RO	X	0x0	Buffer write index for QCB.
<5:0>	QCB_RIDX	RO	X	0x0	Buffer read index for QCB.

PKO Memory Debug8 Register
PKO_MEM_DEBUG8

This register provides internal per-port state intended for debug use only. It is a memory of 256 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:28>	—	RO	X	0x0	Must be zero.
<27:8>	DOORBELL	RO	X	0x0	Doorbell count.
<7:6>	—	RAZ	X	0x0	Must be zero.
<5>	STATIC_P	RO	X	0	Static priority.
<4>	S_TAIL	RO	X	0	Static tail.
<3>	STATIC_Q	RO	X	0	Static priority.
<2:0>	QOS	RO	X	0x0	QOS mask to enable the queue when set

PKO Memory Debug9 Register
PKO_MEM_DEBUG9

This register provides internal per-port state intended for debug use only. It is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:49>	—	RO	X	0x0	Must be zero.
<48:32>	PTRS0	RO	X	0x0	Internal state.
<31:17>	—	RAZ	X	0x0	Must be zero.
<16:0>	PTRS3	RO	X	0x0	Internal state.
<63:28>	—	RO	X	0x0	Must be zero.

PKO Memory Debug10 Register
PKO_MEM_DEBUG10

This register provides internal per-port state intended for debug use only. It is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:49>	—	RO	X	0x0	Must be zero.
<48:32>	PTRS1	RO	X	0x0	Internal state.
<31:17>	—	RAZ	X	0x0	Must be zero.
<16:0>	PTRS2	RO	X	0x0	Internal state.

PKO Memory Debug11 Register
PKO_MEM_DEBUG11

This register provides internal per-port state intended for debug use only. It is a memory of 36 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:23>	—	RO	X	0x0	Must be zero.
<22>	MAJ	RO	X	0x0	Internal state.
<21:19>	UID	RO	X	0x0	Internal state.
<18>	SOP	RO	X	0x0	Internal state.
<17>	LEN	RO	X	0x0	Internal state.
<16>	CHK	RO	X	0x0	Internal state.
<15:3>	CNT	RO	X	0x0	Internal state.
<2:0>	MOD	RO	X	0x0	Internal state.

PKO Memory Debug12 Register
PKO_MEM_DEBUG12

This register provides internal per-port state intended for debug use only. It is a memory of 144 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	FAU	RO	X	0x0	Fetch and add command words.
<35:22>	CMD	RO	X	0x0	Command word.
<21:16>	SEGS	RO	X	0x0	Number of segments/gather size.
<15:0>	SIZE	RO	X	0x0	Packet length in bytes.

PKO Memory Debug13 Register
PKO_MEM_DEBUG13

This register provides internal per-port state intended for debug use only. It is a memory of 144 entries, and consequently PKO_REG_READ_IDX must be written before any CSR read operations to this address can be performed. See Table 8–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	I	RO	X	0	I value used for free operation.
<62:59>	BACK	RO	X	0x0	Back value used for free operation.
<58:56>	POOL	RO	X	0x0	Pool value used for free operation.
<55:40>	SIZE	RO	X	0x0	Size in bytes.
<39:0>	PTR	RO	X	0x0	Data pointer.

Chapter 9

PCI Bus

This chapter contains the following subjects:

Overview

9.1 CN50XX PCI Features

- 32/66 MHz PCI 2.3
- PCI Host Support
  - Internal 4-way bus arbiter
  - Internal PCI clock generator
  - Four interrupt inputs
  - Expanded BAR memory space: 4GB of PCI memory space can map to L2C/DRAM
  - Direct core PCI I/O space, PCI configuration space, PCI IACK, and PCI special command generation
- PCI Memory Space Master Support
  - Four separate 36-bit windows (translated to 64-bit) available for direct core access to PCI memory space
  - Two packet-input ports with efficient host interface
  - Two packet-output ports with efficient host buffer input and condensed interrupts
  - Two DMA engines, each accessible by all cores with flexible host queuing support
- PCI Target Support
  - CN50XX internal CSR access through a PCI memory space BAR0
  - 128MB PCI Memory Space BAR1 (32-entry, translated and protected) 32-bit compatible access to entire L2/DRAM
  - PCI memory space BAR2 (protected) for direct access to L2/DRAM
  - BAR0 can expand to 2GB, mapping the first 2GB of CN50XX's L2/DRAM.
  - BAR1 can expand, ranging up to 2GB, mapping the next portion of CN50XX's L2/DRAM.
  - PCI configuration space access
  - PCI expansion ROM BAR (forwarded to the boot bus)
- Full endian-swapping

9.2 CN50XX Addressing as a PCI Target

As per the PCI specification, CN50XX access via the PCI bus is dictated by the base address registers (BARs) in PCI configuration space. There are three BARs: BAR0, BAR1, and BAR2.

All of the CN50XX BAR0, BAR1, and BAR2 are 64-bit base-address registers supporting PCI memory-space operations as a target. CN50XX has no BARs that accept PCI I/O space operations (as a target).

9.2.1 BAR0 - Memory-Mapped CSR Region

- Can also map the first 2GB of L2/DRAM
- CN50XX's BAR0 PCI control registers reside at offset 0x10–0x17 in PCI configuration space.

Prefetchable	1
Type	Locate anywhere within 64-bit address space.
Size	4KB when PCI_CTL_STATUS_2[BB0] = 0, 2GB when PCI_CTL_STATUS_2[BB0] = 1.

When PCI_CTL_STATUS_2[BB0] = 0, BAR0 is a 64-bit base-address register of 4KB that provides access to CN50XX configuration registers. The CN50XX configuration registers are typically only accessed via BAR0 when CN50XX is a PCI device.

When PCI_CTL_STATUS_2[BB0] = 1, BAR0 is a 64-bit base-address register of 2GB that provides access to CN50XX configuration registers as well as the first 2GB of the L2/DRAM. PCI_CTL_STATUS_2[BB0] is typically clear when CN50XX is a PCI device. CN50XX's BAR0 becomes burstable when PCI_CTL_STATUS_2[BB0] = 1, not single-phase disconnecting.

In all cases, the first 4KB of BAR0 (i.e. addresses on the PCI bus in the range [BAR0 + 0] ... [BAR0 + 0xFFF]) access CN50XX's PCI-type CSRs.

When PCI_CTL_STATUS_2[BB0] = 1, the remaining address space (i.e. addresses on the PCI bus in the range [BAR0 + 0x1000] ... [BAR0 + 0x7FFFFFFF]) are accepted by CN50XX and are mapped to L2/DRAM addresses as follows:

PCI Address Range	CN50XX L2/DRAM Address Range
BAR0 + 0x00001000 ... BAR0 + 0x0FFFFFFF	0x000001000 ... 0x00FFFFFF
BAR0 + 0x10000000 ... BAR0 + 0x1FFFFFFF	0x410000000 ... 0x41FFFFFF
BAR0 + 0x20000000 ... BAR0 + 0x7FFFFFFF	0x020000000 ... 0x07FFFFFF

The consequences of any burst that crosses the end of the PCI address range for BAR0 are unpredictable. CN50XX may disconnect PCI references at this boundary. The results of any burst read that crosses the boundary between [BAR0 + 0x0FFFFFFF] and [BAR0 + 0x10000000] are unpredictable. The consequences of any burst write that crosses this same boundary are unpredictable. The results of any burst read that crosses the boundary between [BAR0 + 0x1FFFFFFF] and [BAR0 + 0x20000000] are unpredictable. The consequences of any burst write that crosses this same boundary are unpredictable.

- CN50XX either retries BAR0 PCI read accesses or responds with a single-data phase and disconnects when PCI_CTL_STATUS_2[BB0] = 0.

- CN50XX either retries BAR0 PCI write accesses or accepts a single-data phase and disconnects when PCI_CTL_STATUS_2[BB0] = 0.

When CN50XX L2/DRAM is accessed via BAR0, PCI_CTL_STATUS_2[BB_ES] is the endian-swap and PCI_CTL_STATUS_2[BB_CA] is the L2 cache-allocation bit for these references.

9.2.2 BAR1 - 32-Bit Memory-Mapped Region

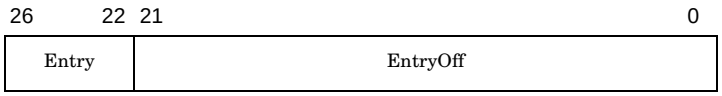
- Can also map the next 2GB of L2/DRAM
- CN50XX's BAR1 PCI control registers reside at offset 0x18–0x1F in PCI configuration space.

Prefetchable	1
Type	Locate anywhere within 64-bit address space.
Size	128MB when PCI_CTL_STATUS_2[BB1] = 0. 1GB (or less) when PCI_CTL_STATUS_2[BB1]=1 and PCI_CTL_STATUS_2[BB1_SIZ]=0. 2GB (or less) when PCI_CTL_STATUS_2[BB1]=1 and PCI_CTL_STATUS_2[BB1_SIZ]=1.

BAR1 is a 64-bit base-address register of 128MB–2GB that is typically used when CN50XX is a PCI host. The first 128MB of BAR1 provides indirect access to the L2/DRAM. When PCI_CTL_STATUS_2[BB1] = 1, the remainder of BAR1, if any, provides direct access to CN50XX low memory (beyond that provided by BAR0).

CN50XX always accepts the first 128MB of BAR1 (i.e. addresses on the PCI bus in the range [BAR1 + 0] ... [BAR1 + 0x07FFFFFF]. When CN50XX receives an address in this BAR1 range, it decodes the lower 27 address bits as follows:

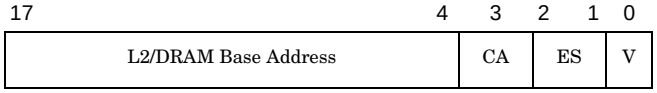
BAR1 27-bit Address



- **Entry** - Selects one of 32 BAR map-table entries.
- **EntryOff** - Offset within the region selected by the BAR1 map table entry.

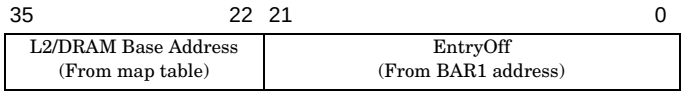
BAR1 Map-Table Entry Format

There are 32 entries configured via the PCI_BAR1_INDEX(0...31) CSRs



- **CA** - Cache attribute. When set, hardware does not allocate into L2 cache.
- **ES** - Endian-swap mode. Refer to Section 9.9.
- **V** - Valid bit. When set, entry is valid. When clear, the entry is invalid and cannot be referenced. When entry is invalid, the operations that reference BAR1 are treated as follows:
  - PCI read - target abort, set PCI_INT_SUM*n*[ILL_RD]
  - PCI write - accept but ignore write, set PCI_INT_SUM*n*[ILL_WR]

The final 36-bit L2/DRAM address referenced by the PCI read/write operations in BAR1 space is:



PCI read/write streams that cross a 4MB boundary access different entries.

- PCI reads disconnect at the 4MB boundary.
- All other BAR1 accesses that cross the 4MB boundary are processed normally.

When PCI_CTL_STATUS_2[BB1] = 1, BAR1 becomes larger than 128 MB, in the 512MB–2GB range. In this case CN50XX maps the remaining address space (i.e. addresses on the PCI bus in the range [BAR1 + 0x08000000] ... [BAR1 + size - 1], where size is the size of BAR1) to CN50XX physical DRAM addresses as follows:

PCI Address Range	CN50XX L2/DRAM Address Range
BAR1 + 0x08000000 ... BAR0 + size – 1	0x88000000 ... 0x7FFFFFFF + size

Table 9–1 shows the effective size of BAR1 when PCI_CTL_STATUS_2[BB1] = 1.

Table 9–1 Effective Size of BAR1

PCI_CTL_STATUS_2		Effective Size	Comment
[BB1_SIZ]	[BB1_HOLE]		
0	0	1024 MB	Normal 1GB BAR
0	1	1008 MB	1GB, 16 MB hole
0	2	992 MB	1GB, 32 MB hole
0	3	960 MB	1GB, 64 MB hole
0	4	896 MB	1GB, 128 MB hole
0	5	768 MB	1GB, 256 MB hole
0	6	512 MB	1GB, 512 MB hole
0	7	Illegal	
1	0	2048 MB	Normal 2GB BAR
1	1	2032 MB	2GB, 16 MB hole
1	2	2016 MB	2GB, 32 MB hole
1	3	1984 MB	2GB, 64 MB hole
1	4	1920 MB	2GB, 128 MB hole
1	5	1792 MB	2GB, 256 MB hole
1	6	1536 MB	2GB, 512 MB hole
1	7	Illegal	

- When PCI_CTL_STATUS_2[BB1] = 1 and PCI_CTL_STATUS_2[BB1_HOLE] = 0, the BAR1 acts completely like an ordinary PCI-defined power-of-2-size BAR.
- When PCI_CTL_STATUS_2[BB1_HOLE] ≠ 0, the BAR1 acts much like an ordinary PCI-defined power-of-2 size BAR, except that CN50XX does not respond to a portion of the PCI addresses. In other words, there is a hole in BAR1. Please note, however, that the hole is always at the end of the BAR1 address range, meaning there really is no hole.

  Table 9–1 indicates the effective size of BAR1 when PCI_CTL_STATUS_2[BB1] = 1 (power-of-two minus hole).

The consequences of any burst that crosses the end of the PCI address range for BAR1 are unpredictable. CN50XX may disconnect PCI references at this boundary.

When L2/DRAM is accessed via the BAR1 address space above 128MB, PCI_CTL_STATUS_2[BB_ES] is the endian-swap and PCI_CTL_STATUS_2[BB_CA] is the L2 cache-allocation bit for these references. Section 9.9 describes the endian-swap modes. Section 9.6 describes the response of CN50XX to BAR1 load/store operations as a target.

9.2.3 BAR2 - 64-bit Memory-Mapped Region

CN50XX's BAR2 PCI control registers reside at offset 0x20–0x27 in PCI configuration space.

PCI_CTL_STATUS_2[BAR2PRES] controls the behavior of BAR2 control registers. Note that PCI_CTL_STATUS_2[BAR2PRES] controls whether external devices can observe the presence of or use BAR2, while PCI_CTL_STATUS_2[BAR2_ENB] determines what happens to accesses by a remote device. PCI_CTL_STATUS_2[BAR2_ENB] is only relevant when PCI_CTL_STATUS_2[BAR2PRES] is set.

When PCI_CTL_STATUS_2[BAR2PRES] = 1, it appears that CN50XX has a BAR2.

When PCI_CTL_STATUS_2[BAR2PRES] = 0, it appears that CN50XX does not have a BAR2:

- reads of offset 0x20–0x27 in PCI configuration space return all 0s
- CN50XX never matches a PCI memory space reference to its BAR2 space.

NOTE: Note that the reset value for PCI_CTL_STATUS_2[BAR2PRES] equals MIO_FUS_DAT3[BAR2_EN]. When MIO_FUS_DAT3[BAR2_EN] = 0, BAR2 appears not present out of reset, and standard PCI discovery sequence firmware (running on a remote host) will not attempt to map BAR2. This is advantageous in systems whose PCI discovery firmware cannot properly map BARs as large as CN50XX's BAR2.

The discussion in the remainder of this section assumes that PCI_CTL_STATUS_2[BAR2PRES] is set.

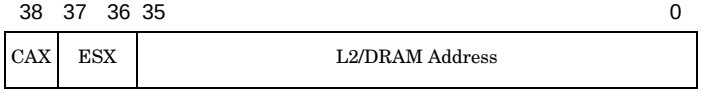
Direct local DRAM addressing.

Prefetchable	1
Type	Locate anywhere within 64-bit address space.
Size	2^{39}

Provides direct access to attached DRAM memory. This BAR is typically used when CN50XX is a PCI host.

CN50XX interprets the lower 39 bits of the PCI address as follows in BAR2:

BAR2 39-bit PCI Addressing



- **CAX** - after XORing with a CSR default (PCI_CTL_STATUS_2[BAR2_CAX]), determines the L2 cache attribute.

  If CAX ⊕ PCI_CTL_STATUS_2[BAR2_CAX] = 1, load/store operations are not allocated to L2 cache.

- **ESX** - after XORing with a CSR default (PCI_CTL_STATUS_2[BAR2_ESX]), determines the endian-swap mode.

- **L2/DRAM Address** - the CN50XX L2/DRAM address.

There is also a CSR enable (PCI_CTL_STATUS_2[BAR2_ENB]) for this region. When this region is disabled, the operations that reference BAR2 are treated as follows:

  - PCI read - target abort, set PCI_INT_SUM*n*[ILL_RD]
  - PCI write - accept but ignore write, set PCI_INT_SUM*n*[ILL_WR]

CA and ES are generated only for the first address referenced. Subsequent addresses use the original value and do not change as the address increments.

Section 9.6 details the response of CN50XX to BAR2 references as a target.

9.2.4 Expansion ROM

CN50XX supports a PCI expansion ROM of 64 KB. CN50XX forwards these expansion ROM reads to its boot bus. See Chapter 12.

- CN50XX either retries expansion ROM PCI read accesses or responds with a single-data phase and disconnects.

9.3 PCI Instruction Input From an External Host

A PCI host may submit instructions to CN50XX. This is a way to feed packets and/or commands into CN50XX, and to create work-queue entries. Instructions are typically submitted to CN50XX when CN50XX is a device on the PCI bus, and are usually not used when CN50XX is the PCI host.

There are two PCI input ports, (enabled via NPI_CTL_STATUS[INS0/1_ENB]). The priority (higher to lower) is Port 0>1. Ports 0/1 correspond to CN50XX internal ports 32/33 (refer to Chapter 7). CN50XX hardware maintains the following for an instruction input ring associated with each port:

- a configured base address (NPI_BASE_ADDR_INPUT0/1
- a configured ring size (NPI_SIZE_INPUT0/1)
- a tail pointer (NPI_P0/1_INSTR_ADDR[NADDR], which gives an approximate view of the tail)

Host software maintains the head of the ring and uses doorbell writes (to PCI_DBELL0/1) to notify CN50XX that the head advanced.

The hardware maintains the doorbell count (PCI_INSTR_COUNT0/1), which also indicates the current number of PCI instructions in the input queue, as well as the distance from head to tail. Another measure of the doorbell count held by the hardware is

NPI_P0/1_INSTR_CNTS[AVAIL] + NPI_P0/1_INSTR_CNTS[FCNT].

Host software can poll the doorbell count to determine how far the hardware has advanced (and prevent ring overflow). The hardware only decrements PCI_INSTR_COUNT*n* after it has read the instruction and all its associated data and gather lists. Thus, when PCI_INSTR_COUNT*n* decrements, host software can reuse the associated host memory locations of the tail PCI instruction.

9.3.1 PCI Instruction Format

Each PCI instruction is either 32 or 64 bytes, controlled by NPI_CTL_STATUS[INS*n*_64B]. Figure 9–1 shows the instruction format:

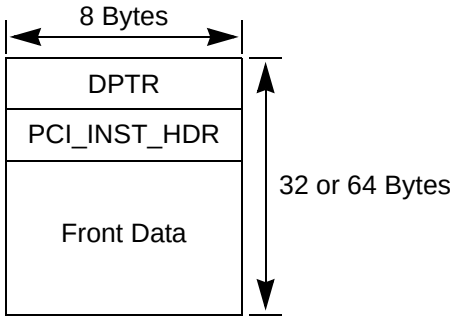


Figure 9–1 Instruction Format

The PCI_INST_HDR format is as follows:

63	62	6148	4742	41 39	38 35	34	3332	31 0
R	G	DLENGSZ	FSZ	QOS	GRP	RS	TT	TAG

- **R** - RAW* packet.

  When R is set, the packet is either RAWFULL or RAWSCH, depending on the parse mode selected for the packet. RAWFULL and RAWSCH packets, as defined in Section 7.2.1, use the QOS, GRP, TT, and TAG fields in the PCI_INST_HDR for their scheduling information (refer to Chapter 7 for more details).

  When R is clear, the packet is neither RAWFULL nor RAWSCH, and the QOS, GRP, TT, and TAG fields are ignored by the hardware.

- **G** - gather is used.

  When G is set, DPTR points at gather-list components (see Figure 9–3) in the host's memory, and the pointers in the gather list point at the packet data in the host's memory. In the case when the G-bit is set:

  - DLENGSZ is the number of entries in the gather list.
  - The length of the packet is
    [sum of the lengths of the gather-list entries] + FSZ + IHI × 8

    where IHI = 1 when the PCI instruction header is included at the beginning of the constructed packet.

  When G is clear, DPTR points directly at the packet data in the host's memory.

  - The length of the packet is DLENGSZ + FSZ + IHI × 8

- **DLENGSZ** - data length/gather-list size.

  When gather is not used (the G bit is clear), this field indicates the length of the packet data (length in bytes) directly pointed at by DPTR.

  When gather is used (the G bit is set), this field indicates the number of entries in the gather list. DLENGSZ must not be zero when the G bit is set. Note that when the G bit is set, the number of gather components used by the instruction is (DLENGSZ + 3)/4.

- **FSZ** - front-data size. Indicates the number of bytes of packet data before the DPTR data (and after the PKT_INST_HDR when it is included in the constructed packet).

  FSZ must be ≤ 23 with a 32-byte instruction, and ≤ 55 with a 64-byte instruction.

An FSZ value between 17 and 23 with a 32-byte instruction can be used to insert a pad between the front data and the DPTR data. The hardware inserts unpredictable bytes for the extra bytes that are not actually contained in the PCI instruction in this case. The same is true for FSZ values between 49 and 55 with a 64-byte instruction.

- **QOS** - quality of service. Selects the one of the eight POW queues that holds the work-queue entry when the R bit is set (i.e. when the packet is RAWFULL or RAWSCH). QOS is ignored by the hardware when the R bit is clear. Refer to Chapters 5 and 7 for more details.
- **GRP** - group. Selects the core group that the work-queue entry is scheduled to when the R bit is set (i.e. when the packet is RAWFULL or RAWSCH). GRP is ignored by the hardware when the R bit is clear. Refer to Chapters 5 and 7 for more details.
- **RS** - real short. When RS is set, the ordinary L2/DRAM copy of the packet is avoided if the packet is dynamically considered short. Section 7.4 describes dynamic-short packets in more detail, but they are largely packets that can fit entirely in the work-queue entry created for the packet.

  The hardware ignores the RS bit if the corresponding PIP_PRT_CFG32/33[DYN_RS] bit is set, which indicates that the port should always avoid buffering dynamic-short packets. The hardware also ignores the RS bit if a PCI_INST_HDR is not included with the constructed packet (i.e. if both the R bit and corresponding NPI_PORT32/33INSTR_HDR[USE_IHDR] are clear).
- **TT** - tag type. Selects the tag type that the work-queue entry has when the R bit is set (i.e. when the packet is RAWFULL or RAWSCH). TT is ignored by the hardware when the R bit is clear. Refer to Chapters 5 and 7 for more details.
- **TAG**. Selects the tag that the work-queue entry has when the R bit is set (i.e. when the packet is RAWFULL or RAWSCH). TAG is ignored by the hardware when the R bit is clear. Refer to Chapters 5 and 7 for more details.

9.3.2 PCI Input Packet

Figure 9–2 shows the packet that the hardware creates from the PCI instruction. The complete packet is the PKT_INST_HDR (optionally included, converted from the PCI_INST_HDR), front data (if any), plus the DPTR data.

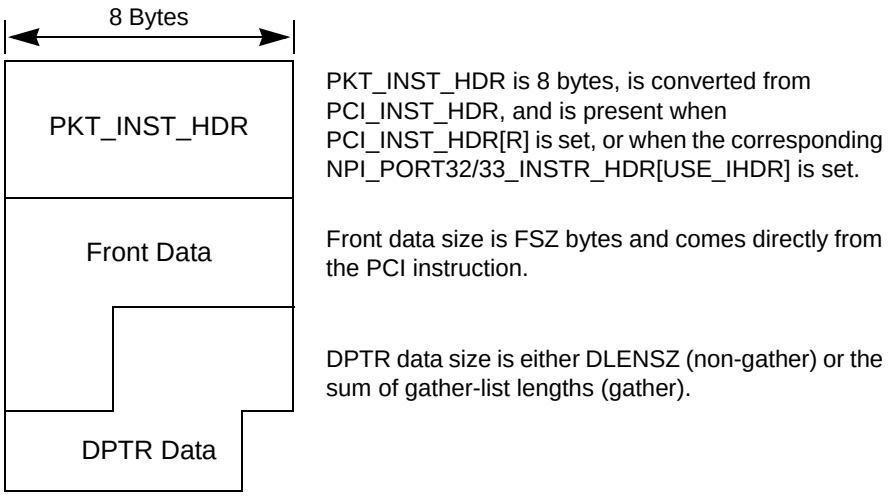


Figure 9–2 Input Packet Format

The PKT_INST_HDR is included with the packet constructed for a PCI instruction when one or both of the two following conditions is met:

- PCI_INST_HDR[R] is set
- the corresponding NPI_PORT32/33INSTR_HDR[USE_IHDR] bit is set, indicating that the PCI instruction header should be included with all packets.

Whenever the PCI instruction header is included in the constructed packet, it is first converted to a PKT_INST_HDR. Sections 7.2.2 and 7.2.5 and Figure 7-2 describe the transformation from PCI_INST_HDR to PKT_INST_HDR in that case. NPI_PORT(0..32)_INSTR_HDR provides the default values for <62:42> of the constructed header for each corresponding port. Additionally, when the corresponding NPI_PORT32/33_INSTR_HDR[PBP] is set, the parse mode and skip values come from the DPTR field directly in the PCI instruction, as described below. Refer to Section 7.2 for more details on parse mode and skip.

When a PCI instruction does not use gather mode (i.e. when PCI_INST_HDR[G] is clear), DPTR in the PCI instruction directly points to the DPTR data in the memory of the remote host. When a PCI instruction uses gather mode, the DPTR in the PCI instruction points to the gather list in the memory of the remote host. A gather list is as many gather components as necessary to contain the entire gather list. Figure 9–3 shows the structure of a gather component. Each LEN, DPTR pair specifies a fragment of the DPTR data in the memory of the remote host. Each gather component must be aligned on a 64-bit boundary (i.e. the DPTR<2:0> must be 0x0 in the PCI instruction when in gather mode), but the DPTRs in the component can have any byte alignment, and the LENs can be any (non-zero) byte length.

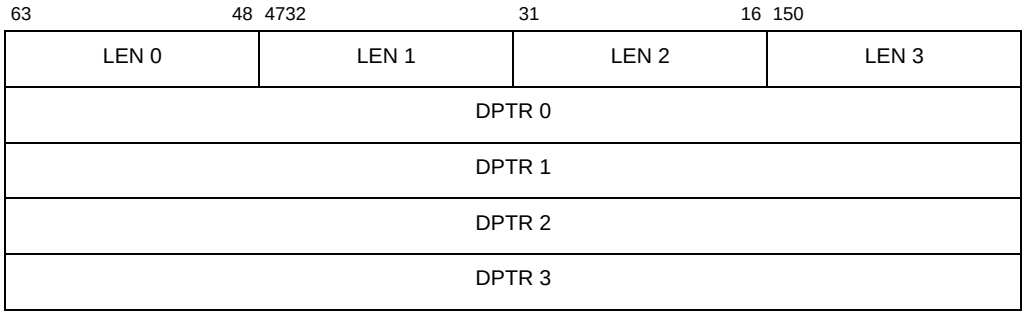


Figure 9–3 Structure of a Gather Component

The CN50XX PCI hardware constructs PCI input packets by PCI read operations for the necessary PCI instructions, gather components, and DPTR data from the remote hosts memory. These PCI read requests require the following elements:

- an address
- an endian-swap mode (see Section 9.9)

There are four possible formats for the DPTRs in the PCI instruction and in the gather components: format 0, format 1, format 2, and format 3.

These four formats produce the following six elements via various means:

1. PCI addresses
2. endian-swap modes
3. no-snoop modes
4. relaxed order
5. parse mode
6. skip lengths

The parse mode and skip lengths are only present in the DPTR in the PCI instruction when selecting the parse mode and skip length on a packet-by-packet basis (i.e. when NPI_PORT32/33_INSTR_HDR[PBP] is set for the port the packet arrived on).

9.3.3 DPTR Formats

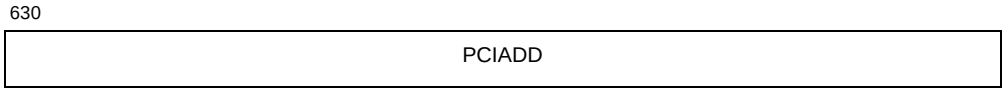
There are four DPTR formats that may be used depending on NPI_INPUT_CONTROL[USE_CSR] (See 'NPI_INPUT_CONTROL" on page 444.) and various NPI_PORT32/33_INSTR_HDR[PBP] values (See 'NPI Port Instruction Header Registers" on page 447.). Table 9–2 describes the DPTR formats used in all cases.

Table 9–2 DPTR Formats

NPI_INPUT_CONTROL [USE_CSR]	NPI_PORT*n*_ INSTR_HDR [PBP] [1]	Not Gather[2]	Gather, DPTR in Instruction[3]	Gather, DPTRs in Components[4]
0	0	Format 1	Format 0	Format 1
0	1	Format 3	Format 2	Format 1
1	0	Format 0	Format 0	Format 0
1	1	Format 2	Format 2	Format 0

1. Ports 32-33 only.
2. Gather is not used: the DPTR field is found directly in the PCI instruction.
3. Gather is used: the DPTR field is found directly in the PCI instruction.
4. Gather is used: the DPTR field is found in the gather component.

DPTR Format 0

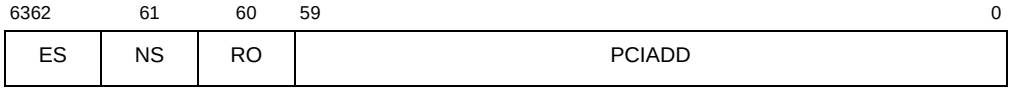


For format 0, the required elements are derived in the following manner:

1. PCIADD<63:0> = PCIADD field shown above
2. endian swap = NPI_INPUT_CONTROL[D_ESR]
3. no snoop = NPI_INPUT_CONTROL[D_NSR]
4. relaxed order = NPI_INPUT_CONTROL[D_ROR]
5. parse mode = N/A
6. skip length = N/A

- Not-gather mode: the DPTR in the PCI instruction is in DPTR format 0 when both the following conditions are met:
  - NPI_INPUT_CONTROL[USE_CSR] = 1.
  - NPI_PORT32/33_INSTR_HDR[PBP] = 0 for the port that the packet arrived on.
- Gather mode:
  - the DPTR in the PCI instruction is in DPTR format 0 when NPI_PORT32/33_INSTR_HDR[PBP] = 0 for the port that the packet arrived on.
  - the DPTRs in the PCI components that represent the gather list for the instruction are in DPTR format 0 when NPI_INPUT_CONTROL[USE_CSR] = 1.

DPTR Format 1

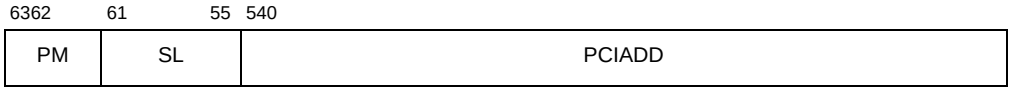


For format 1, the required elements are derived in the following manner:

1. PCIADD<59:0> = PCIADD field shown above
   PCIADD<60> = NPI_INPUT_CONTROL[D_ROR]
   PCIADD<61> = NPI_INPUT_CONTROL[D_NSR]
   PCIADD<63:62> = NPI_INPUT_CONTROL[D_ESR]
2. endian swap = ES field shown above
3. no snoop = NS field shown above
4. relaxed order = RO field shown above
5. parse mode = N/A
6. skip length = N/A

- Not-gather mode: the DPTR in the PCI instruction is in DPTR format 1 when both the following conditions are met:
  - NPI_INPUT_CONTROL[USE_CSR] = 0.
  - NPI_PORT32/33_INSTR_HDR[PBP] = 0 for the port that the packet arrived on.
- Gather mode:
  - the DPTR in the PCI instruction is never DPTR format 1.
  - the DPTRs in the PCI components that represent the gather list for the instruction are DPTR format 1 when NPI_INPUT_CONTROL[USE_CSR] = 0.

DPTR Format 2



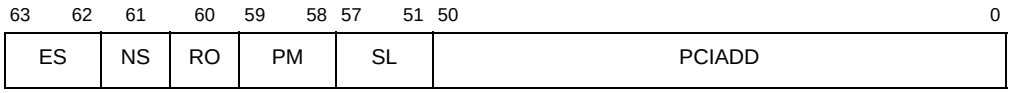
For format 2, the required elements are derived in the following manner:

1. PCIADD<54:0> = PCIADD field shown above
   PCIADD<63:55> = NPI_INPUT_CONTROL[PBP_DHI<12:4>]
2. endian swap = NPI_INPUT_CONTROL[D_ESR]
3. no snoop = NPI_INPUT_CONTROL[D_NSR]
4. relaxed order = NPI_INPUT_CONTROL[D_ROR]
5. parse mode = PM field shown above
6. skip length = SL field shown above

- Not-gather mode: the DPTR in the PCI instruction is in DPTR format 2 when both the following conditions are met:
  - NPI_INPUT_CONTROL[USE_CSR] = 1.
  - NPI_PORT32/33_INSTR_HDR[PBP] = 1 for the port that the packet arrived on.
- Gather mode:

- the DPTR in the PCI instruction is in DPTR format 2 when NPI_PORT32/33_INSTR_HDR[PBP] = 1 for the port that the packet arrived on.
- the DPTRs in the PCI components that represent the gather list for the instruction are never in DPTR format 2.

DPTR Format 3



For format 3, the required elements are derived in the following manner:

1. PCIADD<50:0> = PCIADD field shown above
   PCIADD<59:51) = NPI_INPUT_CONTROL[PBP_DHI<8:0>]
   PCIADD<60> = NPI_INPUT_CONTROL[D_ROR]
   PCIADD<61> = NPI_INPUT_CONTROL[D_NSR]
   PCIADD<63:62> = NPI_INPUT_CONTROL[D_ESR]
2. endian swap = ES field shown above
3. no snoop = NS field shown above
4. relaxed order = RO field shown above
5. parse mode = PM field shown above
6. skip length = SL field shown above

- Not-gather mode: the DPTR in the PCI instruction is in DPTR format 3 when both the following conditions are met:
  - NPI_INPUT_CONTROL[USE_CSR] = 0.
  - NPI_PORT32/33_INSTR_HDR[PBP] = 1 for the port that the packet arrived on.

- Gather mode:
  - the DPTR in the PCI instruction is never DPTR format 3.
  - the DPTRs in the PCI components that represent the gather list for the instruction are never in DPTR format 3.

9.4 PCI Packet Output From CN50XX

There are two output ports destined for PCI in CN50XX, enabled by NPI_CTL_STATUS[OUT0/1_ENB]. Internally, these ports appear much like other ports that exit packet interfaces, until the packets reach the PCI interface (see Chapter 8). These ports are generally used when CN50XX is a device on the PCI bus, when there is a remote host, not when CN50XX is a PCI bus host itself. The CN50XX PCI hardware implements the mechanisms to transfer the packets in the four output ports to the remote host for further processing.

The (remote) host software receives packets by supplying buffer and info pointers. The host software supplies buffer and info pointers to the packet output hardware via an input ring. The ring head is managed by host software: host software rings the doorbell to notify CN50XX of an advance.

The CN50XX hardware maintains the following information for each of the two output queues:

- INPUT RING BASE ADDRESS, INPUT RING SIZE (NPI_BASE_ADDR_OUTPUT*n*[BADDR], NPI_NUM_DESC_OUTPUT*n*[SIZE]). Describes the input ring that supplies buffer/information pairs to CN50XX.
- TAIL POINTER. The hardware uses this to fetch buffer/info pairs from the input ring. NPI_P0/1_DBPAIR_ADDR[NADDR] contains a read-only approximation of the tail pointer for the ring.
- DOORBELL (PCI_PKT_CREDITS0/1[PTR_CNT]). Write operations to the doorbell increments the available size register that CN50XX maintains. Available size is the number of buffer/info pairs that is available for the hardware to fetch from the input ring. The sum of NPI_P0_PAIR_CNTS[AVAIL] and NPI_P0_PAIR_CNTS[FCNT] is the current available-size value.
- PACKET/BYTE COUNT (PCI_PKTS_SENT0/1[PKT_CNT]). Indicates either the number of unacknowledged packets or the number of unacknowledged bytes. The CN50XX hardware increments the unacknowledged count once all buffer pointer and info pointer writes corresponding to a packet are complete. NPI_OUTPUT_CONTROL[P0/1_BMODE] determines whether the increment is a 1 or the number of bytes in the packet. The remote PCI host processor decrements this count (by writing PCI_PKT_CREDITS0/1[PKT_CNT]) to indicate receipt of packets/bytes.
- BUFFER SIZE (NPI_BUFF_SIZE_OUTPUT*n*[BSIZE]). Indicates the number of bytes available at each buffer pointer.
- INFO BYTES (NPI_BUFF_SIZE_OUTPUT0/1[ISIZE]). Indicates the number of bytes from the front of the packet that should appear at the info pointer. The remaining bytes in the packet appear at the data pointer. (In buffer-pointer-only mode (see Section 9.4.2), the INFO BYTES information is not needed nor used.)

When a packet arrives at one of the two PCI output ports, CN50XX must first have been given a buffer/info pointer pair to write the packet into. (If one is not available, the PCI interface hardware will not send the packet (or any subsequent packet arriving at the PCI output port) to the host until the host supplies data/info pairs (via doorbell writes to PCI_PKT_CREDITS0/1[PTR_CNT]).) Once given a buffer/info pair, CN50XX writes the packet into the supplied buffer pointer.

The remote host software should initialize a PCI packet output port by first setting up all the input ring parameters and modes. After that, it should enable the port (NPI_CTL_STATUS[OUT0/1_ENB]). Finally, it can ring the doorbell to provide data/info pairs. The remote host software must not provide any data/info pairs prior to enabling the PCI packet output port.

Figure 9–4 shows the format of a buffer/info pointer pair (in the host processors memory):

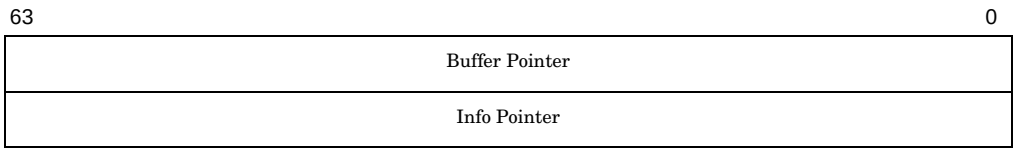


Figure 9–4 Buffer/Info Pointer Pair

Buffer pointers may point to any byte, but all info pointers must point to an aligned, 8-byte quantity (i.e. the lower 3-bits must be zero). The format of the 64-bit buffer and info pointers depends on configuration.

- When the corresponding NPI_OUTPUT_CONTROL[O0/1_CSRM] for the port is set, the formats of the buffer and info pointers are identical to DPTR format 0, as shown in Section 9.3, except that NPI_OUTPUT_CONTROL[O0/1_ES,O0/1_NS,O0/1_RO] are used as the effective endian-swap, no-snoop, and relaxed ordering modes rather than NPI_INPUT_CONTROL[D_ESR,D_NSR,D_ROR], respectively.
- When the corresponding NPI_OUTPUT_CONTROL[O0/1_CSRM] for the port is clear, the formats of the buffer and info pointers is identical to DPTR format 1, as shown in Section 9.3, except that NPI_OUTPUT_CONTROL[O0/1_ES,O0/1_NS,O0/1_RO] are the upper bits <63:60> of the effective PCI address rather than NPI_INPUT_CONTROL[D_ESR,D_NSR,D_ROR], respectively.

The PCI interface can generate interrupts for a remote host in two ways, both based on the PACKET/BYTE COUNT.

1. THRESHOLD COMPARE: whenever the packet/byte count (PCI_PKTS_SENT0/1[PKT_CNT]) exceeds the programmed threshold for the port (PCI_PKTS_SENT_INT_LEV0/1[PKT_CNT]), interrupt bits (PCI_INT_SUM0/1[PCNT0/1]) are set.

2. TIME THRESHOLD INTERRUPT: whenever the packet/byte count (PCI_PKTS_SENT0/1[PKT_CNT]) is non-zero for a selected time threshold (PCI_PKTS_SENT_TIME0/1[PKT_TIME]), and the time interrupt bits are not cleared, time-interrupt bits (PCI_INT_SUM0/1[PTIME0/1]) are set.

There are two packet-output modes (info-pointer mode and buffer-pointer-only mode), selected by NPI_OUTPUT_CONTROL[IPTR_O0/1], and described in Sections 9.4.1 and 9.4.2. Both modes use the identical buffer/info pair format. However, when NPI_OUTPUT_CONTROL[IPTR_O0/1] corresponding to a port is clear (Section 9.4.2 describes this case), the Info Pointer field is not used, though it is still read into CN50XX (via the PCI bus), as it is when NPI_OUTPUT_CONTROL[IPTR_O0/1] corresponding to a port is set (Section 9.4.1 describes this case).

9.4.1 Info-Pointer Mode

Info pointer mode is enabled for a port when the NPI_OUTPUT_CONTROL[IPTR_O0/1] corresponding to the port is set. The CN50XX hardware writes the information to the info pointer after writing the remaining packet data to the buffer pointer in info-pointer mode, so the PCI host can poll in its memory for packet arrival. (The packet is guaranteed to be available after CN50XX writes the packet-length field, so the host can look for changes in the memory at the next packet-length location.) The PCI host can also determine packet arrivals in info-pointer mode by polling the packet/byte count information held in CN50XX, or by receiving an interrupt that is the result of packet/byte-count changes.

The CN50XX hardware writes the packet into the hosts memory as follows in info-pointer mode. In the typical case when the packet (minus Info Bytes) is less than the buffer size for the port, CN50XX writes the packet, excluding the first Info-Bytes bytes of the packet, to the Buffer Pointer location. After this, it writes the first Info-Bytes bytes from the packet together with the packet-length field to the associated info pointer.

If the packet (minus Info Bytes) is larger than the buffer size for the port, CN50XX writes the first buffer-size bytes of the packet, excluding the first Info Bytes bytes of the packet, at the first buffer pointer. It continues allocating and writing further buffer-size byte chunks from the packet into subsequent buffer pointers. After CN50XX writes the entire packet (excluding the first Info Bytes bytes) to the buffer

pointers, it writes the first Info-Bytes bytes from the packet together with the packet-length field to the info pointer associated with the first buffer pointer used to store the packet. Note that when the packet spans across multiple buffer pointers in this way, the info pointers other than the first are not used by CN50XX.

Figure 9–5 shows the format written to the Info Pointer:

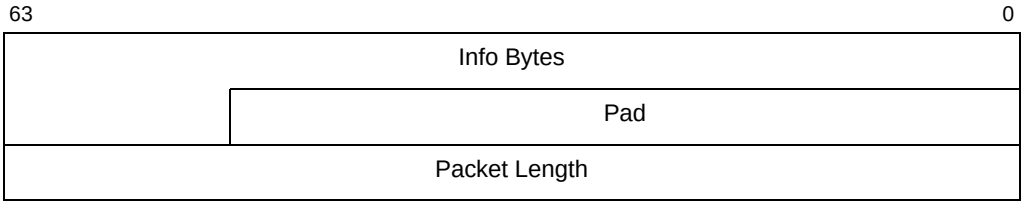


Figure 9–5 Format Written to the Info Pointer

As the info pointer must be 64-bit aligned, the Info Bytes are first written aligned at the info pointer. The hardware adds Pad so that the packet-length field, like the Info-Bytes bytes from the start of the packet, is also aligned on an 8-byte boundary. The value of the Pad bytes is not defined.

The (64-bit) Packet Length field is the total number of bytes in the packet.

PCI host software can calculate the actual number of buffers the PCI hardware used for a packet in info-pointer mode:

```
# bufs = ((Packet Length
            - NPI_BUFF_SIZE_OUTPUT × [ISIZE]    // Info Bytes
            + NPI_BUFF_SIZE_OUTPUT × [BSIZE] - 1// BUFFER SIZE - 1
            / NPI_BUFF_SIZE_OUTPUT × [BSIZE])   // BUFFER SIZE
```

In info-pointer mode, the remote PCI host can learn that a packet arrived in any one of the following ways:

- Receipt of an interrupt (PCI_INT_SUM[PCNT0/1,PTIME0/1] and corresponding PCI_INT_ENB[IPCNT0/1,IPTIME0/1] set) due to a packet/byte count (PCI_PKTS_SENT0/1[PKT_CNT]) change
- Polling packet/byte count (PCI_PKTS_SENT*n*[PKT_CNT]) directly
- Polling for changes in the packet-length field pointed at by an info pointer.

It is illegal to send packets with fewer than Info Bytes in info-pointer mode. (NPI_INT_SUM[PO0/1_2SML] is set when this error happens.)

9.4.2 Buffer-Pointer-Only Mode

Buffer-pointer-only mode is enabled for a port when the NPI_OUTPUT_CONTROL[IPTR_O*n*] corresponding to the port is clear. In this mode, CN50XX ignores the value of the info-pointer field in the buffer/info pair format. Only the buffer pointer is used.

In buffer-pointer-only mode, CN50XX first writes the 8-byte packet-length field (the same format as shown at the bottom of Figure 9–5) at the Buffer Pointer start location, then follows this with the remaining packet data. The packet-length field acts exactly as 8 bytes of additional packet data preceding the real packet data.

As with info-pointer mode, the PCI hardware allocates and writes multiple buffer pointers when necessary in buffer-pointer-only mode. The hardware accounts for the packet-length field, and buffer size always limits the number of bytes written in each supplied buffer pointer.

PCI host software can calculate the actual number of buffers the PCI hardware used for a packet in buffer pointer only mode:

```
# bufs = (Packet Length
         8                                    // for Packet Length field
         + NPI_BUFF_SIZE_OUTPUT × [BSIZE] - 1) // BUFFER SIZE - 1
         / NPI_BUFF_SIZE_OUTPUT × [BSIZE])    // BUFFER SIZE
```

In buffer-pointer-only mode, the remote PCI host can learn that a packet arrived in any one of the following ways:

- Receipt of an interrupt (PCI_INT_SUM[PCNT0/1,PTIME0/1] and corresponding PCI_INT_ENB[IPCNT*n*,IPTIME*n*] set) due to a packet/byte count (PCI_PKTS_SENT*n*[PKT_CNT]) change
- Polling packet/byte count (PCI_PKTS_SENT0/1[PKT_CNT]) directly

NOTE: A remote PCI host cannot learn of packet arrival by polling the packet-length field in buffer-pointer-only mode, because the hardware writes the packet-length field before writing the remainder of the packet.

9.5 PCI DMA Engine Access From Cores

The cores can move data between local (DRAM) memory and remote (PCI) memory by submitting PCI DMA instructions to PCI DMA engines. The PCI DMA engines perform the necessary PCI memory-space and L2/DRAM operations to perform the data move asynchronously to core execution. The engines can notify the cores of completion via either L2/DRAM polling or work-queue entry submittal.

A PCI DMA instruction can be from a minimum of 4 to a maximum of 32 (64-bit) words. Figure 9–6 shows the format of a PCI DMA instruction.

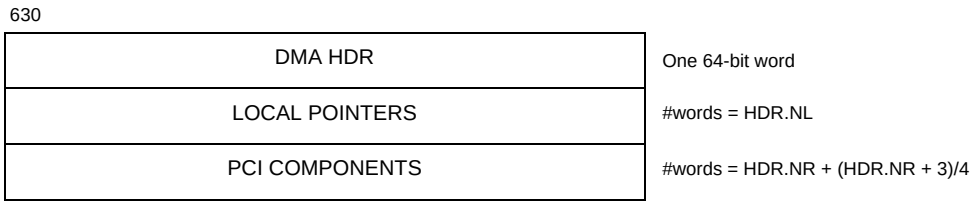


Figure 9–6 PCI DMA Instruction Format

Note that a single PCI DMA instruction may be either contiguous in memory or split into two pieces, depending on how it resides within the PCI DMA instruction chunk, which is defined in the following subsections.

9.5.1 PCI DMA Instruction-Header Format

The DMA header describes the PCI DMA instruction. Figure 9–7 shows the header format.

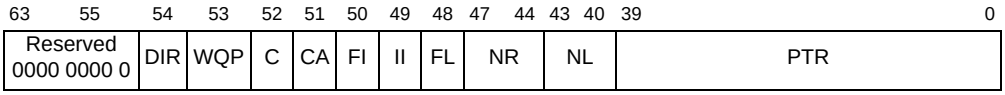


Figure 9–7 PCI DMA Instruction Header Format

- **DIR** - Direction: 1 = inbound, 0 = outbound. A given PCI DMA transfer is either inbound (load from PCI memory space, store into L2/DRAM) or outbound (load from L2/DRAM, store into PCI memory space).
- **WQP** - Work-queue pointer: when WQP = 1, PTR (if non-zero) is a pointer to a work-queue entry that is submitted by the hardware after completing the DMA; when WQP = 0, PTR (if non-zero) is a pointer to a byte in local memory that is written to 0 by the hardware after completing the DMA.
- **C** - Counter: 1 = use counter 1, 0 = use counter 0.

  The C bit selects between the two counters (PCI_DMA_CNT0/PCI_DMA_CNT1) that can optionally be updated after the transfer, and also selects between the two forced-interrupt bits (PCI_INT_SUM*n*[DMA0_FI, DMA1_FI]) that can optionally be set after the transfer. For an inbound PCI DMA operation, C must never be set.

- **CA** - Counter add.

  When CA = 1, the hardware updates the selected counter after it completes the PCI DMA Instruction.

  - If C = 0, PCI_DMA_CNT0 is updated
  - If C = 1, PCI_DMA_CNT1 is updated.

  Note that this update may indirectly cause PCI_INT_SUM*n**[DCNT0,DCNT1,DTIME0,DTIME1] to become set (depending on the PCI_DMA_INT_LEV0* and PCI_DMA_TIME0* settings), so may cause interrupts to occur on a remote PCI host.

  - If NPI_DMA_CONTROL[O_ADD1] = 1, the counter is updated by 1.
  - If NPI_DMA_CONTROL[O_ADD1] = 0, the counter is updated by the total bytes in the transfer.

  When CA = 0, the hardware does not update any counters.

  For an inbound PCI DMA operation, CA must never be set, and the hardware never adds to the counters.

- **FI** - When FI is set, the hardware sets a forced interrupt bit after it completes the PCI DMA Instruction. If C = 0, PCI_INT_SUM*n*[DMA0_FI] is set, else PCI_INT_SUM*n*[DMA1_FI] is set. For an inbound PCI DMA operation, FI must never be set, and the hardware never generates interrupts.
- **II** - Ignore the I bit (i.e. the I bit of the PCI DMA instruction local pointer. See Section 9.5.2).

  For outbound operations (i.e. DIR = 0) when II = 1, ignore the I bit and the FL bit alone determines whether the hardware frees any/all of the local buffers

  - when FL = 1, the local buffer is freed.
  - when FL = 0, the local buffer is not freed.

For outbound operations (i.e. DIR = 0) when II = 0, the I bit in the local pointer selects whether local buffers are freed on a pointer-by-pointer basis: the hardware frees a given local buffer when (FL ⊕ I) is true.

For inbound PCI DMA operations (i.e. DIR = 1), II must never be set, and local buffers are never freed.

- **FL** - Free local buffer.

  When FL = 1, for an outbound operation, it indicates that the local buffer should be freed.

  - If II = 1, the FL bit alone indicates whether the local buffer should be freed
    - if FL = 1, the buffer is freed
    - if FL = 0, the buffer is not freed
  - If II = 0, the local buffer is freed when (FL ⊕ I) is true, where I is the I bit in the local pointer.

  For an inbound PCI DMA operation, FL must never be set, and local buffers are never freed.

- **NR** - The number of PCI pointers (i.e. the number of PCI memory space pointers).
- **NL** - The number of local pointers (i.e. the number of L2/DRAM pointers).
- **PTR** - Pointer, either a work-queue-entry pointer (when WQP = 1) or a local-memory pointer (WQP = 0).
  - When WQP = 1 and PTR ≠ 0x0, the hardware inserts the work-queue entry indicated by PTR into the work queue after the PCI DMA operation is complete. When WQP = 1, PTR<2:0> must be 0x0.
  - When WQP = 0 and PTR ≠ 0x0, the hardware writes the single byte in local memory indicated by PTR to 0x0 after the PCI DMA operation is complete.
  - When PTR = 0x0, the hardware performs no operation after the PCI DMA operation is complete.

  Section 9.5.5 describes when a PCI DMA instruction completes.

9.5.2 PCI DMA Instruction Local-Pointer Format

The local-pointer format, shown in Figure 9–8, describes a memory fragment in the L2/DRAM attached to CN50XX. The number of local pointers in a PCI DMA instruction is HDR.NL.

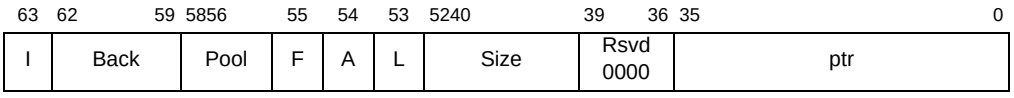


Figure 9–8 Local Pointer Format

- **I** - Invert free. This bit gives the software the ability to free buffers independently for an outbound PCI DMA instruction. See the II bit description in Section 9.5.1.

  I is not used by the hardware when II is set. I must not be set, and buffers are never freed, for an inbound PCI DMA instruction.

- **Back** - Backup amount: allows the start of a buffer that is to be freed to be different from the ptr value. Back specifies the amount to subtract from the pointer to reach the start when freeing a buffer for an outbound PCI DMA instruction.

  The address that is the start of the buffer being freed is:

Buffer start address = ((ptr >> 7) – Back) << 7.

Back is only used by the hardware when the buffer corresponding to ptr is freed. Back must be 0x0, and buffers are never freed, for an inbound PCI DMA instruction.

- **Pool** - Specifies which pool (of the eight hardware-managed FPA free pools) receives the buffer associated with ptr (refer to Chapter 6).

  Pool is only used when the buffer corresponding to ptr is freed. Pool must be 0x0, and buffers are never freed, for an inbound PCI DMA instruction.

- **F** - Full-block writes are allowed. When set, the hardware is permitted to write all the bytes in the cache blocks covered by ptr, ptr + Size - 1. This can improve memory system performance when the write misses in the L2 cache.

  F must not be set for an outbound PCI DMA instruction.

- **A** - Allocate L2. This is a hint to the hardware that the cache blocks should be allocated in the L2 cache, if they were not already allocated.

  Should typically be set to allocate the referenced cache blocks into the L2 cache.

  When the local pointer is a source of data (e.g. an OUTBOUND transfer), the referenced cache blocks are not allocated into the L2 cache as part of completing the DMA (when not already present in the L2) if the A bit is clear.

  When the local pointer is a destination for data (e.g. an INBOUND transfer), the referenced cache blocks are not allocated into the into the cache as part of completing the DMA (when not already present in the L2) if the A bit is clear, and:

  - either the entire cache block is written by this local pointer,
  - or the F bit is set so that the entire cache block can be written

- **L** - Little-endian. When L is set, the data at ptr is in little-endian format rather than big-endian.
- **Size** - Size in bytes of the contiguous space specified by ptr, between 1 and 8184 (i.e. 8K–8).
- **ptr** - L2/DRAM byte pointer. Points to where the packet data starts.

  Ptr can be any byte alignment. Note that ptr is interpreted as a big-endian byte pointer when L is clear, a little-endian byte pointer when L is set.

Note that the sum of the size fields in the HDR.NL local pointers (i.e. the total number of local-pointer bytes) must exactly equal the total number of remote pointer bytes for a given PCI DMA instruction.

9.5.3 PCI DMA Instruction PCI Components and Processing

The PCI components format is a compressed list of PCI memory space pointers and their associated lengths. Up to four lengths are encoded in a length word. A component is a length word and its associated pointers. For example, if HDR.NR = 9, the PCI components portion of the PCI DMA instruction would consist of three components as shown in Figure 9–9.

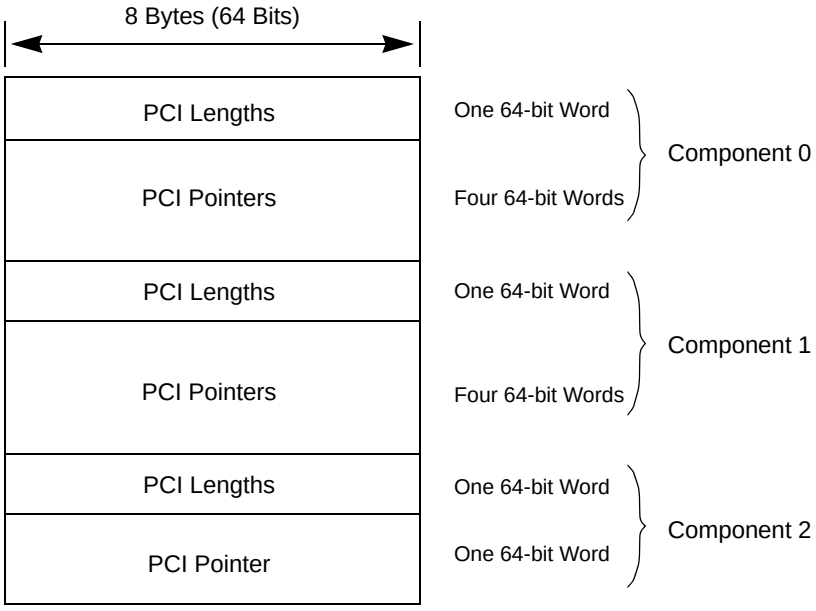


Figure 9–9 PCI Component Format Example When HDR.NR = 9

Note that the sum of the first HDR.NR length fields in the PCI components (i.e. the total number of remote pointer bytes) must exactly equal the total number of local pointer bytes for a given PCI DMA instruction.

Most components are five words: one for the length and four for the pointers. If HDR.NR is not a multiple of 4, the last component is always truncated. The Figure 9–9 example shows a case where three PCI components are necessary to hold the nine remote pointers. The last component is partial and is only two 64-bit words: one to hold the necessary PCI length field, and one to hold the necessary ninth remote pointer.

The format of the PCI length field is shown in Figure 9–10. The values in LEN must be between 1 and 65828 (i.e. 64K–8).

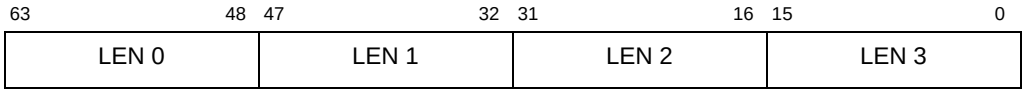


Figure 9–10 PCI Length Field Format

Together with a corresponding length, the PCI pointer describes a fragment of up to 64KB in PCI memory space.

Each PCI pointer is a 64-bit PCI memory-space pointer that takes either DPTR format 0 or DPTR format 1 (as described in Section 9.3), except that NPI_DMA_CONTROL[0_ES,0_NS,0_RO] is used instead of NPI_INPUT_CONTROL[D_ESR,D_NSR,D_ROR] in all places. (NPI_INPUT_CONTROL[D_ESR,D_NSR,D_ROR] is never used by the PCI DMA engines.)

When NPI_DMA_CONTROL[0_MODE] is set, DPTR format 0 is used with NPI_DMA_CONTROL[0_ES,0_NS,0_RO].
When NPI_DMA_CONTROL[0_MODE] is clear, DPTR format 1 is used with NPI_DMA_CONTROL[0_ES,0_NS,0_RO].

With PCI, CN50XX creates either Memory Read or Memory Read Multiple commands (depending on the size of the individual transfer in relation to NPI_PCI_READ_CMD[CMP_SIZE]) to service an inbound PCI DMA request, and Memory Write commands to service outbound PCI DMA transfers.

9.5.4 PCI DMA Instruction Fetching

There are two queues onto which core software pushes PCI DMA INSTRUCTIONS. The PCI DMA engines service the instructions from the two queues at a fixed priority.

Each queue is a linked-list of memory chunks in L2/DRAM memory containing PCI DMA instruction information. Core software adds PCI DMA instructions at the head of a queue and notifies the hardware of the arrival by writing the hardware doorbell register for the corresponding queue (NPI_HIGHP_DBELL[DBELL] or NPI_LOWP_DBELL[DBELL]), typically with the number of words in the instructions. The hardware accumulates the doorbell write operations. The hardware reads an instruction from the tail of a queue and performs the DMA operation. When the hardware exhausts the chunk at the tail of the queue (i.e. reads all the instructions that filled the chunk), the hardware frees the tail chunk and starts reading instructions from the next chunk.

Figure 9–11 is an example of the list:

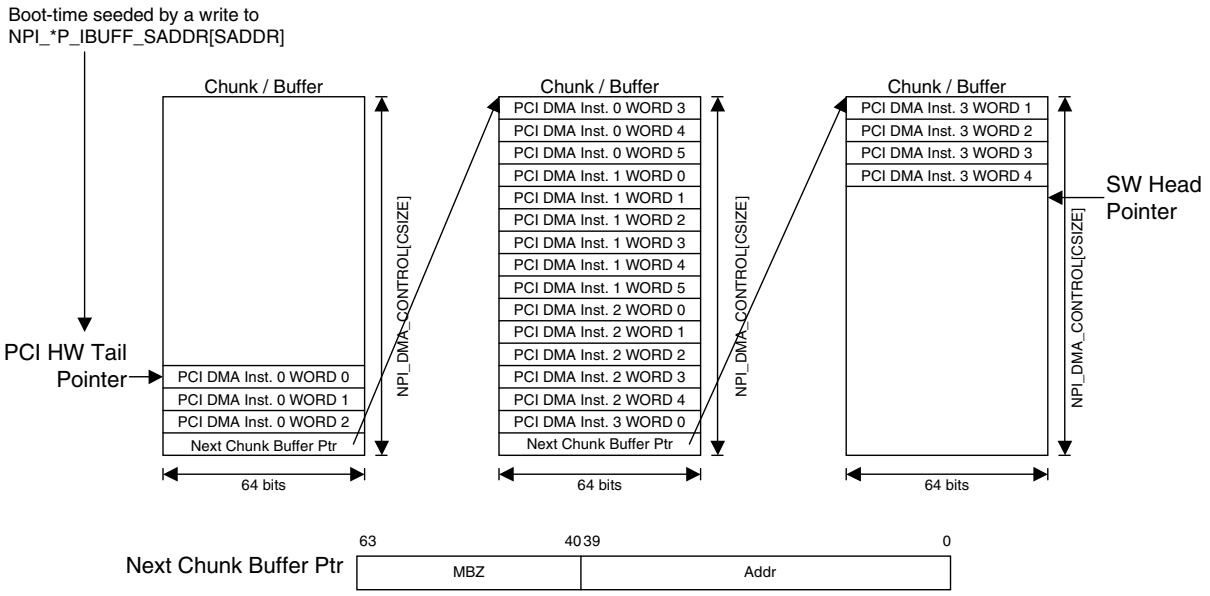


Figure 9–11 Example: Queue Linked-List Memory Chunks

The hardware knows the chunk size through the configuration variable NPI_DMA_CONTROL[CSIZE]. The NPI_DMA_CONTROL[FPA_QUEUE] field selects the free pool that receives a chunk once the tail advances through the next_ptr of the chunk (refer to Chapter 6, the FPA chapter).

The software writes, and the hardware reads, instructions contiguously in a chunk, starting at the first word. A given PCI DMA instruction is normally contiguous in local DRAM memory, but a single instruction must span two chunks if the contiguous

words in the instruction would otherwise over-run the next_ptr of the chunk. In this case, the remainder of the instruction resides at the beginning of the next chunk. The software must allocate the next chunk (and update the next_ptr in the previous chunk) whenever it inserts an instruction that either abuts or over-runs the next chunk buffer pointer. This allows the hardware to read the next_ptr, and traverse to the next chunk, with the same fetch that it uses to read the last instruction word of the chunk.

The hardware knows there are PCI DMA instructions available to be fetched from the queue after the software updates the doorbell count (via writes to NPI_HIGHP_DBELL[DBELL] or NPI_LOWP_DBELL[DBELL]). The software may launch multiple instructions with a single doorbell write. Software can also use multiple doorbell write operations to announce the availability of a single instruction, but this provides poorer performance than a single write operation. For this and other reasons, hardware does not assume that individual doorbell write operations correlate to entire PCI DMA instructions. The hardware only uses the doorbell write operations to indicate valid instruction words in the queue.

The distance between the head pointer and the tail pointer is both the size of the PCI DMA instruction queue and the outstanding doorbell count. The size of the PCI DMA instruction queue is limited only by the available memory.

At boot time, software must configure the PCI DMA instruction queues with the original next chunk buffer pointer (i.e. the starting tail pointer) with a write to NPI_*P_IBUFF_SADDR[SADDR].

Figure 9–11 also shows the next chunk buffer pointer format. The primary component is the Addr field that selects a legal L2/DRAM byte location. Though Addr is a byte address, it must be naturally aligned on a 128 byte cache block boundary, so its least-significant 7 bits must be zero. The same is true for the original next chunk buffer pointer.

Since the hardware maintains the doorbell counts and tail pointers for the queues, the software normally does not need to know them. Approximations of them are available for debug purposes, however.

- NPI_DMA_HIGHP_COUNTS[DBELL] + NPI_DMA_HIGHP_COUNTS[FCNT] approximates the doorbell count for the high-priority queue.
- NPI_DMA_LOWP_COUNTS[DBELL] + NPI_DMA_LOWP_COUNTS[FCNT] approximates the doorbell count for the low-priority queue.
- NPI_DMA_HIGHP_NADDR[ADDR] and NPI_DMA_LOWP_NADDR[ADDR] approximate the tail pointers for the high and low-priority queues, respectively.

9.5.5 PCI DMA Instruction Ordering and Completion

If two outbound PCI DMA instructions use the same queue, the PCI write operations for the two instructions appear on the PCI bus in the order that the instructions were queued. Within an instruction, all write operations appear in order.

Similarly, the PCI read addresses from two inbound instructions in the same queue appear on the PCI bus in queue order, and the read addresses from an individual instruction appear in order.

An outbound PCI DMA transfer completes after the last PCI write transaction appears on the PCI bus. CN50XX hardware does not increment the counter selected by the C bit, nor does it set any interrupt, until the last PCI write appears on the PCI bus.

An inbound PCI DMA transfer completes only after all its local memory stores commit and there are no “prior” uncommitted PCI BAR1/BAR2 stores (from external devices). CN50XX does not guarantee the order of the local memory stores, though it does guarantee that all local memory stores are complete/committed before the instruction completes.

9.5.6 PCI DMA Engine Don’t-Write-Back Calculation

An FPA hardware pool free command includes a don't-write-back (DWB) argument that specifies the maximum number of DWB commands to execute on the coherent memory bus. The PCI DMA engine generates FPA free commands in two different cases:

- To free an instruction chunk
- To free a buffer when directed for an outbound transfer

When freeing an instruction chunk, NPI_DMA_CONTROL[DWB_ICHK] is the DWB count used.

When freeing a buffer, a DWB count of zero is used when NPI_DMA_CONTROL[DWB_DENB] is clear. If NPI_DMA_CONTROL[DWB_DENB] is set, the following equation specifies the DWB count that is used:

```
(((ptr & 127) + Size + 127) >> 7) + Back
```

which may result in a coherent-memory-bus DWB command for all cache blocks from the start of the buffer up to the block that includes the last byte of the segment. Back, ptr, and Size here come directly from the local pointer that caused the FPA free to be generated.

9.5.7 Host Output Queueing Via the PCI DMA Engine

The PCI DMA engine provides flexible primitives which, when combined with core and host software protocols, can efficiently implement a variety of host output queueing protocols and handle many simultaneous host queues.

We will illustrate with an example. Suppose one million host output queues are required, and each host queue is a simple ring similar to the rings that are managed by the hardware for the two PCI-instruction input ports and two PCI-packet output ports. (This is just one simple example. Many other queue/buffer structures are also possible.) Each ring has a head, tail, and outstanding doorbell count.

Here is how to implement the example one million queues using the hardware PCI DMA engine:

- The core software maintains base and head/tail pointers for each of the one million queues. This information resides in memory locations present in either the L2 cache or the DRAM attached to CN50XX. The core software selects the detailed format of the information.

- The host software must issue “doorbell rings” to indicate space availability. For the four hardware-managed packet output queues, a doorbell ring is a direct PCI CSR write (to PCI_PKT_CREDITS0/1) by the host. But since the doorbells/heads/tails for the one million queues are maintained by core software in this case, their doorbell ring is instead a special work-queue entry or packet received by the core software from the host. The host software can create/submit the special work-queue entry or packet accomplishing the doorbell ring via PCI instruction input as described in Section 9.3. Core and host software selects the detailed format of the doorbell rings.
- In addition to the one million rings, the core and host software also maintain two additional rings for control. These control rings are tied to PCI_DMA_CNT0/PCI_DMA_CNT1 (and indirectly to PCI_INT_SUM[DCNT0, DCNT1, DTIME0, DTIME1]), as selected by the C and CA bits in the PCI DMA instruction header.

  As with the one million rings, the base and head/tail pointers for each of the two control queues is maintained in memory, and “doorbell rings” for them would arrive via PCI instruction input.

- When the core software wishes to send a packet to one of the one million rings, it will write the packet data into the appropriate host memory ring and also write a control packet to one of the two available control rings. This control packet includes ring identifier and packet size fields and is the indication to host software that the packet data arrived at the identified ring.

  Both the packet and the control packet can be transferred via a single outbound PCI DMA Instruction.

To construct the PCI DMA Instruction, the core software first creates PCI DMA local pointers and PCI DMA host pointers (PCI components) to transfer the packet data from L2/DRAM into the host memory. The core software then appends additional PCI DMA local pointers and PCI DMA host pointers (PCI components) to the lists to transfer the control packet into the host memory after the packet data.

The core software consults the base and head/tail pointers that it maintains in L2/DRAM to determine the locations where both the packet data and the control packet data should be placed in the host memory. These addresses are supplied in the PCI DMA Instruction PCI components of the outbound transfer.

The core software can set the CA bit in the PCI DMA Instruction HDR to indicate the arrival of the control packet. (Of course, it must also set the C bit correctly to select which of the two control queues was used.)

9.6 PCI Memory Space Loads/Stores to BAR1/2

9.6.1 Referencing L2/DRAM With CN50XX as a PCI Target

CN50XX may retry, delay, complete, or disconnect a BAR1/2 PCI read (Memory Read, Memory Read Multiple, or Memory Read Line). CN50XX may retry these reads if it has already accepted a delayed read request that does not match the current read, or if the delayed read has not returned sufficiently quickly. Otherwise, until part of the data from the outstanding delayed read has been returned or configuration controlled latency time-out occurs, all reads will be retried.

PCI read requests do not include a length. CN50XX contains a 512 byte prefetch buffer to guarantee that long read bursts can be satisfied quickly. CN50XX launches the memory loads to fill the prefetch buffer at the moment that it accepts a new request for prefetch. (CN50XX accepts a new request for prefetch whenever there is not already a pending prefetch. The request for prefetch may be delayed or may not

be.) The amount of data CN50XX prefetches to service the request depends on the command and CSR configuration (PCI_READ_CMD_*[PREFETCH]). (CN50XX also puts the prefetched data into the L2 cache according to the specified attribute.) Separately for each PCI command, (PCI_READ_CMD_*[PREFETCH]) selects the prefetch depth from among these options:

- Only the requested 64 or 32-bit word (and disconnect after a single data phase)
- No more than one (128-byte) cache block ahead (i.e. fetch to the next cache block boundary
- No more than two (128-byte) cache blocks ahead
- No more than three (128-byte) cache blocks ahead
- No more than four (128-byte) cache blocks ahead

CN50XX supplies the data for the PCI read from the prefetch buffer, and also prefetches ahead as indicated by (PCI_READ_CMD_*[PREFETCH]). If the prefetch buffer drains so that data cannot be provided to service the read in the minimum "target subsequent latency" of eight PCI cycles (i.e. if the prefetch buffer "underflows"), CN50XX disconnects. To reduce the likelihood of these underflow disconnects (fewer underflows means less prefetched data and (possibly) lower latency since CN50XX flushes the prefetch buffer after an underflow), CN50XX has one (PCI_READ_CMD_*[MIN_DATA]) variable per each of the three input commands. This configuration variable specifies the minimum amount of data in the prefetch buffer before CN50XX responds with the first word. CN50XX flushes the prefetch buffer whenever either:

a. It returns all the data for the original request, or
b. It returns all the data for the delayed request, or
c. It disconnects, or
d. The delayed read request times out.

All PCI writes (Memory Write and Memory Write and Invalidate) are treated identically in BAR1/2. CN50XX may retry these requests when sufficient write buffering is not available, but generally, CN50XX must service these write requests. (For example, CN50XX must service PCI writes when the prefetch buffer is active with a delayed read.) CN50XX accumulates sequential writes into aligned cache blocks before sending the writes to memory.

NOTE: Full cache-block writes are efficient since they avoid address overhead and memory/cache reads. CN50XX gets the performance improvement from them for both the Memory Write and Memory Write and Invalidate commands. Also, if the cache attribute specifies it, full cache-block writes can be written-through to memory. Partial cache block writes always enter the L2 cache, independent of the cache attribute.

Once the CN50XX hardware accumulates a cache block to write, it sends it to L2/memory. CN50XX also sends unaccumulated stores to memory when the PCI store transaction terminates. CN50XX waits for the commit of each cache block individually before it writes the next cache block to L2/memory. This is necessary to guarantee that the cores observe the writes in the same order as they occurred on the PCI bus. (CN50XX also includes NPI_CTL_STATUS[WAIT_COM] for better performance. When this mode-bit is clear, PCI cache block stores do not wait for prior cache block stores to commit.)

9.7 CN50XX PCI Internal Arbiter

CN50XX contains a PCI arbiter that can be used when CN50XX is a PCI host. It supports up to four external devices, and bus parking on either the last device or a fixed device. It is configured via NPI_PCI_INT_ARB_CFG.

9.8 CN50XX PCI MSI Support

CN50XX can support MSI as a device.

CN50XX also has a register in the BAR0 space (PCI_MSI_RCV) to receive message-signalled interrupts when functioning as a PCI host. Write operations to PCI_MSI_RCV[INT] cause the bit NPI_MSI_RCV[INT_VEC<PCI_MSI_RCV[INT]>] to be set. There are four CN50XX-internal interrupt wires in the central interrupt unit emanating from NPI_MSI_RCV, one for each of the following uses:

- when any of NPI_MSI_RCV[INT_VEC<15:0>] are set
- when any of NPI_MSI_RCV[INT_VEC<31:16>] are set
- when any of NPI_MSI_RCV[INT_VEC<47:32>] are set
- when any of NPI_MSI_RCV[INT_VEC<63:48>] are set.

Refer to Chapter 11, the CIU chapter.

9.9 Endian Swapping

For all cases, assume the 32-bit data on the PCI bus is represented as:

[E-F-G-H] (cycle 0)
[A-B-C-D] (cycle 1)

where each letter represents a byte. Four bytes represent 32-bits. A and E are the most-significant bytes, D and H are the least-significant bytes.

In the case of 32-bit PCI bus transfers, [E-F-G-H] is the first 32-bits, corresponding to the case when <2> of the PCI byte address is clear, and [A-B-C-D] is the second 32-bits, corresponding to the case when <2> of the PCI byte address is set.

9.9.1 PASS_THRU MODE (== 0)

No swap by the PCI I/O bus device: [A-B-C-D-E-F-G-H]. CN50XX internal byte order is A-B-C-D-E-F-G-H, from first to last.

An aligned 64-bit load/store references [A-B-C-D-E-F-G-H].

An aligned 32-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[E-F-G-H]	[A-B-C-D]
OFFSET<2:0> == 4 references	[A-B-C-D]	[E-F-G-H]

An aligned 16-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[G-H]	[A-B]
OFFSET<2:0> == 2 references	[E-F]	[C-D]
OFFSET<2:0> == 4 references	[C-D]	[E-F]
OFFSET<2:0> == 6 references	[A-B]	[G-H]

An 8-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[H]	[A]
OFFSET<2:0> == 1 references	[G]	[B]
OFFSET<2:0> == 2 references	[F]	[C]
OFFSET<2:0> == 3 references	[E]	[D]
OFFSET<2:0> == 4 references	[D]	[E]
OFFSET<2:0> == 5 references	[C]	[F]
OFFSET<2:0> == 6 references	[B]	[G]
OFFSET<2:0> == 7 references	[A]	[H]

9.9.2 64b_BYTE_SWAP Mode (== 1)

Full swap by the PCI I/O bus device, [H-G-F-E-D-C-B-A], CN50XX internal byte order is H-G-F-E-D-C-B-A, from first to last.

An aligned 64-bit load/store references [H-G-F-E-D-C-B-A].

An aligned 32-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[D-C-B-A]	[H-G-F-E]
OFFSET<2:0> == 4 references	[H-G-F-E]	[D-C-B-A]

An aligned 16-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[B-A]	[H-G]
OFFSET<2:0> == 2 references	[D-C]	[F-E]
OFFSET<2:0> == 4 references	[F-E]	[D-C]
OFFSET<2:0> == 6 references	[H-G]	[B-A]

An 8-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[A]	[H]
OFFSET<2:0> == 1 references	[B]	[G]
OFFSET<2:0> == 2 references	[C]	[F]
OFFSET<2:0> == 3 references	[D]	[E]
OFFSET<2:0> == 4 references	[E]	[D]
OFFSET<2:0> == 5 references	[F]	[C]
OFFSET<2:0> == 6 references	[G]	[B]
OFFSET<2:0> == 7 references	[H]	[A]

9.9.3 32b_BYTE_SWAP Mode (== 2)

No swap by the PCI I/O Bus device: [D-C-B-A-H-G-F-E]. CN50XX internal byte order is D-C-B-A-H-G-F-E, from first to last.

An aligned 64-bit load/store references [D-C-B-A-H-G-F-E].

An aligned 32-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[H-G-F-E]	[D-C-B-A]
OFFSET<2:0> == 4 references	[D-C-B-A]	[H-G-F-E]

An aligned 16-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[F-E]	[D-C]
OFFSET<2:0> == 2 references	[H-G]	[B-A]
OFFSET<2:0> == 4 references	[B-A]	[H-G]
OFFSET<2:0> == 6 references	[D-C]	[F-E]

An 8-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[E]	[D]
OFFSET<2:0> == 1 references	[F]	[C]
OFFSET<2:0> == 2 references	[G]	[B]
OFFSET<2:0> == 3 references	[H]	[A]
OFFSET<2:0> == 4 references	[A]	[H]
OFFSET<2:0> == 5 references	[B]	[G]
OFFSET<2:0> == 6 references	[C]	[F]
OFFSET<2:0> == 7 references	[D]	[E]

9.9.4 32b_LW_SWAP Mode (== 3)

No swap by the PCI I/O Bus device: [E-F-G-H-A-B-C-D]. CN50XX internal byte order is E-F-G-H-A-B-C-D, from first to last.

An aligned 64-bit load/store references [E-F-G-H-A-B-C-D].

An aligned 32-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[A-B-C-D]	[E-F-G-H]
OFFSET<2:0> == 4 references	[E-F-G-H]	[A-B-C-D]

An aligned 16-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[C-D]	[E-F]
OFFSET<2:0> == 2 references	[A-B]	[G-H]
OFFSET<2:0> == 4 references	[G-H]	[A-B]
OFFSET<2:0> == 6 references	[E-F]	[C-D]

An 8-bit load/store:

	Core in LE mode	Core in BE mode
OFFSET<2:0> == 0 references	[D]	[E]
OFFSET<2:0> == 1 references	[C]	[F]
OFFSET<2:0> == 2 references	[B]	[G]
OFFSET<2:0> == 3 references	[A]	[H]
OFFSET<2:0> == 4 references	[H]	[A]
OFFSET<2:0> == 5 references	[G]	[B]
OFFSET<2:0> == 6 references	[F]	[C]
OFFSET<2:0> == 7 references	[E]	[D]

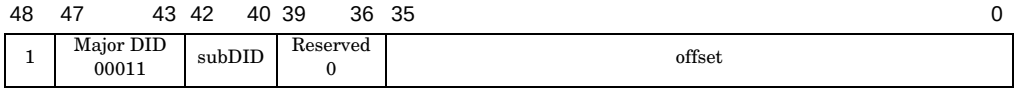
9.10 PC Bus Operations

Operations by the cores to the PCI unit generate PCI read/write commands.

Cores access PCI bus addresses with physical addresses of the following format:

9.10.1 Load/Store Operations

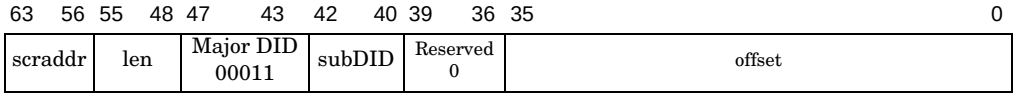
Physical Address Format for Loads/Stores



- **Major DID = 3** - Device ID of the PCI block.
- **subDID** - The Address space for SubDID values 1-6 to the PCI block are defined below.
- **Offset** - Used differently depending on SubDID value.

9.10.2 IOBDMA Operations

IOBDMA Addressing



- **scraddr** - Defined in "cnMIPS™ Cores" on page 143.
- **len** - Indicates the length of the IOBDMA in 64-bit words. Can legally range from 1 to 128. Must be 0x1 for subDIDs 0 and 7.
- **subDID** - Selects the space from among those described in Sections 9.10.3, 9.10.6, and 9.10.7.

NOTE: IOBDMAs are not supported to PCI Config / IACK / Special space (subDID==1), PCI I/O space (subDID==2), nor to the PCI Configuration registers (accessible via a portion of the CSR subDID==7 space).

9.10.3 RSL Access Space (SubDID == 0)

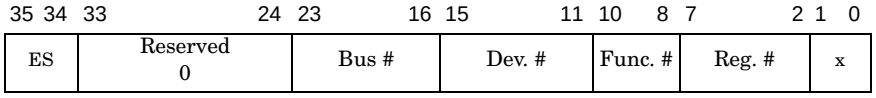
Cores can access CN50XX's various RSL-type CSRs via loads/stores to this subDID==0 space. Any size load is allowed, but only 64-bit stores are allowed to this space. IOBDMAs to this space must be of length 1 (8B).

9.10.4 PCI Config / IACK / Special Space (SubDID == 1)

Cores can issue PCI CONFIG, PCI IACK, and PCI SPECIAL commands via loads/stores to this subdid = 1 space. IOBDMA and 64-bit load/store operations are not allowed to this space.

The following is the format of the OFFSET field.

OFFSET Field Format



- **Bus #** = selects a bus in the system.
- **Dev. #** = selects a device on the bus.
- **Func. #** = selects a function in the device.
- **Reg. #** = selects a register in the configuration space of the intended target.

The following is how the hardware decodes the loads/stores to this region:

```
If OFFSET<23:3> == 0x1FE0 (i.e. Bus = 0, Dev = 1's, Func = 1's, Reg = 0 or 1)
```

If the operation is a store:

The store must be an aligned 32-bit store. 8, 16, and 64-bit stores are illegal.

Generate a PCI Special Cycle. The value on the bus during the (single) data phase is the data from the 32-bit store.

else if the operation is a load:

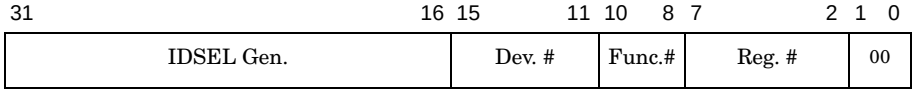
The load may be either 32, 16, or 8-bit naturally aligned. 64-bit loads are illegal.

Generate a PCI IACK Cycle. The value on the bus during the (single) data phase is returned to the load instruction. The mask value presented on the PCI bus corresponds to the lower address bits, ES, and the size of the load.

```
else if OFFSET<23:16> == 0 (i.e. Bus == 0)
```

Generate a PCI config type 0 transaction. The address on the PCI bus is generated according to the “PCI Type 1 to Type 0 Configuration Address” in Figure 2-11 and Table 2-5 IDSEL Generation in “PCI Addendum to the PCI Local Bus Specification”:

Type 0 Configuration Address



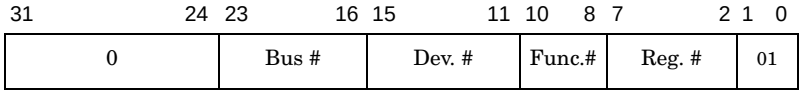
IDSEL Gen. is all zeroes if OFFSET<15> (i.e. device number<4>) is set, otherwise, it is a decode of OFFSET<14:11> (i.e. device number <3:0>)

32-bit, 16-bit, and 8-bit aligned loads/stores are allowed. The mask value presented on the PCI bus corresponds to the lower address bits, ES, and the size of the load. 64-bit accesses are illegal.

```
else (i.e. if OFFSET<23:16> != 0, Bus != 0)
```

Generate a PCI config type 1 transaction. The address on the PCI bus is generated as follows:

Type 1 Configuration Address

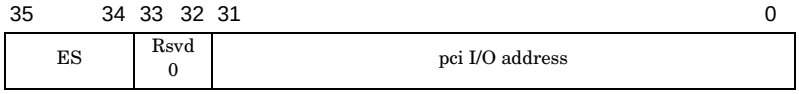


32-bit, 16-bit, and 8-bit aligned loads/stores are allowed. The mask value presented on the PCI bus corresponds to the lower address bits, ES, and the size of the load. 64-bit accesses are illegal.

9.10.5 PCI I/O Space (SubDID == 2)

Cores can issue PCI I/O Loads/Stores in this space. OFFSET<31:0> is the I/O address presented on the PCI bus. IOBDMA and 64-bit load/store operations are not allowed are not allowed to this space.

OFFSET Field Format in I/O space



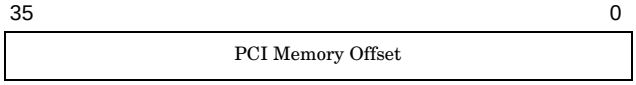
- **ES** - Indicates the endian-swap mode.

9.10.6 Memory Space (SubDID == 3, 4, 5, 6)

Cores can issue PCI Memory Space Load/Store/IOBDMA operations in this space (i.e. Memory Read Line/Block and Memory Write Commands). OFFSET<35:0> is the Memory space address that is presented on the PCI bus. The four different SubDID possibilities select from among four possible CSR values (NPI_MEM_ACCESS_SUBID(3..6)[*]) for each of:

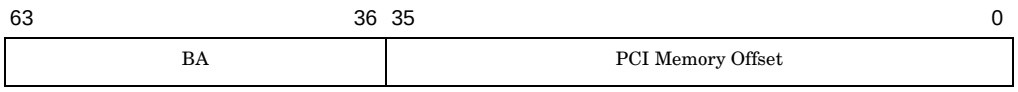
- **ESR** - endian-swap mode for read operations
- **ESW** - endian-swap mode for write operations
- **NSR** - No Snoop for read operations
- **NSW** - No Snoop for write operations
- **ROR** - Relaxed Ordering for read operations
- **ROW** - Relaxed Ordering for write operations
- **BA** - The PCI address bits <63:36>

OFFSET DECODE in MEMORY space



The 64-bit address presented on the PCI bus for these references is:

64-bit Form PCI Memory Offset



CN50XX cores perform no-store merging on stores to the PCI memory space. They pass individual PCI memory-space stores on to the PCI hardware. The PCI hardware can combine sequential stores in the same memory space and send them out as a single (burst) PCI transaction of up to 256 bytes, with the maximum size selected by NPI_CTL_STATUS[MAX_WORD]. The combining hardware is very simple and limited by these constraints:

- Both cores share a single active combining buffer, and CN50XX combines stores originating from any core. A single combined store may contain store data originating from different cores.
- CN50XX permits combining only when the new store references bytes within the aligned 64-bit location, sequentially, after the one referenced by the last store received by the PCI logic.
- CN50XX disables combining and activates the combined store if another store does not arrive within a programmable time period (NPI_CTL_STATUS[TIMER]).
- CN50XX disables combining and activates the previous combined store immediately if a load, IOBDMA, or a store operation that will not combine arrives. CN50XX does the same for load/store operations from the core in the I/O or CONFIG/ACK/SPECIAL spaces.
- CN50XX does not combine store operations and sends out all store operations immediately when combining is disabled. Combining is disabled when NPI_CTL_STATUS[MAX_WORD] = 1.
- CN50XX disables combining and sends out the store when the size of the merged store equals the maximum indicated by NPI_CTL_STATUS[MAX_WORD].

With PCI in this space, CN50XX creates the following commands:

  - Memory Read command to service an ordinary load operation
  - Memory Read or Memory Read Multiple command to service an IOBDMA operation. This depends on the length by comparing against a configuration variable.
  - Memory Write command to service a (possibly combined) store operation.

9.10.7 PCI-Related, NCB-Direct, PCICONFIG, and PCI_NCB CSR Access (SubDID == 7)

Cores can access CN50XX's PCI NCB-direct (see Section 9.15), PCI configuration (see Section 9.13), and PCI_NCB (see Section 9.14.1) registers through this space. IOBDMA operations are allowed to the PCI NCB-direct and PCI_NCB registers, but are not permitted to the PCICONFIG registers. IOBDMA operations to this space must be of length 1 (8B).

9.11 PCI Reset Sequence

The reset sequence for the PCI varies depending on whether the PCI is in host mode or non-host mode. The following subsections describe both situations.

9.11.1 PCI Reset Sequence in Host Mode

In host mode (i.e. when CN50XX's external pin PCI_HOST_MODE=1), CN50XX drives PCI_RST_L with the value selected by CIU_SOFT_PRST[SOFT_PRST]. CIU_SOFT_PRST only affects behavior in PCI host mode. (When CIU_SOFT_PRST[SOFT_PRST] = 1, PCI_RST_L is driven low, to indicate an active reset.)

In host mode, the system typically uses the PCI clocks generated and driven from CN50XX (i.e. CN50XX's external pins PCI_CLK_OUT_*). As CN50XX does not internally use the PCI clocks it generates, it is also possible for the system to generate the PCI clock externally. The selected PCI clock must be connected to CN50XX's PCI_PCLK input pin, and must also be connected to the clock inputs of the other devices on the PCI bus.

CN50XX samples PCI_M66EN at the time of DCOK assertion. At that point, CN50XX's PCI_CLK_OUT_* are stable PCI clocks with the proper frequency. (This assumes that PCI_REF_CLKIN is a 133-MHz reference clock.) CN50XX eventually drives PCI_DEVSEL_L, PCI_STOP_L, and PCI_TRDY_L with the encoding appropriate for the selected frequency and mode.

The reset value for CIU_SOFT_PRST[SOFT_PRST] is 1, so the PCI bus is in reset while CN50XX boots up when CN50XX is a PCI host. CN50XX software must not access any PCI_NCB or PCICONFIG CSRs whenever CIU_SOFT_PRST[SOFT_PRST] = 1 in host mode.

Eventually, PCI-host boot software transitions CIU_SOFT_PRST[SOFT_PRST] from 1 to 0 to take the PCI bus out of reset in host mode. The PCI input clock to CN50XX, PCI_PCLK, must be stable prior to this point. Figure 9–12 shows some events that happen as this point.

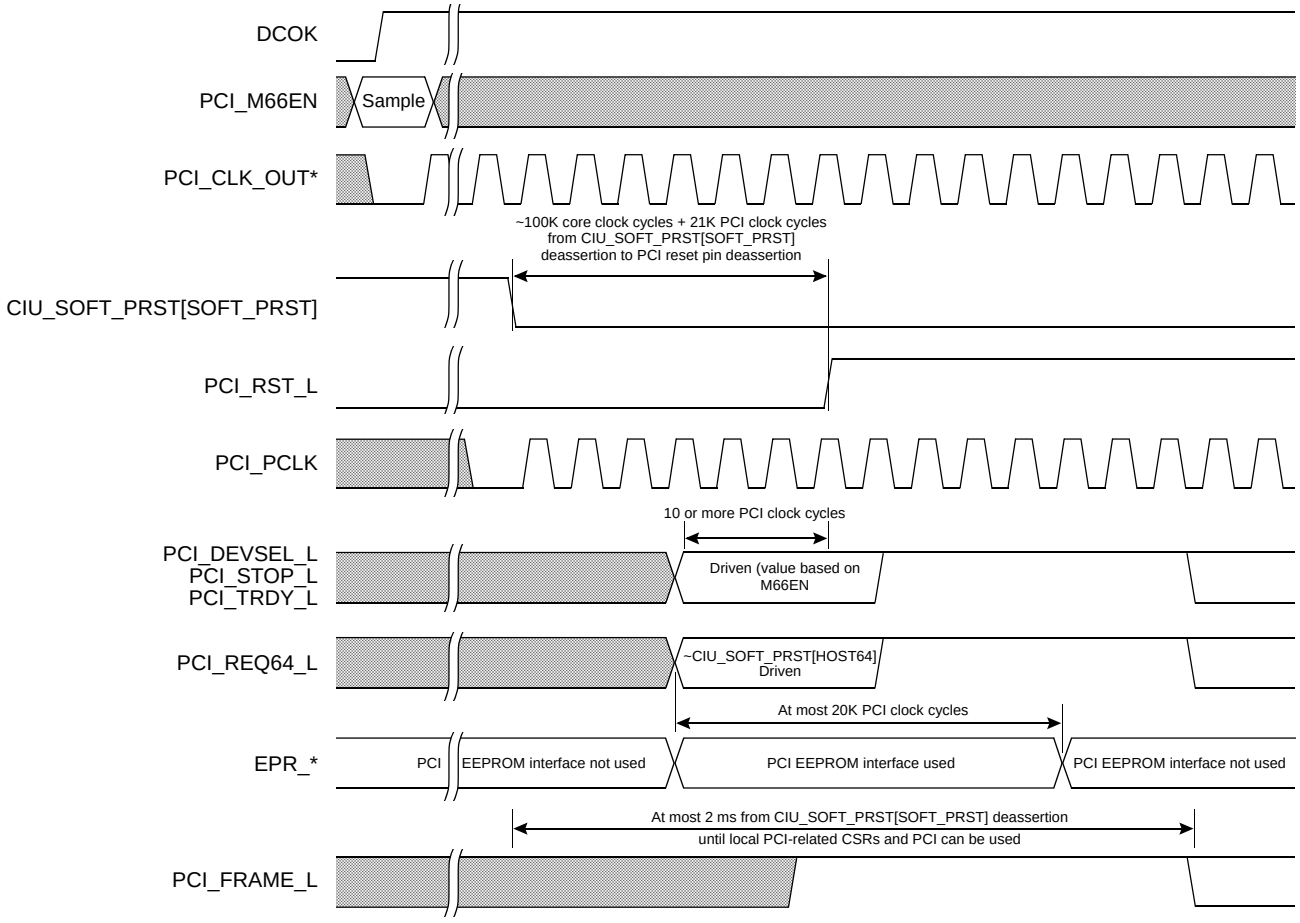


Figure 9–12 PCI Reset Timing in Host Mode

CN50XX does not deassert the PCI reset pin immediately following CIU_SOFT_PRST[SOFT_PRST] deassertion. Instead, CN50XX deasserts PCI_RST_L approximately 16K core-clock cycles plus 15 PCI clock cycles later. Just prior to deasserting PCI_RST_L, CN50XX drives PCI_REQ_L with the value selected by CIU_SOFT_PRST[HOST64], and drives PCI_DEVSEL_L, PCI_STOP_L, and PCI_TRDY_L to the value selected by PCI_M66EN. CN50XX and the other devices on the PCI bus sample the PCI_REQ_L, PCI_DEVSEL_L, PCI_STOP_L, and PCI_TRDY_L at the time that PCI_RST_L deasserts to determine bus frequency, mode, and width.

CN50XX also drives and samples the PCI EEPROM signals near the time that PCI_RST_L deasserts. If an EEPROM is not present, as should be the case when CN50XX is in host mode, the EEPROM load sequence completes quickly.

Following its CIU_SOFT_PRST[SOFT_PRST] deassertion, the PCI-host boot software must continue to avoid accessing CN50XX's NCB_PCI registers for at least ~16K cycles.

Also, it has the following restrictions until after the PCI EEPROM load is complete, which is at most ~16K core-clock cycles + 20K PCI-clock cycles, or at most 2 milliseconds following the CIU_SOFT_PRST[SOFT_PRST] deassertion:

- it must not access the PCI bus
- it must not access any PCICONFIG CSRs
- it must not reassert CIU_SOFT_PRST[SOFT_PRST]

In host mode, whenever CIU_SOFT_PRST[SOFT_PRST] is set, CN50XX tri-states all PCI bus signals until the first PCI bus transaction, except for the following four signals:

- PCI_DEVSEL_L
- PCI_STOP_L
- PCI_TRDY_L
- PCI_REQ64_L

Eventually, PCI-host software may later transition CIU_SOFT_PRST[SOFT_PRST] from 0 to 1 to reassert PCI_RST_L in host mode. Following its CIU_SOFT_PRST[SOFT_PRST] reassertion, PCI-host software should read CIU_SOFT_PRST, and wait for the result to return, before performing any other operations. As mentioned above, no PCI_NCB or PCICONFIG CSRs can be accessed when CIU_SOFT_PRST[SOFT_PRST] is set.

In host mode when CIU_SOFT_PRST[SOFT_PRST] is set, CN50XX resets its logic directly connected to the PCI bus, as well as all PCI_NCB, PCI, and PCICONFIG CSRs. By default, CN50XX resets no more internal logic when CIU_SOFT_PRST[SOFT_PRST] is set. However, if CIU_SOFT_PRST[NPI] is set when CIU_SOFT_PRST[SOFT_PRST] transitions from 0 to 1, logic blocks deeper into CN50XX (NPI and PNI) are also briefly reset. Normally, it should not be necessary to set CIU_SOFT_PRST[NPI], but the option is there. If CIU_SOFT_PRST[NPI] is set when CIU_SOFT_PRST[SOFT_PRST] transitions from 0 to 1, all PCI_* and NPI_* CSRs (of type NCB, PCI_NCB, PCI and PCICONFIG) are reset, as is all PCI-related logic in CN50XX.

9.11.2 PCI Reset Sequence in Non-Host Mode

In non-host mode (i.e. when CN50XX's external pin PCI_HOST_MODE=0), PCI_RST_L has the following characteristics:

- it is a CN50XX input
- it is both the PCI bus reset and CN50XX's chip reset
- it must be driven by the external system.

CIU_SOFT_PRST has no effect in non-host mode.

Also in non-host mode, CN50XX does the following:

- CN50XX typically receives its PCI clock from the remote host. The PCI clock must be connected to CN50XX's PCI input clock (PCI_PCLK).

- CN50XX tristates all PCI bus signals during the entire reset sequence up until the first following PCI bus transaction.

Some of the events in a non-host-mode reset are shown in Figure 9–13.

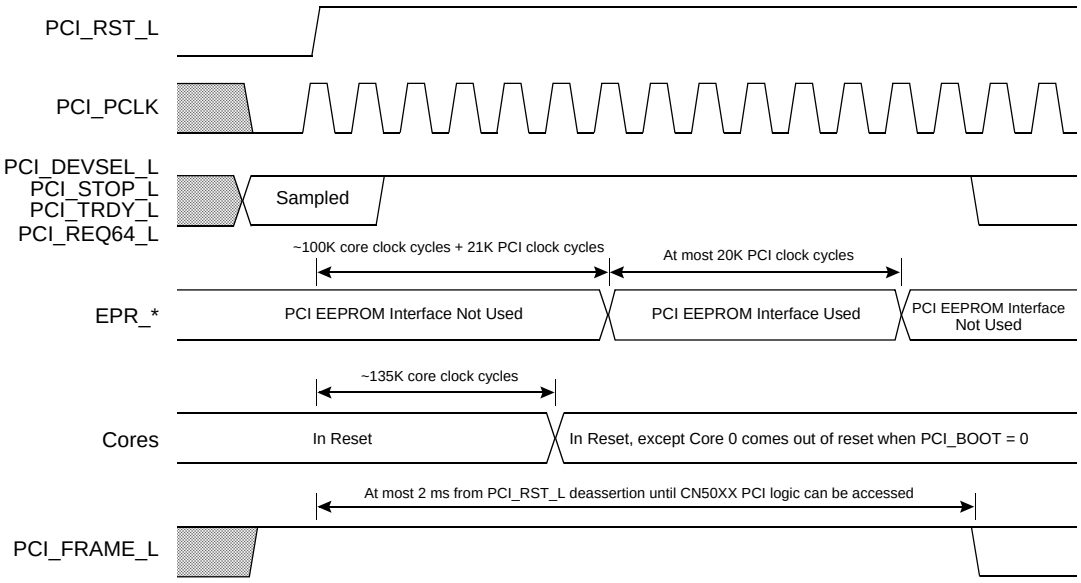


Figure 9–13 PCI Reset Timing in Non-Host Mode

CN50XX samples PCI_DEVSEL_L, PCI_STOP_L, PCI_TRDY_L, and PCI_REQ64_L on the deassertion of PCI_RST_L to determine the bus characteristics.

Immediately following the deassertion of PCI_RST_L in non-host mode, CN50XX locks its PCI DLL/PLLs and performs internal BIST.

Approximately 16K cycles following PCI_RST_L deassertion in non-host mode, CN50XX's internal PCI logic comes out of reset and attempts to read the PCI EEPROM via the EPR_* external pins. If a PCI EEPROM is not present, the read stops soon after it starts. If a PCI EEPROM is present, the read continues.

Approximately 180K cycles following PCI_RST_L deassertion in non-host mode, the remainder of the logic (i.e. non-PCI logic) comes out of reset. At this point, core 0 comes out of reset if CN50XX is not in PCI boot mode (i.e. if PCI_BOOT=0). Core 0 must not access CN50XX's PCICONFIG CSRs, nor the PCI bus, before the PCI EEPROM load is complete. CN50XX's PCI EEPROM load always completes within 20K PCI clocks.

When CN50XX is not in host mode, a remote PCI host must not access CN50XX via the PCI bus until CN50XX completes both the PCI EEPROM load and its internal reset sequence. In all circumstances in non-host mode, CN50XX can be accessed by the remote host 2 ms after reset deasserts.

9.12 PCI Checklist

The following checklist highlights important CN50XX PCI-related configuration problems. The more likely problem areas are at the top of the list.

CSR/Field	Comment
CIU_SOFT_PRST[SOFT_PRST]	When CN50XX is a PCI host (i.e. when PCI_HOST_MODE = 1), this bit controls PCI_RST_L. Refer to Section 9.11.1.
PCI_CFG01	ME and MSAE should always be set. Should normally be written with 0x346.
NPI_PCI_INT_ARB_CFG	When CN50XX is a PCI host, most systems will use CN50XX's internal arbiter. For those that do, the internal arbiter must be enabled before any PCI traffic can occur.
PCI_CFG19	TDOMC must be set to 0x1 in PCI mode. Otherwise, should not change from its reset value. Do not write PCI_CFG19 in PCI mode (0x82000001 is the reset value). MRBCI, MDWE,and MDRE must be 0. MRBCM must be 1.
PCI_BAR1_INDEX(0..31)	If ADDR_V = 0, CN50XX does not service the portion of its BAR1 serviced by this table entry. (CN50XX accepts these invalid writes, discards them, and sets PCI_INT_SUM*[ILL_WR]. CN50XX target aborts these invalid PCI reads, and returns an error response for these invalid PCI-X reads, setting PCI_INT_SUM*[ILL_RD] for both.) Endian-swapping is another typical problem area. END_SWP and ADDR_IDX must also be set up properly. Refer to Section 9.2.2.
PCI_CFG22	Should be written to 0x4FF00. MTTV must be 0x0, FLUSH must be 1, MRV should be 0xFF.
PCI_CFG16	Preferably written to 0x1 to set MLTD. RDSATI, TRTAE, TWTAE, TMAE, and DPPMR must be 0. TILT must not be set to 0x1..0x7.
PCI_CTL_STATUS_2 [BAR2PRES,BAR2_ENB]	It appears that CN50XX's BAR2 does not exist when BAR2PRES clear. CN50XX cannot service BAR2 references when either BAR2PRES or BAR2_ENB are clear (refer to Section 9.2.3). MIO_FUS_DAT3[BAR2_EN] is the reset value for PCI_CTL_STATUS_2[BAR2PRES]. CN50XX accepts writes when BAR2PRES = 1 AND BAR2_ENB = 0, discards them, and sets PCI_INT_SUM*[ILL_WR]. CN50XX target aborts similarly invalid PCI read operations, setting PCI_INT_SUM*[ILL_RD]. Endian-swapping is another typical problem area. BAR2_ESX must be set up properly.
NPI_MEM_ACCESS_SUBID3 NPI_MEM_ACCESS_SUBID4 NPI_MEM_ACCESS_SUBID5 NPI_MEM_ACCESS_SUBID6	BA, ESR, and ESW should be set up properly for CN50XX core accesses to physical addresses 0x1 1B0X XXXX XXXX, 0x1 1C0X XXXX XXXX, 0x1 1D0X XXXX XXXX, and 0x1 1E0X XXXX XXXX, respectively, to translate into PCI bus read/write operations. Endian-swapping is a common problem area. Review your ESR/ESW settings carefully.

CSR/Field	Comment
NPI_NUM_DESC_OUTPUT(0..3) NPI_BASE_ADDR_OUTPUT(0..3) NPI_BUFF_SIZE_OUTPUT(0..3) NPI_OUTPUT_CONTROL NPI_CTL_STATUS [OUT0_ENB,OUT1_ENB, OUT2_ENB,OUT3_ENB] PCI_PKTS_SENT_INT_LEV(0..3) PCI_PKTS_SENT_TIME(0..3)	Configures CN50XX's PCI packet output ports. Endian-swapping is a common problem area. Review your NPI_OUTPUT_CONTROL[ESR_SL*,O*_CSRM,O*_ES] settings (refer to Section 9.4).
NPI_BASE_ADDR_INPUT(0..3) NPI_SIZE_INPUT(0..3) NPI_INPUT_CONTROL(0..3) NPI_PORT32/33_INSTR_HDR NPI_CTL_STATUS [INS0_64B,INS1_64B,INS2_64B ,INS3_64B,INS0_ENB,INS1_EN B,INS2_ENB,INS3_ENB]	Configures CN50XX's PCI packet input ports. Endian-swapping is a common problem area. Review your NPI_INPUT_CONTROL[ESR,USE_CSR,D_ESR] and NPI_PORT32/ 33_INSTR_HDR[PBP] settings and Section 9.3.
NPI_LOWP_IBUFF_SADDR NPI_HIGHP_IBUFF_SADDR NPI_DMA_CONTROL PCI_DMA_INT_LEV(0..1) PCI_DMA_TIME(0..1)	Configures CN50XX's PCI DMA engine. Endian-swapping is a common problem area. Review your NPI_DMA_CONTROL[O_ES,O_MODE] settings and Section 9.5
PCI_READ_CMD_6 PCI_READ_CMD_C PCI_READ_CMD_E	Affects PCI performance when CN50XX services reads to its BAR1/BAR2 (refer to Section 9.6.1). The recommended values are 0x22, 0x33, and 0x33 for PCI_READ_CMD_6, PCI_READ_CMD_C, and PCI_READ_CMD_E, respectively. Note that these values differ from their reset values.
NPI_PCI_READ_CMD	Affects PCI performance by selecting a memory read or a memory read multiple command. Recommended value depends on the system.
NPI_PCI_BURST_SIZE	Affects PCI performance since it limits the size of individual read/write bursts mastered by CN50XX. The recommended value is 0x2040. Note this is not the reset value.
NPI_CTL_STATUS [TIMER,MAX_WORD]	Affects PCI performance by changing the way that CN50XX merges core-generated stores, (refer to Section 9.10.6).
PCI_READ_TIMEOUT	Should be written to enable the timeout. The recommended value is 0x80010000. Note this is not the reset value.
PCI_CFG03	LT and CLS may be updated. 64 may be a reasonable LT value (write 0x4000 + CLS). CALLS does not affect CN50XX behavior.
PCI_CFG15	Need not be written.
PCI_CFG17, PCI_CFG18, PCI_CFG20, PCI_CFG21	Should not be written.

9.13 PCI Configuration Registers

The PCI configuration registers are of type PCICONFIG, which can be accessed directly from the PCI bus and from the I/O bus.

- Each PCICONFIG register has both a PCI bus address and a CN50XX-internal address.
  - Cores can access the PCICONFIG registers via the CN50XX-internal address.
  - Remote PCI devices/hosts can access PCICONFIG registers with a PCI config space read or write that targets CN50XX.
- All PCI config registers are 32-bits. 32-bit load/store operations must be used by cores to access the PCICONFIG registers.

Table 9–3 PCI Configuration Registers

Register	Address (Little-Endian) (CN50XX internal)	Address (Big-Endian) (CN50XX internal)	Address (PCI Config Space)	CSR Type[1]	Detailed Description
PCI_CFG00	0x00011F0000001800	0x00011F0000001804	0x0000000000000000	PCICONFIG	See page 404
PCI_CFG01	0x00011F0000001804	0x00011F0000001800	0x0000000000000004	PCICONFIG	See page 404
PCI_CFG02	0x00011F0000001808	0x00011F000000180C	0x0000000000000008	PCICONFIG	See page 405
PCI_CFG03	0x00011F000000180C	0x00011F0000001808	0x000000000000000C	PCICONFIG	See page 405
PCI_CFG04	0x00011F0000001810	0x00011F0000001814	0x0000000000000010	PCICONFIG	See page 405
PCI_CFG05	0x00011F0000001814	0x00011F0000001810	0x0000000000000014	PCICONFIG	See page 405
PCI_CFG06	0x00011F0000001818	0x00011F000000181C	0x0000000000000018	PCICONFIG	See page 406
PCI_CFG07	0x00011F000000181C	0x00011F0000001818	0x000000000000001C	PCICONFIG	See page 406
PCI_CFG08	0x00011F0000001820	0x00011F0000001824	0x0000000000000020	PCICONFIG	See page 406
PCI_CFG09	0x00011F0000001824	0x00011F0000001820	0x0000000000000024	PCICONFIG	See page 406
PCI_CFG10	0x00011F0000001828	0x00011F000000182C	0x0000000000000028	PCICONFIG	See page 406
PCI_CFG11	0x00011F000000182C	0x00011F0000001828	0x000000000000002C	PCICONFIG	See page 407
PCI_CFG12	0x00011F0000001830	0x00011F0000001834	0x0000000000000030	PCICONFIG	See page 407
PCI_CFG13	0x00011F0000001834	0x00011F0000001830	0x0000000000000034	PCICONFIG	See page 407
PCI_CFG15	0x00011F000000183C	0x00011F0000001838	0x000000000000003C	PCICONFIG	See page 407
PCI_CFG16	0x00011F0000001840	0x00011F0000001844	0x0000000000000040	PCICONFIG	See page 408
PCI_CFG17	0x00011F0000001844	0x00011F0000001840	0x0000000000000044	PCICONFIG	See page 409
PCI_CFG18	0x00011F0000001848	0x00011F000000184C	0x0000000000000048	PCICONFIG	See page 409
PCI_CFG19	0x00011F000000184C	0x00011F0000001848	0x000000000000004C	PCICONFIG	See page 410
PCI_CFG20	0x00011F0000001850	0x00011F0000001854	0x0000000000000050	PCICONFIG	See page 411
PCI_CFG21	0x00011F0000001854	0x00011F0000001850	0x0000000000000054	PCICONFIG	See page 411
PCI_CFG22	0x00011F0000001858	0x00011F000000185C	0x0000000000000058	PCICONFIG	See page 412
PCI_CFG58	0x00011F00000018E8	0x00011F00000018EC	0x00000000000000E8	PCICONFIG	See page 413
PCI_CFG59	0x00011F00000018EC	0x00011F00000018E8	0x00000000000000EC	PCICONFIG	See page 413
PCI_CFG60	0x00011F00000018F0	0x00011F00000018F4	0x00000000000000F0	PCICONFIG	See page 414
PCI_CFG61	0x00011F00000018F4	0x00011F00000018F0	0x00000000000000F4	PCICONFIG	See page 414
PCI_CFG62	0x00011F00000018F8	0x00011F00000018FC	0x00000000000000F8	PCICONFIG	See page 414
PCI_CFG63	0x00011F00000018FC	0x00011F00000018F8	0x00000000000000FC	PCICONFIG	See page 414

1. PCICONFIG-type registers can be accessed directly from either the PCI bus or the I/O Bus.

PCI Vendor and Device Register
PCI_CFG00

This register contains the first 32-bits of the PCI configuration space (PCI vendor and device). See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	DEVID	RO	0x70	0x70	Device ID for CN50XX. The possible values are: 0x4 = CN38XX (pass 2) 0x40 = CN58XX 0x90 = CN51XX 0x5 = CN38XX (pass 3) 0x50 = CN54/5/6/7XX 0x20= CN31XX/CN3020 0x70 = CN5020/CN5000F 0x30= CN3010/CN3005 0x80 = CN52XX
<15:0>	VENDID	RO	0x177D	0x177D	Cavium Networks' vendor ID

Command/Status Register
PCI_CFG01

Second 32-bits of PCI configuration space (PCI command/status). See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	DPE	R/W1C	0	0	Detected parity error
<30>	SSE	R/W1C	0	0	Signaled system error
<29>	RMA	R/W1C	0	0	Received master abort
<28>	RTA	R/W1C	0	0	Received target abort
<27>	STA	R/W1C	0	0	Signaled target abort
<26:25>	DEVT	RO	0x1	0x1	DEVSEL# timing
<24>	MDPE	R/W1C	0	0	Master data parity error
<23>	FBB	RO	1	—	Fast back-to-back transactions capable mode: must be set to 1 = PCI mode.
<22>	—	RAZ	—	—	Reserved.
<21>	M66	RO	1	1	66MHz capable
<20>	CLE	RO	1	1	Capabilities list enable
<19>	I_STAT	RO	0	0	When INT*x*# is asserted by CN50XX, this bit is set. When deasserted by CN50XX, this bit is cleared.
<18:11>	—	RAZ	—	—	Reserved.
<10>	I_DIS	R/W	0	0	When set to 1, this bit disables the generation of INT*x*# by CN50XX. When cleared to 0, allows assertion of INT*x*# by CN50XX.
<9>	FBBE	R/W	0	1	Fast back-to-back transaction enable
<8>	SEE	R/W	0	1	System error enable
<7>	ADS	RO	0	0	Address/data stepping. Note that CN50XX does not address/data stepping.
<6>	PEE	R/W	0	1	PERR# enable
<5>	VPS	RO	0	0	VGA palette snooping
<4>	MWICE	R/W	0	0	Memory Write & Invalidate command enable
<3>	SCSE	RO	0	0	Special cycle snooping enable
<2>	ME	R/W	0	1	Master enable. Must be set for CN50XX to master a PCI transaction. This should always be set any time that CN50XX is connected to a PCI bus.
<1>	MSAE	R/W	0	1	Memory space access enable. Must be set to receive a PCI memory space transaction. This must always be set any time that CN50XX is connected to a PCI bus.
<0>	ISAE	RO	0	0	I/O space access enable. This bit must never be set. It is read-only and CN50XX does not respond to I/O-space accesses.

Class Code/Revision ID Register
PCI_CFG02

This register contains the third 32-bits of the PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:8>	CC	RO	0x0B3000	0x0B3000	Class code (network encryption/decryption class). Possible values: 0x100000 = CN38XX (pass 2) and CN31XX 0x0B3000 = CN38XX (pass 3), CN30XX, CN58XX, and CN5020 0x0B = CN56XX [BCC]
<7:0>	RID	RO	0x0	0x0	Revision ID.

BIST, HEADER Type, Latency Timer, Line Size Register
PCI_CFG03

Fourth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	BCAP	RO	0	0	BIST capable
<30>	BRB	R/W	0	0	BIST request/busy-bit. CN50XX does not support PCI BIST, therefore this bit should remain 0.
<29:28>	—	RAZ	—	—	Reserved
<27:24>	BCOD	RO	0x0	0x0	BIST code
<23:16>	HT	RO	0x0	0x0	Header type (type 0)
<15:8>	LT	R/W	0x0	0x0	Latency timer
<7:0>	CLS	R/W	0x0	—	Cache line size

Base Address Register 0 - Low
PCI_CFG04

Fifth 32-bits of PCI configuration space. Description of BAR0: 4KB, 64-bit, prefetchable memory space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:12>	LBASE	R/W	0x0	—	Base address[31:12]. User may define the base address. Base address[30:12] is read as 0x0 if PCI_CTL_STATUS_2[BB0] is set to 1.
<11:4>	LBASEZ	RO	0x0	0x0	Base address[11:4] (read as 0x0 to imply 4KB of space).
<3>	PF	RO	1	1	Prefetchable space: 1 = prefetchable, 0 = not prefetchable
<2:1>	TYP	RO	0x2	0x2	Type: 0x0 = 32b, 0x1 = below 1MB, **0x2 = 64b**. 0x3 = reserved
<0>	MSPC	RO	0	0	Memory space indicator

Base Address Register 0 - High
PCI_CFG05

Sixth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	HBASE	R/W	0x0	—	Base address[63:32]

Base Address Register 1 - Low
PCI_CFG06

Seventh 32-bits of PCI configuration space. Description of BAR1: 128MB, 64-bit, prefetchable memory space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:27>	LBASE	R/W	0x0	—	Base address[31:27]. User may define the base address. Base address[29:27] is read as 000 if PCI_CTL_STATUS_2[BB1] is set to 1. Base address[30] is read as 0 if both PCI_CTL_STATUS_2[BB1] and PCI_CTL_STATUS_2[BB1_SIZE] are set to 1s.
<26:4>	LBASEZ	RO	0x0	0x0	Base address[26:4] (read as 0x0 to imply 128 MB of space).
<3>	PF	RO	1	1	Prefetchable space: 1 = prefetchable, 0 = not prefetchable
<2:1>	TYP	RO	0x2	0x2	Type: 0x0 = 32b, 0x1 = below 1MB, **0x2 = 64b**. 0x3 = reserved
<0>	MSPC	RO	0	0	Memory space indicator

Base Address Register 1 - High
PCI_CFG07

Eighth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	HBASE	R/W	0x0	—	Base Address[63:32]

Base Address Register 2 - Low
PCI_CFG08

Ninth 32-bits of PCI configuration space. Description of BAR2: 2^{39} (512GB), 64-bit, prefetchable memory space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:4>	LBASEZ	RO	0x0	0x0	Base address[31:4] (read as 0x0).
<3>	PF	RO	1	1	Prefetchable space: 1 = prefetchable, 0 = not prefetchable
<2:1>	TYP	RO	0x2	0x2	Type: 0x0 = 32b, 0x1 = below 1MB, **0x2 = 64b**. 0x3 = reserved
<0>	MSPC	RO	0	0	Memory space indicator

Base Address Register 2 - High Register
PCI_CFG09

Tenth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:7>	HBASE	R/W	0x0	—	Base address[63:39]
<6:0>	HBASEZ	RO	0x0	0x0	Base address[38:31] (read as 0x0)

Card Bus CIS Pointer Register
PCI_CFG10

Eleventh 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	CISP	RO	0x0	0x0	CardBus CIS pointer (unused)

SubSystem ID/Subsystem Vendor ID Register
PCI_CFG11

Twelfth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	SSID	RO	0x1	0x1	Subsystem ID
<15:0>	SSVID	RO	0x177D	0x177D	Subsystem vendor ID

Expansion ROM Base Address Register
PCI_CFG12

Thirteenth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	ERBAR	R/W	0x0	—	Expansion ROM base address <31:16>. 64KB in size.
<15:11>	ERBARZ	RO	0x0	0x0	Expansion ROM base address (read as 0)
<10:1>	—	RAZ	—	—	Reserved
<0>	ERBAR_EN	R/W	0	0	Expansion ROM address decode enable.

Capabilities Pointer Register
PCI_CFG13

Fourteenth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:8>	—	RAZ	—	—	Reserved
<7:0>	CP	RO	0xE0	0xE0	Capabilities pointer

Interrupt/Arbitration/Latency Register
PCI_CFG15

Sixteenth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:24>	ML	RO	0x40	0x40	Maximum latency
<23:16>	MG	RO	0x40	0x40	Minimum grant
<15:8>	INTA	RO	0x1	0x1	Interrupt pin (INTA#)
<7:0>	IL	R/W	0x0	—	Interrupt line

Target Implementation Register
PCI_CFG16

Seventeenth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	TRDNPR	R/W1C	0	0	Target read delayed transaction for I/O and nonprefetchable regions discarded.
<30>	TRDARD	R/W1C	0	0	Target read delayed transaction for all regions discarded.
<29>	RDSATI	R/W	0	0	Target (I/O and memory) read delayed/split at time-out/immediately (default time-out). **NOTE:** CN50XX requires that this bit be 0.
<28>	TRDRS	R/W	0	0	Target (I/O and memory) read delayed/split or retry select (of the application interface is not ready) 0 = Delayed-split transaction 1 = Retry transaction (always immediate retry, no AT_REQ to application).
<27>	TRTAE	R/W	0	0	Target (I/O and memory) read target abort enable (if application interface is not ready at the latency time-out). **NOTE:** CN50XX as target will never target-abort, therefore this bit should never be set.
<26>	TWSEI	R/W	0	0	Target (I/O) write-split enable (at time-out/immediately; default time-out)
<25>	TWSEN	R/W	0	0	Target (I/O) write-split enable (if the application interface is not ready)
<24>	TWTAE	R/W	0	0	Target (I/O and memory) write target abort enable (if the application interface is not ready at the start of the cycle). **NOTE:** CN50XX as target will never split transactions, therefore this bit should never be set.
<23>	TMAE	R/W	0	0	Target (read/write) master abort enable; check at the start of each transaction. **NOTE:** This bit can be used to force a Master Abort when CN50XX is acting as the intended target device.
<22:20>	TSLTE	R/W	0x0	0x0	Target subsequent (2nd-last) latency time-out enable valid range: [1..7] and 0=8.
<19:16>	TILT	R/W	0x0	0x0	Target initial (1st data) latency time-out in PCI mode valid range: [8..15] and 0=16.
<15:4>	PBE	R/W	0x0	0x0	Programmable boundary enable to disconnect/prefetch for target burst-read cycles to prefetchable region in PCI. A value of 1 indicates end of boundary (64KB down to 16 bytes).
<3>	DPPMR	R/W	0	0	Disconnect/prefetch to prefetchable memory regions enable. Prefetchable memory regions are always disconnected on a region boundary. Non-prefetchable regions for PCI are always disconnected on the first transfer. **NOTE:** CN50XX as target will never target-disconnect, therefore this bit should never be set.
<2>	—	RAZ	—	—	Reserved
<1>	TSWC	R/W	0	0	Target split write control: 0 = Blocks all requests except PMW 1 = Blocks all requests including PMW until split completion occurs.
<0>	MLTD	R/W	0	1	Master latency timer disable. Cavium recommends that this bit be set.

Target Split-Completion Message Enable Register PCI_CFG17

Eighteenth 32-bits of PCI configuration space. This register is for debug purposes only. It should only be written with all 0s. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	TSCME	R/W1C	0	—	Target split-completion message enable. [31:30]: 00 [29]: Split-completion error indication [28]: 0 [27:20]: Split-completion message index [19:0]: 0x00000 **NOTE:** This register is intended for debug use only. Cavium recommends that only 0s be written to this register.

Target Delayed/Split Request Pending Sequences Register PCI_CFG18

Nineteenth 32-bits of PCI configuration space. This register is for debug purposes only. It should only be written with all 0s. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	TDSRPS	R/W1C	0x0	0x0	Target delayed/split request pending sequences. The application uses this address to remove a pending split sequence from the target queue by clearing the appropriate bit. For example, clearing bit <14> clears the pending sequence #14. An application or configuration write to this address can clear this register. **NOTE:** This register is intended for debug use only. Writes to this register must only be all 0s.

Target Delayed/Split Request Pending Sequences Register PCI_CFG19

Twentieth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	MRBCM	R/W	1	1	Master request (memory read) byte count/byte enable select. 0 = Byte enables valid. In PCI mode, a burst transaction cannot be performed using Memory Read command=4'h6. 1 = DWORD byte-count valid (default). In PCI mode, the memory-read byte enables are automatically generated by the core. **NOTE:** This bit must always be 1 for proper operation.
<30>	MRBCI	R/W	0	0	Master request (I/O and CR cycles) byte count/byte enable select. 0 = Byte enables valid (default) 1 = DWORD byte count valid **NOTE:** This bit must always be 0 for proper operation (in support of Type0/1 CfgWr Space accesses which require byte-enable generation directly from a read mask).
<29>	MDWE	R/W	0	0	Master (retry) deferred write enable (allows read requests to pass). 0 = New read requests are not accepted until the current write cycle completes. [Reads cannot pass writes] 1 = New read requests are accepted, even when there is a write cycle pending [Reads can pass writes]. **NOTE:** This bit must always be 0 for proper operation.
<28>	MDRE	R/W	0	0	Master (retry) deferred read enable (allows read/write requests to pass). 0 = New read/write requests are NOT accepted until the current read cycle completes. [Read/write requests CANNOT pass reads] 1 = New read/write requests are accepted, even when there is a read cycle pending. Read/write requests CAN pass reads. **NOTE:** This bit must always be 0 for proper operation.
<27>	MDRIMC	R/W	0	0	Master I/O deferred/split request outstanding maximum count. 0 = value in MDRRMC 1 = 1.
<26:24>	MDRRMC	R/W	0x2	0x2	Master deferred read request outstanding max count (PCI only). 000 = Max SAC Cycles: 8, Max DAC Cycles: 4 001 = 1, 0 010 = 2, 1 011 = 3, 1 100 = 4, 2 101 = 5, 2 110 = 6, 3 111 = 7, 3 **NOTE:** For example, if these bits are programmed to 100, the core can support two DAC cycles, four SAC cycles or a combination of one DAC and two SAC cycles.
<23:16>	TMES	RO	0	0	Target/master error sequence number.
<15>	TECI	RO	0	0	Target error command indication: 0 = delayed/split, 1 = others
<14>	TMEI	RO	0	0	Target/master error indication: 0 = target, 1 = master
<13>	TMSE	R/W1C	0	0	Target/master system error. This bit is set whenever ATM_SERR_O is active.
<12>	TMDPES	R/W1C	0	0	Target/master data PERR# error status. This bit is set whenever ATM_DATA_PERR_O is active.
<11>	TMAPES	R/W1C	0	0	Target/master address PERR# error status. This bit is set whenever ATM_ADDR_PERR_O is active.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<10:9>	—	RAZ	—	—	Reserved
<8>	TIBCD	R/W1C	0	0	Target illegal I/O DWORD byte combinations detected.
<7>	TIBDE	R/W	0	0	Target illegal I/O DWORD byte detection enable
<6>	—	RAZ	—	—	Reserved
<5>	TIDOMC	R/W	0	0	Target I/O delayed/split request outstanding maximum count. 0 = TDOMC[4:0] 1 = 1
<4:0>	TDOMC	R/W	0x1	0x1	Target delayed/split request outstanding maximum count. Values = 0x1–0xFF, and 0 = 0x20. **NOTE:** If you program these bits beyond the designed maximum outstanding count, then the designed maximum table depth is used instead. No additional deferred/split transactions are accepted if this outstanding maximum count is reached. Furthermore, no additional deferred/split transactions are accepted if the I/O delay/ I/O split request outstanding maximum is reached. **NOTE:** This field must be programmed to 0x1. (CN50XX can only handle 1 delayed read at a time).

Master Deferred/Split Sequence Pending Register
PCI_CFG20

Twenty-first 32-bits of PCI configuration space. This register is for debug purposes only. It should only be written with all 0s. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	—	R/W1C	0	—	Reserved.

Master Split Completion Message Register
PCI_CFG21

Twenty-second 32-bits of PCI configuration space. This register is for debug purposes only. It should only be written with all 0s. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	—	R/W1C	0	—	Reserved.

Master Arbiter Control Register
PCI_CFG22

Twenty-third 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:25>	MAC	R/W	0	0	**Master arbiter control** [31:26]: Used only in fixed priority mode (when [25]=1) [31:30]: MSI request 00 = highest priority 01 = medium priority 10 = lowest priority 11 = Reserved [29:28]: Target split completion 00 = highest priority 01 = medium priority 10 = lowest priority 11 = Reserved [27:26]: New request; deferred read,deferred write 00 = highest priority 01 = medium priority 10 = lowest priority 11 = Reserved [25]: Fixed/round robin priority selector 0 = round robin 1 = fixed **NOTE:** When [25]=1, the three levels specified in [31:26] must be programmed to have mutually exclusive priority levels for proper operation. Failure to do so may result in the PCI becoming hung.
<24:19>	—	RAZ	—	—	Reserved
<18>	FLUSH	R/W	1	1	AM_D0_FLUSH_I control. **NOTE:** This bit must be 1 for proper CN50XX operation.
<17>	MRA	R/W1C	0	0	Master retry aborted
<16>	MTTA	R/W1C	0	0	Master TRDY time-out aborted
<15:8>	MRV	R/W	0x0	0xFF	Master retry value [1..255] and 0=infinite
<7:0>	MTTV	R/W	0x0	0x0	Master TRDY time-out value [1..255] and 0=disabled **NOTE:** This bit must be 0 for proper operation. CN50XX does not support master TRDY time-out (target is expected to be well-behaved).

Power Management Capabilities Register
PCI_CFG58

Fifty-ninth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:27>	PMES	RO	0x0	0x0	PME support (D0 to D3cold)
<26>	D2S	RO	0	0	D2_support
<25>	D1S	RO	0	0	D1_support
<24:22>	AUXC	RO	0x0	0x0	AUX_current (0...375mA)
<21>	DSI	RO	0	0	Device specific initialization
<20>	—	RAZ	—	—	Reserved
<19>	PMEC	RO	0	0	PME clock
<18:16>	PCIMIV	RO	0x2	0x2	Indicates the version of the PCI Management Interface Specification with which the core complies. 0x2 = complies with PCI Management Interface Specification Revision 1.1
<15:8>	NCP	RO	0xF0	0xF0	Next capability pointer
<7:0>	PMCID	RO	0x1	0x1	Power management capability ID

Power Management Data/PMCSR Register
PCI_CFG59

Sixtieth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description [1]
<31:24>	PMDIA	RO	0	0	Power Management data input from application (PME_DATA)
<23>	BPCCEN	RO	0	0	BPCC_En (bus power/clock control) enable
<22>	BD3H	RO	0	0	B2_B3#, B2/B3 Support for D3hot
<21:16>	—	RAZ	—	—	Reserved
<15>	PMESS	R/W1C	0	0	PME_Status sticky bit
<14:13>	PMEDSIA	RO	0	0	PME_Data_Scale input from application Device (PME_DATA_SCALE[1:0]) Specific
<12:9>	PMDS	R/W	0	0	Power Management Data_select
<8>	PMEENS	R/W	0	0	PME_En sticky bit
<7:2>	—	RAZ	—	—	Reserved
<1:0>	PS	R/W	0	0	Power State (D0 to D3) - CN50XX does not support D1/D2 Power Management states, therefore writing to this register has no effect.

1. This is not a conventional R/W style register.

MSI Capabilities Register
PCI_CFG60

Sixty-first 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:24>	—	RAZ	—	—	Reserved
<23>	M64	RO	1	1	32/64-bit message
<22:20>	MME	R/W	0x0	0x0	Multiple message enable: 000 = 1, 001 = 2, 010 = 4, 011 = 8, 100 = 16, 101 =32, 110, 111 = reserved
<19:17>	MMC	RO	0x0	0x0	Multiple message capable: 000 = 1, 001 = 2, 010 = 4, 011 = 8, 100 = 16, 101 = 32, 110,111 = reserved
<16>	MSIEN	R/W	0	0	MSI enable
<15:8>	NCP	RO	0x0	0x0	Next capability pointer
<7:0>	MSICID	RO	0x5	0x5	MSI capability ID

MSI Lower Address Register
PCI_CFG61

Sixty-second 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:2>	MSI31t2	R/W	0	—	Application specific MSI address [31:2]
<1:0>	—	RAZ	—	—	Reserved.

MSI Upper Address Register
PCI_CFG62

Sixty-third 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	MSI	R/W	0	—	MSI address [63:32]

MSI Message Data Register
PCI_CFG63

Sixty-fourth 32-bits of PCI configuration space. See Table 9–3 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	—	RAZ	—	—	Reserved
<15:0>	MSIMD	R/W	0	—	MSI message data

9.14 PCI Bus Registers

The PCI configuration-space registers and some of the BAR0 CSRs are of type PCI_NCB. This indicates that they can be accessed by either of two mechanisms:

- A remote PCI (host) device can access these registers directly via PCI addresses.
- A local core can access these registers directly via normal CN50XX internal I/O addresses.

The listed CSR addresses are all little-endian format. Note that as most CN50XX CSRs are 64-bit, and all 64-bit addresses are endian-neutral, most addresses in this specification are endian-neutral.

The PCI registers that are available to the CN50XX cores are listed in Table 9–4. The PCI registers that unavailable to the cores are listed in Table 9–5.

Table 9–4 PCI Registers (Available to Cores)

Register	Address (Little-Endian) (CN50XX Internal)	Address (Big-Endian) (CN50XX Internal)	Address (PCI BAR0 Memory Space Offset)	CSR Type[1]	Detailed Description
PCI_BAR1_INDEX0 PCI_BAR1_INDEX1 ... PCI_BAR1_INDEX30 PCI_BAR1_INDEX31	0x00011F0000001100 0x00011F0000001104 ... 0x00011F0000001178 0x00011F000000117C	0x00011F0000001104 0x00011F0000001100 ... 0x00011F000000117C 0x00011F0000001178	0x0000000000000100 0x0000000000000104 ... 0x0000000000000178 0x000000000000017C	32-bit PCI_NCB	See page 417
PCI_READ_CMD_6	0x00011F0000001180	0x00011F0000001184	0x0000000000000180	32-bit PCI_NCB	See page 417
PCI_READ_CMD_C	0x00011F0000001184	0x00011F0000001180	0x0000000000000184	32-bit PCI_NCB	See page 418
PCI_READ_CMD_E	0x00011F0000001188	0x00011F000000118C	0x0000000000000188	32-bit PCI_NCB	See page 418
PCI_CTL_STATUS_2	0x00011F000000118C	0x00011F0000001188	0x000000000000018C	32-bit PCI_NCB	See page 419
NPI_MSI_RCV	0x00011F0000001190	0x00011F0000001190	0x0000000000000190	64-bit PCI_NCB	See page 422
PCI_INT_SUM2	0x00011F0000001198	0x00011F0000001198	0x0000000000000198	64-bit PCI_NCB	See page 424
PCI_INT_ENB2	0x00011F00000011A0	0x00011F00000011A0	0x00000000000001A0	64-bit PCI_NCB	See page 423

1. NCB-type registers are accessed directly across the I/O Bus.
PCI_NCB-type registers can be accessed directly from either the PCI bus or the I/O Bus.

Table 9–5 PCI Registers (Unavailable to Cores)

Register	Address (PCI BAR0 Memory Space Offset)	CSR Type[1]	Detailed Description
PCI_WIN_WR_ADDR	0x0000000000000000	PCI	See page 426
PCI_WIN_RD_ADDR	0x0000000000000008	PCI	See page 426
PCI_WIN_WR_DATA	0x0000000000000010	PCI	See page 427
PCI_WIN_WR_MASK	0x0000000000000018	PCI	See page 427
PCI_WIN_RD_DATA	0x0000000000000020	PCI	See page 427
PCI_INT_SUM	0x0000000000000030	PCI	See page 427
PCI_INT_ENB	0x0000000000000038	PCI	See page 429
PCI_PKTS_SENT0 PCI_PKTS_SENT1	0x0000000000000040 0x0000000000000050	PCI	See page 429
PCI_PKT_CREDITS0 PCI_PKT_CREDITS1	0x0000000000000044 0x0000000000000054	PCI	See page 430
PCI_PKTS_SENT_INT_LEV0 PCI_PKTS_SENT_INT_LEV1	0x0000000000000048 0x0000000000000058	PCI	See page 430
PCI_PKTS_SENT_TIME0 PCI_PKTS_SENT_TIME1	0x000000000000004C 0x000000000000005C	PCI	See page 430
PCI_DBELL0 PCI_DBELL_1	0x0000000000000080 0x0000000000000088	PCI	See page 430
PCI_INSTR_COUNT0 PCI_INSTR_COUNT1	0x0000000000000084 0x000000000000008C	PCI	See page 431
PCI_DMA_CNT0	0x00000000000000A0	PCI	See page 431
PCI_DMA_INT_LEV0	0x00000000000000A4	PCI	See page 431
PCI_DMA_CNT1	0x00000000000000A8	PCI	See page 431
PCI_DMA_INT_LEV1	0x00000000000000AC	PCI	See page 431
PCI_DMA_TIME0	0x00000000000000B0	PCI	See page 431
PCI_DMA_TIME1	0x00000000000000B4	PCI	See page 431
PCI_MSI_RCV	0x00000000000000F0	PCI	See page 431

1. PCI-type registers are accessed directly from the PCI bus and cannot be accessed by the local cores.

9.14.1 PCI_NCB-Type Registers

All the registers in this section are PCI_NCB type registers, as described in Table 9–4.

PCI BAR1 Index Registers
PCI_BAR1_INDEX(0...31)

Contains address index and control bits for access to memory ranges of BAR1. PCI-supplied address bits [26:22] determine which register is selected. See Table 9–4 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:18>	—	RAZ	—	—	Reserved.
<17:4>	ADDR_IDX	R/W	0x0	—	Address bits <35:22> sent to L2C.
<3>	CA	R/W	0	0	Cache attribute. When set to 1, load/store operations are not cached in the L2 cache.
<2:1>	END_SWP	R/W	0x0	—	Endian-swap mode. 0x0 = No swizzle 0x1 = Byte swizzle (per-quadword), 0x2 = Byte swizzle (per-longword) 0x3 = Longword swizzle
<0>	ADDR_V	R/W	0	—	Address valid. Set to 1 when the selected address range is valid.

PCI Read Command 6 Register
PCI_READ_CMD_6

Contains control information related to a received PCI Command 6. See Table 9–4 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:9>	—	RAZ	—	—	Reserved
<8:3>	MIN_DATA	R/W	0x0	0x4	The number of words to have buffered in the PNI before informing the PCI-core that we have read data available for the outstanding delayed read. 0 is treated as a 64. For read operations to the expansion, this value is not used.
<2:0>	PREFETCH	R/W	0x0	0x2	Controls the amount of data to be prefetched when this type of bhmstREAD command is received. 000 = One 32/64-bit word. 001 = From address to end of 128-byte block. 010 = From address to end of 128-byte block plus 128 bytes. 011 = From address to end of 128-byte block plus 256 bytes. 100 = From address to end of 128-byte block plus 384 bytes. 101,110,111 = Reserved For read operations to the expansion, this value is not used.

PCI Read Command C Register
PCI_READ_CMD_C

Contains control information related to a received PCI Command C. See Table 9–4 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:9>	—	RAZ	—	—	Reserved
<8:3>	MIN_DATA	R/W	0x0	0x6	The number of words to have buffered in the PNI before informing the PCI-core that we have read data available for the outstanding delayed read. 0 is treated as a 64. For read operations to the expansion, this value is not used.
<2:0>	PREFETCH	R/W	0x0	0x3	Control the amount of data to be prefetched when this type of READ command is received. 000 = One 32/64-bit word. 001 = From address to end of 128-byte block. 010 = From address to end of 128-byte block plus 128 bytes. 011 = From address to end of 128-byte block plus 256 bytes. 100 = From address to end of 128-byte block plus 384 bytes. 101,110,111 = Reserved For read operations to the expansion, this value is not used.

PCI Read Command E Register
PCI_READ_CMD_E

Contains control information related to a received PCI Command E. See Table 9–4 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:9>	—	RAZ	—	—	Reserved
<8:3>	MIN_DATA	R/W	0x0	0x6	The number of words to have buffered in the PNI before informing the PCI-core that we have read data available for the outstanding delayed read. 0 is treated as a 64. For read operations to the expansion, this value is not used.
<2:0>	PREFETCH	R/W	0x0	0x3	Control the amount of data to be prefetched when this type of READ command is received. 000 = One 32/64-bit word. 001 = From address to end of 128-byte block. 010 = From address to end of 128-byte block plus 128 bytes. 011 = From address to end of 128-byte block plus 256 bytes. 100 = From address to end of 128-byte block plus 384 bytes. 101,110,111 = Reserved For read operations to the expansion, this value is not used.

PCI Control and Status 2 Register
PCI_CTL_STATUS_2

Control and status register accessible from both PCI and I/O bus. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:29>	—	RAZ	—	—	Reserved
<28:26>	BB1_HOLE	R/W	0x0	0x0	Big BAR1 region hole. When BB1 = 1, defines an encoded size of the upper BAR1 region that CN50XX masks out (i.e. does not respond to). (See definition of BB1_HOLE and BB1_SIZ encodings in the BB1 description).
<25>	BB1_SIZ	R/W	0	0	Big BAR1 size. When BB1 = 1, defines the programmable size of BAR1: 0 = 1GB, 1 = 2GB
<24>	BB_CA	R/W	0	0	Big BAR0/1 cache allocation. When set to 1, references to Big BAR0/1 do not allocate into the L2 cache.
<23:22>	BB_ES	R/W	0x0	0x0	Big BAR node endian-swap mode. 0 = No swizzle 1 = Byte swizzle (per-quadword), 2 = Byte swizzle (per-longword) 3 = Longword swizzle
<21>	BB1	R/W	0	0	Big BAR1 enable. When set to 1, the following differences occur: • CN50XX's BAR1 region becomes somewhere in the range 512–2048 MB rather than the default 128MB. • The following table indicates the effective size of BAR1 when BB1 is set:

BB1_SIZ	BB1_HOLE	Effective Size	Comment
0	0x0	1024 MB	Normal 1 GB BAR
0	0x1	1008 MB	1 GB, 16 MB hole
0	0x2	992 MB	1 GB, 32 MB hole
0	0x3	960 MB	1 GB, 64 MB hole
0	0x4	896 MB	1 GB, 128 MB hole
0	0x5	768 MB	1 GB, 256 MB hole
0	0x6	512 MB	1 GB, 512 MB hole
0	0x7	Illegal	
1	0x0	2048 MB	Normal 2 GB BAR
1	0x1	2032 MB	2 GB, 16 MB hole
1	0x2	2016 MB	2 GB, 32 MB hole
1	0x3	1984 MB	2 GB, 64 MB hole
1	0x4	1920 MB	2 GB, 128 MB hole
1	0x5	1792 MB	2 GB, 256 MB hole
1	0x6	1536 MB	2 GB, 512 MB hole
1	0x7	Illegal	

• When BB1_SIZ = 0, PCI_CFG06[LBASE<2:0>] read as 0x0 and are ignored on writes.
When BB1_HOLE = 0, BAR1 is an entirely ordinary 1 GB (power-of-two) BAR in all aspects.
When BB1_HOLE ≠ 0, BAR1 addresses are programmed as if the BAR were 1 GB, but CN50XX does not respond to addresses in the programmed holes.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
					• When BB1_SIZ = 1, PCI_CFG06[LBASE<3:0>] read as 0x0 and are ignored on writes. When BB1_HOLE = 0, BAR1 is an entirely ordinary 2 GB (power-of-two) BAR in all aspects. When BB1_HOLE ≠ 0, BAR1 addresses are programmed as if the BAR were 2 GB, but CN50XX does not respond to addresses in the programmed holes. **NOTE:** The BB1_HOLE value has no effect on the PCI_CFG06[LBASE] behavior. BB1_HOLE only affects whether CN50XX accepts an address. BB1_SIZ does affect PCI_CFG06[LBASE] behavior, however. • The first 128MB (i.e. addresses on the PCI bus in the range [BAR1+0 ... BAR1+0x07FFFFFF]) access CN50XX's DRAM addresses with PCI_BAR1_INDEX CSRs as usual. • The remaining address space (i.e. addresses on the PCI bus in the range [BAR1+0x08000000 ... BAR1+**size**–1] where **size** is the size of BAR1 as selected by the above table, based on the BB1_SIZ and BB1_HOLE values), are mapped to CN50XX physical DRAM addresses as follows: **PCI Address Range** **CN50XX Physical Address Range** BAR1+0x0800 0000 ... BAR1 + **size**–1 0x8800 0000 ... 0x7FFF FFFF + **size** BB_ES is the endian-swap and BB_CA is the L2C allocation bit for these references. The consequences of any burst that crosses the end of the PCI address range for BAR1 are unpredicable. • The consequences of any burst access that crosses the boundary between [BAR1 + 0x07FF FFFF] and [BAR1 + 0x0800 0000] are unpredictable in PCI-X mode. CN50XX may disconnect PCI references at this boundary.
<20>	BB0	R/W	0	0	Big BAR0 enable. When set to 1, the following differences occur: • CN50XX BAR0 becomes 2GB rather than the default 4KB. PCI_CFG04[LBASE<18:0>] reads as 0x0 and is ignored on writes. • CN50XX BAR0 becomes burstable. When BB0 = 0, CN50XX single-phase disconnects PCI BAR0 reads and writes. • The first 4KB (i.e. addresses on the PCI bus in the range [BAR0 + 0 ... BAR0 + 0xFFF]) access CN50XX PCI-type CSRs as in normal operation (i.e. when BB0 = 0). • The remaining address space (i.e. addresses on the PCI bus in the range [BAR0 + 0x1000 ... BAR0 + 0x7FFF FFFF] are mapped to CN50XX physical DRAM addresses as follows: **PCI Address Range** **CN50XX Physical Address Range** BAR0+0x0000 1000 ... BAR0+0x0FFF FFF 0x0 0000 1000 ... 0x0 0FFF FFFF BAR0+0x1000 0000 ... BAR0+0x1FFF FFF 0x4 1000 0000 ... 0x4 1FFF FFFF BAR0+0x2000 0000 ... BAR0+0x7FFF FFF 0x0 2000 0000 ... 0x0 7FFF FFFF BB_ES is the endian-swap and BB_CA is the L2 cache allocation bit for these references. The consequences of any burst that crosses the end of the PCI address range for BAR0 are unpredicable. • The consequences of any burst access that crosses the boundary between [BAR0+0xFFF[and [BAR0+0x1000[are unpredictable in PCI-X mode. CN50XX may disconnect PCI references at this boundary. • The results of any burst read that crosses the boundary between [BAR0+0x0FFF FFFF] and [BAR0+0x1 000 0000] or between [BAR0+0x1FFF FFFF] and [BAR0+0x2000 0000] are unpredictable. The consequences of any burst write that crosses this same boundary are unpredictable.
<19>	ERST_N	RO	1	1	Reset active low.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<18>	BAR2PRES	R/W	—	—	From fuse block: • When fuse MIO_FUS_DAT3[BAR2_EN] is not blown, the value of this field is 0 after reset and BAR2 is not present. • When the fuse is blown, the value of this field is 1 after reset and BAR2 is present. **NOTE:** Software can change this field after reset.
<17>	SCMTYP	RO	0	—	Split completion message CMD type: 0 = RD, 1 = WR). When SCM=1, SCMTYP specifies the CMD intent (R/W)
<16>	SCM	RO	0	—	Split completion message detect (read or write): 1= detected, 0= not detected
<15>	EN_WFILT	R/W	0	1	Window-access filter enable: 1 = enabled, 0 = not enabled. Unfiltered writes: MIO, SubId0 MIO, SubId7 NPI, SubId0 NPI, SubId7 IPD, SubId7 USBN, SubId7 POW, SubId7 Unfiltered reads MIO, SubId0 MIO, SubId7 NPI, SubId0 NPI, SubId7 IPD, SubId7 USBN, SubId7 POW, SubId1 POW, SubId2 POW, SubId3 POW, SubId7
<14>	—	RAZ	—	—	Reserved
<13>	AP_PCIX	RO	0	—	PCX core mode status: 0 = PCI, 1 = PCIX
<12>	AP_64AD	RO	0	—	PCX core bus status: 0 = 32-bit bus, 1 = 64-bit bus
<11>	B12_BIST	RO	0	0	Bist status for memory in B12
<10>	PMO_AMOD	R/W	0	0	PMO-ARB mode: 0 = FP mode {HP=CMD1,LP=CMD0}, 1 = RR mode
<9:7>	PMO_FPC	R/W	0x0	0x0	PMO-ARB fixed priority counter When PMO_AMOD=0 (FP mode), this field represents the # of CMD1 requests that are issued (at higher priority) before a single lower-priority CMD0 is allowed to issue (to ensure forward progress). 0 = one CMD1 request issued before CMD0 allowed ... 7 = eight CMD1 requests issued before CMD0 allowed

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<6:4>	TSR_HWM	R/W	0x1	—	Target split-read allowable disconnect boundary (ADB) high-water mark. Specifies the number of ADBs (128-byte aligned chunks) that are accumulated (pending) before the target-split completion is attempted on the PCI bus. 0x0 = Reserved/Illegal 0x1 = 2 (129-256 bytes) 0x2 = 3 (257-384 bytes) 0x3 = 4 (385-512 bytes) 0x4 = 5 (513-640 bytes) 0x5 = 6 (641-768 bytes) 0x6 = (769-896 bytes) 0x7 = (897-1024 bytes) **Example:** Suppose a 1KB target-memory request with starting byte offset address[6:0] = 0x7F is split by the CN50XX and the TSR_HWM=1 (2 ADBs). The CN50XX starts the target split completion on the PCI bus after 1 byte (1st ADB) + 128 bytes (2nd ADB) = 129 bytes of data have been received from memory (even though the remaining 895 bytes have not yet been received). CN50XX continues the split completion until it has consumed all of the pended split data. If the full transaction length (1KB) of data was not entirely transferred, CN50XX terminates the split completion and again waits for another 2 ADB-aligned data chunks (256 bytes) of pended split data to be received from memory before starting another split completion request. This allows CN50XX (as split completer), to send back multiple split completions for a given large split transaction without having to wait for the entire transaction length to be received from memory. **NOTE:** For split-transaction sizes smaller than the specified TSR_HWM value, the split completion is started when the last datum has been received from memory. **NOTE:** This field must never be written to a 0x0 value. A value of 0x0 is reserved/illegal and can result in PCI bus hangs.
<3>	BAR2_ENB	R/W	0	1	BAR2 enable. 1 = BAR2 is enabled and will respond 0 = BAR2 access is target-aborted.
<2:1>	BAR2_ESX	R/W	0x0	—	Endian swap mode. Value is XORed with PCI_ADDRESS[37:36] to determine the endian-swap mode: 0 = No swizzle 1 = Byte swizzle (per-quadword), 2 = Byte swizzle (per-longword) 3 = Longword swizzle
<0>	BAR2_CAX	R/W	0	0	L2 cache attribute. Value is XORed with PCI_ADDRESS[38] to determine the L2 cache attribute. Refer to Section 9.2.3. If BAR2_CAX ⊕ [38] = 1, load/store operations are not allocated to L2 cache.

NPI MSI Receive Vector Register
NPI_MSI_RCV

A bit is set in this register relative to the vector received during an MSI. Cleared by writing a 1 to the register. See Table 9–4 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	INT_VEC	R/W1C	0	0	Refer to PCI_MSI_RCV.

PCI Interrupt Enable2 Register
PCI_INT_ENB2

Enables interrupt bits in the PCI_INT_SUM2 register. See Table 9–4 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:34>	—	RAZ	—	—	Reserved
<33>	ILL_RD	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[ILL_RD]
<32>	ILL_WR	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[ILL_WR]
<31>	WIN_WR	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM[WIN_WR]
<30>	DMA1_FI	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[DMA1_FI]
<29>	DMA0_FI	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[DMA0_FI]
<28>	RDTIME1	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[DTIME1]
<27>	RDTIME0	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[DTIME0]
<26>	RDCNT1	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[RDCNT1]
<25>	RDCNT0	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[RDCNT0]
<24>	SPR3	R/W	0	—	Spare.
<23>	SPR2	R/W	0	—	Spare.
<22>	SPR5	R/W	0	—	Spare.
<21>	RPTIME0	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[PTIME0]
<20>	SPR1	R/W	0	—	Spare.
<19>	SPR0	R/W	0	—	Spare.
<18>	SPR4	R/W	0	—	Spare.
<17>	RPCNT0	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[PCNT0]
<16>	RRSL_INT	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[RSL_INT]
<15>	ILL_RRD	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[ILL_RRD]
<14>	ILL_RWR	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[ILL_RWR]
<13>	RDPERR	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[DPERR]
<12>	RAPERR	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[APERR]
<11>	RSERR	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[SERR]
<10:9>	—	R/W	0	—	Reserved.
<8>	RMSI_MABT	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[MSI_MABT]
<7>	RMSI_TABT	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[MSI_TABT]
<6>	RMSI_PER	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[MSI_PER]
<5>	RMR_TTO	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[MR_TTO]
<4>	RMR_ABT	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[MR_ABT]
<3>	RTR_ABT	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[TR_ABT]
<2>	RMR_WTTO	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[MR_WTTO]
<1>	RMR_WABT	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[MR_WABT]
<0>	RTR_WABT	R/W	0	—	RSL chain interrupt enable for PCI_INT_SUM2[TR_WABT]

PCI Interrupt Summary2 Register
PCI_INT_SUM2

The PCI Interrupt Summary2 Register copy used for RSL interrupts. See Table 9–4 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:34>	—	RAZ	—	—	Reserved
<33>	ILL_RD	R/W1C	0	0	A read to a disabled area of BAR1 or BAR2, when the mem area is disabled.
<32>	ILL_WR	R/W1C	0	0	A write to a disabled area of BAR1 or BAR2, when the mem area is disabled.
<31>	WIN_WR	R/W1C	0	0	A write to the disabled window write-data register took place.
<30>	DMA1_FI	R/W1C	0	0	A DMA operation finished that was required to set the FORCE-INT bit for counter 1.
<29>	DMA0_FI	R/W1C	0	0	A DMA operation finished that was required to set the FORCE-INT bit for counter 0.
<28>	DTIME1	R/W1C	0	0	When the value in the PCI_DMA_CNT1 register is not 0, the DMA_CNT1 timer counts. When the DMA1_CNT timer has a value greater than the PCI_DMA_TIME1 register, this bit is set. The timer is reset when the bit is written with a one.
<27>	DTIME0	R/W1C	0	0	When the value in the PCI_DMA_CNT0 register is not 0 the DMA_CNT0 timer counts. When the DMA0_CNT timer has a value greater than the PCI_DMA_TIME0 register this bit is set. The timer is reset when the bit is written with a one.
<26>	DCNT1	R/W1C	0	0	This bit indicates that PCI_DMA_CNT1 value is greater than the value in the PCI_DMA_INT_LEV1 register.
<25>	DCNT0	R/W1C	0	0	This bit indicates that PCI_DMA_CNT0 value is greater than the value in the PCI_DMA_INT_LEV0 register.
<24>	SPR3	R/W1C	0	0	Spare.
<23>	SPR2	R/W1C	0	0	Spare.
<22>	SPR5	R/W1C	0	0	Spare.
<21>	PTIME0	R/W1C	0	0	When the value in PCI_PKTS_SENT0 is not 0, the Sent0 timer counts. When the Sent0 timer has a value greater than the PCI_PKTS_SENT_TIME0 value, this bit is set. The timer is reset when the bit is written with a 1.
<20>	SPR1	R/W1C	0	0	Spare.
<19>	SPR0	R/W1C	0	0	Spare.
<18>	SPR4	R/W1C	0	0	Spare.
<17>	PCNT0	R/W1C	0	0	This bit indicates that PCI_PKTS_SENT0 value is greater than the value in the PCI_PKTS_SENT_INT_LEV0 register.
<16>	RSL_INT	RO	0	0	This bit is set when the RSL Chain has generated an interrupt.
<15>	ILL_RRD	R/W1C	0	0	A read to the disabled PCI registers took place.
<14>	ILL_RWR	R/W1C	0	0	A write to the disabled PCI registers took place.
<13>	DPERR	R/W1C	0	0	Data parity error detected by PCI core
<12>	APERR	R/W1C	0	0	Address parity error detected by PCI core
<11>	SERR	R/W1C	0	0	SERR# detected by PCI core
<10>	TSR_ABT	R/W1C	0	0	Target split-read abort detected
<9>	MSC_MSG	R/W1C	0	0	Master split completion message detected
<8>	MSI_MABT	R/W1C	0	0	PCI MSI master abort.
<7>	MSI_TABT	R/W1C	0	0	PCI MSI target abort.
<6>	MSI_PER	R/W1C	0	0	PCI MSI parity error.
<5>	MR_TTO	R/W1C	0	0	PCI master retry time-out on read.
<4>	MR_ABT	R/W1C	0	0	PCI master abort on read.
<3>	TR_ABT	R/W1C	0	0	PCI target abort on read.
<2>	MR_WTTO	R/W1C	0	0	PCI master retry time-out on write.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<1>	MR_WABT	R/W1C	0	0	PCI master abort detected on write.
<0>	TR_WABT	R/W1C	0	0	PCI target abort detected on write.

9.14.2 PCI-Type Registers

All the registers in this section are PCI type registers, as described in Table 9–5.

PCI Window Write Address Register
PCI_WIN_WR_ADDR

Contains the address to be written to when a write operation is started by writing the PCI_WIN_WR_DATA register. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:49>	—	RAZ	—	—	Reserved
<48>	IOBIT	RAZ	0x0	0x0	A 1 or 0 can be written here, but this always reads as 0.
<47:3>	WR_ADDR	R/W	0x0	—	The address that is written to when the PCI_WIN_WR_DATA register is written. [47:40] = I/O BUS_ID [39:3] = address When [47:43] = NPI AND [42:40] = 0x0 bits, [39:0] are the following: [39:32] = don't care (not used) [31:27] = RSL_ID [12:3] = RSL register offset [2:0] = don't care (not used)
<2:0>	—	RAZ	—	—	Reserved

PCI Window Read Address Register
PCI_WIN_RD_ADDR

Writing the least-significant byte of this register causes a read operation to take place, unless a read operation is already taking place. A read is considered to end when the PCI_WIN_RD_DATA register is read. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:49>	—	RAZ	—	—	Reserved
<48>	IOBIT	RAZ	0x0	0x0	A 1 or 0 can be written here, but this will always read as 0.
<47:2>	RD_ADDR	R/W	0x0	—	The address to be read from. Whenever the LSB of this register is written, the read operation will take place. [47:40] = I/O Bus_ID [39:3] = Address When [47:43] = NPI AND [42:40] = 0x0 bits, [39:0] are the following: [39:32] = don't care (not used) [31:27] = RSL_ID [12:2] = RSL register offset [1:0] = don't care (not used)
<1:0>	—	RAZ	—	—	Reserved

PCI Window Write Data Register
PCI_WIN_WR_DATA

Contains the data to write to the address located in the PCI_WIN_RD_ADDR register. Writing the least-significant byte of this register causes a write operation to take place. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	WR_DATA	R/W	0x0	—	The data to be written. Whenever the least-significant byte of this register is written, the window write takes place.

PCI Window Write Mask Register
PCI_WIN_WR_MASK

Contains the mask for the data in the PCI_WIN_WR_DATA Register. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved
<7:0>	WR_MASK	R/W	0x0	0x0	Data-to-be-written mask. When a bit is set to 1, the corresponding byte is not written.

PCI Window Read Data Register
PCI_WIN_RD_DATA

Contains the result from the read operation that took place when the least-significant byte of the PCI_WIN_RD_ADDR register was written. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	RD_DATA	RO	0x0	—	Read data.

PCI Interrupt Summary Register
PCI_INT_SUM

See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:34>	—	RAZ	—	—	Reserved
<33>	ILL_RD	R/W1C	0	0	A read to a disabled area of BAR1 or BAR2, when the mem area is disabled.
<32>	ILL_WR	R/W1C	0	0	A write to a disabled area of BAR1 or BAR2, when the mem area is disabled.
<31>	WIN_WR	R/W1C	0	0	A write to the disabled window write-data or read-address register took place.
<30>	DMA1_FI	R/W1C	0	0	A DMA operation that was required to set the FORCE-INT bit for counter 1 finished.
<29>	DMA0_FI	R/W1C	0	0	A DMA operation that was required to set the FORCE-INT bit for counter 0 finished.
<28>	DTIME1	R/W1C	0	0	When the value in the PCI_DMA_CNT1 register is not 0, the DMA_CNT1 timer counts. When the DMA1_CNT timer has a value greater than the PCI_DMA_TIME1 register this bit is set. The timer is reset when the bit is written with a 1.
<27>	DTIME0	R/W1C	0	0	When the value in the PCI_DMA_CNT0 register is not 0 the DMA_CNT0 timer counts. When the DMA0_CNT timer has a value greater than the PCI_DMA_TIME0 register this bit is set. The timer is reset when the bit is written with a one.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<26>	DCNT1	R/W1C	0	0	This bit indicates that PCI_DMA_CNT1 value is greater than the value in the PCI_DMA_INT_LEV1 register.
<25>	DCNT0	R/W1C	0	0	This bit indicates that PCI_DMA_CNT0 value is greater than the value in the PCI_DMA_INT_LEV0 register.
<24>	SPR3	R/W1C	0	0	Spare.
<23>	SPR2	R/W1C	0	0	Spare.
<22>	SPR5	R/W1C	0	0	Spare.
<21>	PTIME0	R/W1C	0	0	When the value in the PCI_PKTS_SENT0 register is not 0, the Sent0 timer counts. When the Sent0 timer has a value greater than the PCI_PKTS_SENT_TIME0 register, this bit is set. The timer is reset when the bit is written with a 1.
<20>	SPR1	R/W1C	0	0	Spare.
<19>	SPR0	R/W1C	0	0	Spare.
<18>	SPR4	R/W1C	0	0	Spare.
<17>	PCNT0	R/W1C	0	0	This bit indicates that PCI_PKTS_SENT0 value is greater than the value in the PCI_PKTS_SENT_INT_LEV0 register.
<16>	RSL_INT	RO	0	0	This bit is set when the MIO_PCI_INTA_DR wire is asserted by the MIO.
<15>	ILL_RRD	R/W1C	0	0	Indicates a read to the disabled PCI registers took place.
<14>	ILL_RWR	R/W1C	0	0	Indicates a write to the disabled PCI registers took place.
<13>	DPERR	R/W1C	0	0	Data parity error detected by PCI core
<12>	APERR	R/W1C	0	0	Address parity error detected by PCI core
<11>	SERR	R/W1C	0	0	SERR# detected by PCI core
<10>	TSR_ABT	R/W1C	0	0	Target split-read abort detected. CN50XX (as completer), has encountered an error that prevents the split transaction from completing. The format for the SCM is defined in the PCI_CFG21 register, (Master Split Completion Message Register). In this event, CN50XX (as completer), sends an SCM (split completion message) to the initiator. [31:28]: message class = 2 (completer error) [27:20]: message index = 0x80 [18:12]: remaining lower address [11:0]: remaining byte count
<9>	MSC_MSG	R/W1C	0	0	Master split completion message (SCM), detected for either a split-read/ -write error case. This bit is set if: • A split-write SCM is detected with SCE = 1. • A split-read SCM is detected (regardless of SCE status). The split completion message(SCM) is also latched into the PCI_SCM_REG[SCM] to assist SW with error recovery.
<8>	MSI_MABT	R/W1C	0	0	PCI MSI master abort.
<7>	MSI_TABT	R/W1C	0	0	PCI MSI target abort.
<6>	MSI_PER	R/W1C	0	0	PCI MSI parity error.
<5>	MR_TTO	R/W1C	0	0	PCI master retry time-out on read.
<4>	MR_ABT	R/W1C	0	0	PCI master abort on read.
<3>	TR_ABT	R/W1C	0	0	PCI target abort on read.
<2>	MR_WTTO	R/W1C	0	0	PCI master retry time-out on write.
<1>	MR_WABT	R/W1C	0	0	PCI master abort detected on write.
<0>	TR_WABT	R/W1C	0	0	PCI target abort detected on write.

PCI Interrupt Enable Register
PCI_INT_ENB

Enables interrupt bits in the PCI_INT_SUM register See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:34>	—	RAZ	—	—	Reserved
<33>	ILL_RD	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[ILL_RD]
<32>	ILL_WR	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[ILL_WR]
<31>	WIN_WR	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[WIN_WR]
<30>	DMA1_FI	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[DMA1_FI]
<29>	DMA0_FI	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[DMA0_FI]
<28>	IDTIME1	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[DTIME1]
<27>	IDTIME0	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[DTIME0]
<26>	IDCNT1	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[DCNT1]
<25>	IDCNT0	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[DCNT0]
<24>	SPR3	R/W	0	—	Spare.
<23>	SPR2	R/W	0	—	Spare.
<22>	SPR5	R/W	0	0	Spare.
<21>	IPTIME0	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[PTIME0]
<20>	SPR1	R/W	0	—	Spare.
<19>	SPR0	R/W	0	—	Spare.
<18>	SPR4	R/W	0	0	Spare.
<17>	IPCNT0	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[PCNT0]
<16>	IRSL_INT	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[RSL_INT]
<15>	ILL_RRD	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[ILL_RRD]
<14>	ILL_RWR	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[ILL_RWR]
<13>	IDPERR	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[DPERR]
<12>	IAPERR	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[APERR]
<11>	ISERR	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[SERR]
<10>	ITSR_ABT	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[TSR_ABT]
<9>	IMSC_MSG	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[MSC_MSG]
<8>	IMSI_MABT	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[MSI_MABT]
<7>	IMSI_TABT	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[MSI_TABT]
<6>	IMSI_PER	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[MSI_PER]
<5>	IMR_TTO	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[MR_TTO]
<4>	IMR_ABT	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[MR_ABT]
<3>	ITR_ABT	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[TR_ABT]
<2>	MR_WTTO	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[MR_WTTO]
<1>	IMR_WABT	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[MR_WABT]
<0>	ITR_WABT	R/W	0	—	INTA# pin interrupt enable for PCI_INT_SUM[TR_WABT]

PCI Packets Sent Registers
PCI_PKTS_SENT0/1

The number of packets sent to the host memory from PCI output 0/1. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	PKT_CNT	RO	0	0	Each time a packet is written to the memory via PCI from CN50XX's PCI outputs 0/1, this counter is incremented by 1 or the byte count of the packet as set in NPI_OUTPUT_CONTROL[P0/1_BMODE].

PCI Packet Credits For Output Credit Registers
PCI_PKT_CREDITS0/1

This register is used to decrease the number of packets to be processed by the host from Output 0/1 and return buffer/info-pointer pairs to CN50XX Output 0/1. The value in this register is acted upon when the least-significant byte of this register is written. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	PKT_CNT	R/W	0	—	The value written to this field is subtracted from PCI_PKTS_SENT0/1[PKT_CNT].
<15:0>	PTR_CNT	R/W	0	—	This value is added to the NPI's internal buffer/info-pointer pair count.

PCI Packets Sent Interrupt Level For Output Registers
PCI_PKTS_SENT_INT_LEV0/1

Interrupt when number of packets sent is equal to or greater than the register value. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	PKT_CNT	R/W	0	—	When the corresponding port's packet-sent register exceeds or is equal to the value in this register, an interrupt occurs.

PCI Packets Sent Timer For Output Registers
PCI_PKTS_SENT_TIME0/1

Time to wait from packet being sent to host from Output 0/1 before issuing an interrupt. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	PKT_TIME	R/W	0	—	The number of PCI clock cycles to wait before issuing an interrupt to the host when a packet from this port has been sent to the host. The timer is reset when the PCI_INT_SUM[21] is cleared.

PCI Doorbell Registers
PCI_DBELL0/1

The value to write to the doorbell 0/1 register. The value in this register is acted upon when the least-significant byte of this register is written. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	—	RAZ	—	—	Reserved
<15:0>	INC_VAL	R/W	0	—	Software writes this register with the number of new instructions to be processed on the instruction queue. When read, this register contains the last write value.

PCI Instructions Outstanding Request Count Registers
PCI_INSTR_COUNT0/1

The number of instructions to be fetched by the Instruction 0/1 engines. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	ICNT	RO	0x0	0x0	Number of instructions to be fetched by the Instruction 0 engine in CN50XX.

PCI DMA Count0/1 Registers
PCI_DMA_CNT0/1

Keeps track of the number of DMA operations or bytes sent by DMA operations. The value in this register is acted upon when the least-significant byte of this register is written. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	DMA*n*_CNT	R/W	0x0	0x0	Update with the number of DMA operations completed or the number of bytes sent for DMA operations associated with this counter. When this register is written, the value written to [15:0] is subtracted from the value in this register.

PCI DMA Sent Interrupt Level For DMA 0/1 Registers
PCI_DMA_INT_LEV0/1

Interrupt when the value in PCI_DMA_CNT0 is equal to or greater than the register value. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	PKT_CNT	R/W	0x0	—	When PCI_DMA_CNT0/1 register exceeds the value in this register, an interrupt occurs if enabled.

PCI DMA Sent Timer For DMA0/1 Registers
PCI_DMA_TIME0/1

Time to wait from the DMA operation being sent before issuing an interrupt. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	DMA_TIME	R/W	0x0	—	Number of PCI clock cycles to wait before issuing an interrupt to the host when a DMA operation associated with this DMA counter is completed. The timer is reset when the PCI_INT_SUM[DTIME*n*] bit is cleared (bits [28,27]).

PCI MSI Received Vector Register
PCI_MSI_RCV

A bit is set in this register relative to the vector received during a MSI. The value in this register is acted upon when the least-significant byte of this register is written. See Table 9–5 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:6>	—	RAZ	—	—	Reserved.
<5:0>	INT	WO	0x0	—	When an MSI is received on the PCI, the bit selected by data [5:0] is set in this register. To clear this bit, a write must take place to the NPI_MSI_RCV register where any bit set to 1 is cleared. Reading this register returns an unpredictable value.

9.15 NPI Registers

The NPI registers are listed in Table 9–6 and Table 9–7.

Table 9–6 NPI Registers

Register	Address	CSR Type[1]	Detailed Description
NPI_RSL_INT_BLOCKS	0x00011F0000000000	NCB	See page 433
NPI_DBG_SELECT	0x00011F0000000008	NCB	See page 433
NPI_CTL_STATUS	0x00011F0000000010	NCB	See page 434
NPI_INT_SUM	0x00011F0000000018	NCB	See page 434
NPI_INT_ENB	0x00011F0000000020	NCB	See page 436
NPI_MEM_ACCESS_SUBID3 ... NPI_MEM_ACCESS_SUBID6	0x00011F0000000028 ... 0x00011F0000000040	NCB	See page 438
NPI_PCI_READ_CMD	0x00011F0000000048	NCB	See page 438
NPI_NUM_DESC_OUTPUT0 NPI_NUM_DESC_OUTPUT1	0x00011F0000000050 0x00011F0000000058	NCB	See page 438
NPI_BASE_ADDR_INPUT0 NPI_BASE_ADDR_INPUT1	0x00011F0000000070 0x00011F0000000080	NCB	See page 439
NPI_SIZE_INPUT0 NPI_SIZE_INPUT1	0x00011F0000000078 0x00011F0000000088	NCB	See page 439
PCI_READ_TIMEOUT	0x00011F00000000B0	NCB	See page 439
NPI_BASE_ADDR_OUTPUT0 NPI_BASE_ADDR_OUTPUT1	0x00011F00000000B8 0x00011F00000000C0	NCB	See page 439
NPI_PCI_BURST_SIZE	0x00011F00000000D8	NCB	See page 440
NPI_BUFF_SIZE_OUTPUT0 NPI_BUFF_SIZE_OUTPUT1	0x00011F00000000E0 0x00011F00000000E8	NCB	See page 440
NPI_OUTPUT_CONTROL	0x00011F0000000100	NCB	See page 441
NPI_LOWP_IBUFF_SADDR	0x00011F0000000108	NCB	See page 441
NPI_HIGHP_IBUFF_SADDR	0x00011F0000000110	NCB	See page 441
NPI_LOWP_DBELL	0x00011F0000000118	NCB	See page 442
NPI_HIGHP_DBELL	0x00011F0000000120	NCB	See page 442
NPI_DMA_CONTROL	0x00011F0000000128	NCB	See page 443
NPI_PCI_INT_ARB_CFG	0x00011F0000000130	NCB	See page 443
NPI_INPUT_CONTROL	0x00011F0000000138	NCB	See page 444
NPI_DMA_LOWP_COUNTS	0x00011F0000000140	NCB	See page 444
NPI_DMA_HIGHP_COUNTS	0x00011F0000000148	NCB	See page 445
NPI_DMA_LOWP_NADDR	0x00011F0000000150	NCB	See page 445
NPI_DMA_HIGHP_NADDR	0x00011F0000000158	NCB	See page 445
NPI_P0_PAIR_CNTS NPI_P1_PAIR_CNTS	0x00011F0000000160 0x00011F0000000168	NCB	See page 445
NPI_P0_DBPAIR_ADDR NPI_P1_DBPAIR_ADDR	0x00011F0000000180 0x00011F0000000188	NCB	See page 445
NPI_P0_INSTR_CNTS NPI_P1_INSTR_CNTS	0x00011F00000001A0 0x00011F00000001A8	NCB	See page 446
NPI_P0_INSTR_ADDR NPI_P1_INSTR_ADDR	0x00011F00000001C0 0x00011F00000001C8	NCB	See page 446
NPI_WIN_READ_TO	0x00011F00000001E0	NCB	See page 446
DBG_DATA	0x00011F00000001E8	NCB	See page 446
NPI_PORT_BP_CONTROL	0x00011F00000001F0	NCB	See page 440
NPI_PORT32_INSTR_HDR NPI_PORT33_INSTR_HDR	0x00011F00000001F8 0x00011F0000000200	NCB	See page 447
NPI_BIST_STATUS	0x00011F00000003F8	NCB	See page 448

1. NCB-type registers are accessed directly across the I/O Bus.

Table 9–7 NPI/PCI Register

Register	Address (NCB)	Address (PCI_NCB)	CSR Type[1]	Detailed Description
NPI_MSI_RCV	0x00011F0000001190	0x0000000000000190	NCB, PCI_NCB	See page 422

1. NCB-type registers are accessed directly across the I/O Bus. PCI_NCB-type registers can be accessed directly from either the PCI bus or the I/O Bus.

NPI RSL Interrupt Blocks Register
NPI_RSL_INT_BLOCKS

Reading this register returns a vector with a bit set to 1 for a corresponding RSL block that presently has an interrupt pending. The field description below supplies the name of the register that software should read to find out why that interrupt bit is set. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:31>	—	RAZ	—	—	Reserved
<30>	IOB	RO	0	0	IOB_INT_SUM
<29:23>	—	RAZ	—	—	Reserved
<22>	ASX0	RO	0	0	ASX0_INT_REG
<21>	—	RAZ	—	—	Reserved
<20>	PIP	RO	0	0	PIP_INT_REG
<19:18>	—	RAZ	—	—	Reserved
<17>	LMC	RO	0	0	LMC_MEM_CFG0
<16>	L2C	RO	0	0	L2T_ERR and L2D_ERR
<15:14>	—	RAZ	—	—	Reserved
<13>	USB	RO	0	0	USBN_INT_SUM
<12>	POW	RO	0	0	POW_ECC_ERR
<11>	TIM	RO	0	0	TIM_REG_ERROR
<10>	PKO	RO	0	0	PKO_REG_ERROR
<9>	IPD	RO	0	0	IPD_INT_SUM
<8:6>	—	RAZ	—	—	Reserved
<5>	FPA	RO	0	0	FPA_INT_SUM
<4>	—	RAZ	—	—	Reserved
<3>	NPI	RO	0	0	NPI_INT_SUM
<2>	—	RAZ	—	—	Reserved
<1>	GMX0	RO	0	0	GMX0_RX*n*_INT_REG & GMX0_TX_INT_REG
<0>	MIO	RO	0	0	MIO_BOOT_ERR

NPI Debug Select Register
NPI_DBG_SELECT

Contains the debug select value last written to the RSLs. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	DBG_SEL	R/W	0x0	0x0	When this register is written, its value is sent to all RSLs.

NPI Control Status Register
NPI_CTL_STATUS

Contains control and status for NPI. Writes to this register are not ordered with write/read operations to the PCI memory space. To ensure that a write has completed, read the register before making an access (i.e. PCI memory space) that requires the value of this register to be updated. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	—	RAZ	—	—	Reserved
<62:55>	CHIP_REV	RO	—	—	The revision of CN50XX. 0x0 = pass 1.0 0x1 = pass 1.1
<54>	DIS_PNIW	R/W	0	1	When set to 1, access from the PNI window registers are disabled.
<53>	SPR5	R/W	0	1	Spare.
<52>	SPR4	R/W	0	1	Spare.
<51>	SPR8	R/W	0	0	Spare.
<50>	OUT0_ENB	R/W	0	1	When set to 1, the output0 engine is enabled. After enabling the values of the associated address and size register should not be changed.
<49>	SPR3	R/W	0	1	Spare.
<48>	SPR2	R/W	0	1	Spare.
<47>	SPR7	R/W	0	0	Spare.
<46>	INS0_ENB	R/W	0	1	When set to 1, the gather0 engine is enabled. After enabling the values of the associated address and size register should not be changed.
<45>	SPR1	R/W	0	0	Spare.
<44>	SPR0	R/W	0	0	Spare.
<43>	SPR6	R/W	0	—	Spare.
<42>	INS0_64B	R/W	0	—	When set to 1, the instructions read by the gather0 engine are 64 byte instructions; when deasserted to 0, instructions are 32 bytes.
<41>	PCI_WDIS	R/W	0	0	When set to 1, disables access to registers in the PNI in address range 0x1000 - 0x17FF from the PCI.
<40>	WAIT_COM	R/W	0	1	When set to 1, causes the NPI to wait for a commit from the L2C before sending additional access to the L2C from the PCI.
<39:37>	SPARES1	R/W	0x0	0x0	These bits are reserved and should be set to 0.
<36:32>	MAX_WORD	R/W	0x2	0x0	The maximum number of words to merge into a single write operation from the cores to the PCI. Legal values are 1 to 32, where a 0 is treated as 32.
<31:10>	SPARES0	R/W	0x0	0x0	These bits are reserved and should be set to 0.
<9:0>	TIMER	R/W	0x0	0x32	When the NPI starts a core-to-PCI write, it waits no longer than the value of TIMER in core clock cycles to merge additional writes from the cores into one large write. The values for this field are 1 to 1024 where a value of 0 is treated as 1024.

NPI Interrupt Summary Register
NPI_INT_SUM

Set when an interrupt condition occurs, write 1 to clear. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:62>	—	RAZ	—	—	Reserved
<61>	Q1_A_F	R/W1C	0	0	Attempted to subtract when Queue 1 FIFO is empty.
<60>	Q1_S_E	R/W1C	0	0	Attempted to subtract when Queue 1 FIFO is empty.
<59>	PDF_P_F	R/W1C	0	0	Attempted to push a full PCN data FIFO.
<58>	PDF_P_E	R/W1C	0	0	Attempted to pop an empty PCN-data FIFO.
<57>	PCF_P_F	R/W1C	0	0	Attempted to push a full PCN-CNT FIFO.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<56>	PCF_P_E	R/W1C	0	0	Attempted to pop an empty PCN-CNT FIFO.
<55>	RDX_S_E	R/W1C	0	0	Attempted to subtract when DPI-XFR wait count is 0.
<54>	RWX_S_E	R/W1C	0	0	Attempted to subtract when RDN-XFR wait count is 0.
<53>	PNC_A_F	R/W1C	0	0	Attempted to add when PNI-NPI credits are max.
<52>	PNC_S_E	R/W1C	0	0	Attempted to subtract when PNI-NPI credits are 0.
<51>	COM_A_F	R/W1C	0	0	Attempted to add when PCN-commit counter is max.
<50>	COM_S_E	R/W1C	0	0	Attempted to subtract when PCN-commit counter is 0.
<49>	Q3_A_F	R/W1C	0	0	Attempted to add when Queue 3 FIFO is full.
<48>	Q3_S_E	R/W1C	0	0	Attempted to subtract when Queue 3 FIFO is empty.
<47>	Q2_A_F	R/W1C	0	0	Attempted to add when Queue 2 FIFO is full.
<46>	Q2_S_E	R/W1C	0	0	Attempted to subtract when Queue 2 FIFO is empty.
<45>	PCR_A_F	R/W1C	0	0	Attempted to add when POW credits is full.
<44>	PCR_S_E	R/W1C	0	0	Attempted to subtract when POW credits is empty.
<43>	FCR_A_F	R/W1C	0	0	Attempted to add when FPA credits is full.
<42>	FCR_S_E	R/W1C	0	0	Attempted to subtract when FPA credits is empty.
<41>	IOBDMA	R/W1C	0	0	Requested IOBDMA read size exceeded 128 words.
<40>	P_DPERR	R/W1C	0	0	Data written to L2C from PCI that had a data parity error.
<39>	WIN_RTO	R/W1C	0	0	Windowed load timed out.
<38>	SPR17	R/W1C	0	0	Spare.
<37>	SPR16	R/W1C	0	0	Spare.
<36>	SPR26	R/W1C	0	0	Spare.
<35>	I0_PPERR	R/W1C	0	0	Port0 instruction had a parity error.
<34>	SPR15	R/W1C	0	0	Spare.
<33>	SPR14	R/W1C	0	0	Spare.
<32>	SPR25	R/W1C	0	0	Spare.
<31>	P0_PTOUT	R/W1C	0	0	Port0 output had a read time-out on a data/info pair.
<30>	SPR13	R/W1C	0	0	Spare.
<29>	SPR12	R/W1C	0	0	Spare.
<28>	SPR24	R/W1C	0	0	Spare.
<27>	P0_PPERR	R/W1C	0	0	Port0 output had a parity error on a data/info pair.
<26>	SPR11	R/W1C	0	0	Spare.
<25>	SPR10	R/W1C	0	0	Spare.
<24>	SPR23	R/W1C	0	0	Spare.
<23>	G0_RTOUT	R/W1C	0	0	Port0 had a read time-out while attempting to read a gather list.
<22>	SPR9	R/W1C	0	0	Spare.
<21>	SPR8	R/W1C	0	0	Spare.
<20>	SPR22	R/W1C	0	0	Spare.
<19>	P0_PERR	R/W1C	0	0	Port0 had a parity error on packet data.
<18>	SPR7	R/W1C	0	0	Spare.
<17>	SPR6	R/W1C	0	0	Spare.
<16>	SPR21	R/W1C	0	0	Spare.
<15>	P0_RTOUT	R/W1C	0	0	Port0 had a read time-out while attempting to read packet data.
<14>	SPR5	R/W1C	0	0	Spare.
<13>	SPR4	R/W1C	0	0	Spare.
<12>	SPR20	R/W1C	0	0	Spare.
<11>	I0_OVERF	R/W1C	0	0	Port0 had a doorbell overflow. Bit[31] of the doorbell count was set.
<10>	SPR3	R/W1C	0	0	Spare.
<9>	SPR2	R/W1C	0	0	Spare.
<8>	SPR19	R/W1C	0	0	Spare.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<7>	I0_RTOUT	R/W1C	0	0	Port0 had a read time-out while attempting to read instructions.
<6>	SPR1	R/W1C	0	0	Spare.
<5>	SPR0	R/W1C	0	0	Spare.
<4>	SPR18	R/W1C	0	0	Spare.
<3>	PO0_2SML	R/W1C	0	0	The packet being sent out on Port0 is smaller than the NPI_BUFF_SIZE_OUTPUT0[ISIZE] field.
<2>	PCI_RSL	RO	0	0	Set to 1 when any bit in PCI_INT_SUM2 and the corresponding bit in the PCI_INT_ENB2 are both set to 1.
<1>	RML_WTO	R/W1C	0	0	Set to 1 when the RML does not receive a commit back from an RSL after sending a write command to an RSL.
<0>	RML_RTO	R/W1C	0	0	Set to 1 when the RML does not receive read data back from an RSL after sending a read command to an RSL.

NPI Interrupt Enable Register
NPI_INT_ENB

Used to enable the various interrupting conditions of IPD. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:62>	—	RAZ	—	—	Reserved
<61>	Q1_A_F	R/W	0	1	Enables NPI_INT_SUM[Q1_A_F] to generate an interrupt.
<60>	Q1_S_E	R/W	0	1	Enables NPI_INT_SUM[Q1_S_E] to generate an interrupt.
<59>	PDF_P_F	R/W	0	1	Enables NPI_INT_SUM[PDF_P_F] to generate an interrupt.
<58>	PDF_P_E	R/W	0	1	Enables NPI_INT_SUM[PDF_P_E] to generate an interrupt.
<57>	PCF_P_F	R/W	0	1	Enables NPI_INT_SUM[PCF_P_F] to generate an interrupt.
<56>	PCF_P_E	R/W	0	1	Enables NPI_INT_SUM[PCF_P_E] to generate an interrupt.
<55>	RDX_S_E	R/W	0	1	Enables NPI_INT_SUM[RDX_S_E] to generate an interrupt.
<54>	RWX_S_E	R/W	0	1	Enables NPI_INT_SUM[RWX_S_E] to generate an interrupt.
<53>	PNC_A_F	R/W	0	1	Enables NPI_INT_SUM[PNC_A_F] to generate an interrupt.
<52>	PNC_S_E	R/W	0	1	Enables NPI_INT_SUM[PNC_S_E] to generate an interrupt.
<51>	COM_A_F	R/W	0	1	Enables NPI_INT_SUM[COM_A_F] to generate an interrupt.
<50>	COM_S_E	R/W	0	1	Enables NPI_INT_SUM[COM_S_E] to generate an interrupt.
<49>	Q3_A_F	R/W	0	1	Enables NPI_INT_SUM[Q3_A_F] to generate an interrupt.
<48>	Q3_S_E	R/W	0	1	Enables NPI_INT_SUM[Q3_S_E] to generate an interrupt.
<47>	Q2_A_F	R/W	0	1	Enables NPI_INT_SUM[Q2_A_F] to generate an interrupt.
<46>	Q2_S_E	R/W	0	1	Enables NPI_INT_SUM[Q2_S_E] to generate an interrupt.
<45>	PCR_A_F	R/W	0	1	Enables NPI_INT_SUM[PCR_A_F] to generate an interrupt.
<44>	PCR_S_E	R/W	0	1	Enables NPI_INT_SUM[PCR_S_E] to generate an interrupt.
<43>	FCR_A_F	R/W	0	1	Enables NPI_INT_SUM[FCR_A_F] to generate an interrupt.
<42>	FCR_S_E	R/W	0	1	Enables NPI_INT_SUM[FCR_S_E] to generate an interrupt.
<41>	IOBDMA	R/W	0	1	Enables NPI_INT_SUM[IOBDMA] to generate an interrupt.
<40>	P_DPERR	R/W	0	1	Enables NPI_INT_SUM[P_DPERR] to generate an interrupt.
<39>	WIN_RTO	R/W	0	1	Enables NPI_INT_SUM[WIN_RTO] to generate an interrupt.
<38>	SPR17	R/W	0	1	Enables NPI_INT_SUM[SPR17] to generate an interrupt.
<37>	SPR16	R/W	0	1	Enables NPI_INT_SUM[SPR16] to generate an interrupt.
<36>	SPR26	R/W	0	1	Enables NPI_INT_SUM[SPR26] to generate an interrupt.
<35>	I0_PPERR	R/W	0	1	Enables NPI_INT_SUM[I0_PPERR] to generate an interrupt.
<34>	SPR15	R/W	0	1	Enables NPI_INT_SUM[SPR15] to generate an interrupt.
<33>	SPR14	R/W	0	1	Enables NPI_INT_SUM[SPR14] to generate an interrupt.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<32>	SPR25	R/W	0	1	Enables NPI_INT_SUM[SPR25] to generate an interrupt.
<31>	P0_PTOUT	R/W	0	1	Enables NPI_INT_SUM[P0_PTOUT] to generate an interrupt.
<30>	SPR13	R/W	0	1	Enables NPI_INT_SUM[SPR13] to generate an interrupt.
<29>	SPR12	R/W	0	1	Enables NPI_INT_SUM[SPR12] to generate an interrupt.
<28>	SPR24	R/W	0	1	Enables NPI_INT_SUM[SPR24] to generate an interrupt.
<27>	P0_PPERR	R/W	0	1	Enables NPI_INT_SUM[P0_PPERR] to generate an interrupt.
<26>	SPR11	R/W	0	1	Enables NPI_INT_SUM[SPR11] to generate an interrupt.
<25>	SPR10	R/W	0	1	Enables NPI_INT_SUM[SPR10] to generate an interrupt.
<24>	SPR23	R/W	0	1	Enables NPI_INT_SUM[SPR23] to generate an interrupt.
<23>	G0_RTOUT	R/W	0	1	Enables NPI_INT_SUM[G0_RTOUT] to generate an interrupt.
<22>	SPR9	R/W	0	1	Enables NPI_INT_SUM[SPR9] to generate an interrupt.
<21>	SPR8	R/W	0	1	Enables NPI_INT_SUM[SPR8] to generate an interrupt.
<20>	SPR22	R/W	0	1	Enables NPI_INT_SUM[SPR22] to generate an interrupt.
<19>	P0_PERR	R/W	0	1	Enables NPI_INT_SUM[P0_PERR] to generate an interrupt.
<18>	SPR7	R/W	0	1	Enables NPI_INT_SUM[SPR7] to generate an interrupt.
<17>	SPR6	R/W	0	1	Enables NPI_INT_SUM[SPR6] to generate an interrupt.
<16>	SPR21	R/W	0	1	Enables NPI_INT_SUM[SPR21] to generate an interrupt.
<15>	P0_RTOUT	R/W	0	1	Enables NPI_INT_SUM[P0_RTOUT] to generate an interrupt.
<14>	SPR5	R/W	0	1	Enables NPI_INT_SUM[SPR5] to generate an interrupt.
<13>	SPR4	R/W	0	1	Enables NPI_INT_SUM[SPR4] to generate an interrupt.
<12>	SPR20	R/W	0	1	Enables NPI_INT_SUM[SPR20] to generate an interrupt.
<11>	I0_OVERF	R/W	0	1	Enables NPI_INT_SUM[I0_OVERF] to generate an interrupt.
<10>	SPR3	R/W	0	1	Enables NPI_INT_SUM[SPR3] to generate an interrupt.
<9>	SPR2	R/W	0	1	Enables NPI_INT_SUM[SPR2] to generate an interrupt.
<8>	SPR19	R/W	0	1	Enables NPI_INT_SUM[SPR19] to generate an interrupt.
<7>	I0_RTOUT	R/W	0	1	Enables NPI_INT_SUM[I0_RTOUT] to generate an interrupt.
<6>	SPR1	R/W	0	1	Enables NPI_INT_SUM[SPR1] to generate an interrupt.
<5>	SPR0	R/W	0	1	Enables NPI_INT_SUM[SPR0] to generate an interrupt.
<4>	SPR18	R/W	0	1	Enables NPI_INT_SUM[SPR18] to generate an interrupt.
<3>	PO0_2SML	R/W	0	1	Enables NPI_INT_SUM[PO0_2SML] to generate an interrupt.
<2>	PCI_RSL	R/W	0	1	Enables NPI_INT_SUM[PCI_RSL] to generate an interrupt.
<1>	RML_WTO	R/W	0	1	Enables NPI_INT_SUM[RML_WTO] to generate an interrupt.
<0>	RML_RTO	R/W	0	1	Enables NPI_INT_SUM[RML_RTO] to generate an interrupt.

NPI Memory Access SubID Registers
NPI_MEM_ACCESS_SUBID(3..6)

Carries Read/Write parameters for core access to PCI memory that use NPI SubIDs 3..6. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:38>	—	RAZ	—	—	Reserved
<37>	SHORT	R/W	0	—	Generate CMD-6 on PCI when 1. Load operations from the cores to the corresponding Subid that are 32-bits or smaller generate a PCI Memory Read command, regardless of the NPI_PCI_READ_CMD[CMD_SIZE] value.
<36>	NMERGE	R/W	0	—	No merge.
<35:34>	ESR	R/W	0x0	—	Endian swap on read.
<33:32>	ESW	R/W	0x0	—	Endian swap on write.
<31>	NSR	R/W	0	—	No snoop on read.
<30>	NSW	R/W	0	—	No snoop on write.
<29>	ROR	R/W	0	—	Relax read on read.
<28>	ROW	R/W	0	—	Relax order on write.
<27:0>	BA	R/W	0x0	—	PCI address bits [63:36].

NPI / PCI Read Command Register
NPI_PCI_READ_CMD

Controls the type of read command sent. Write operations to this register are not ordered with write/read operations to the PCI memory space. To ensure that a write operation has completed, read the register before making an access that requires the value of this register to be updated (i.e. to PCI memory space).

Also, any previously issued read/write operations to PCI memory space that are still stored in the outbound FIFO use the value of this register after it has been updated. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:11>	—	RAZ	—	—	Reserved
<10:0>	CMD_SIZE	R/W	0x9	0x9	When the number bytes to be read is equal to or greater than this size, the PCI in PCI mode uses a Memory-Read-Multiple. This register has a value from 8 to 2048. Values of 0–7 are treated as a value of 2048.

NPI Number Of Descriptors Available For Outputs Registers
NPI_NUM_DESC_OUTPUT0/1

The size of the buffer/info pointer pair ring for output 0/1. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31:0>	SIZE	R/W	0x0	—	The size of the buffer/info-pointer pair ring.

NPI Base Address Input Registers
NPI_BASE_ADDR_INPUT0/1

The address to start reading instructions from for inputs 0/1. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	BADDR	R/W	0x0	—	The address from which to read the instruction for inputs 0/1. This address is 8-byte aligned, meaning address bits [2:0] are always 0x0.
<2:0>	—	RAZ	—	—	Reserved

NPI Size for Input Registers
NPI_SIZE_INPUT0/1

The address to start reading instructions from, for Input0/1. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31:0>	SIZE	R/W	0x10	—	Specifies the size of the instruction queue used by CN50XX. The value represents the number of instructions. A value of 0x0 in this field is illegal.

PCI Read Time-out Register
PCI_READ_TIMEOUT

The address to start reading instructions from for input 3. See Table 9–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31>	ENB	R/W	0	1	Enable the use of the time-out function.
<30:0>	CNT	R/W	0x2710	0x2710	The number of core clock cycles to wait after issuing a read request to the PNI before setting a time-out and not expecting the data to return. This is considered a fatal condition by the NPI.

NPI Base Address Output Registers
NPI_BASE_ADDR_OUTPUT0/1

The address to start reading instructions from for outputs 0/1. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	BADDR	R/W	0x0	—	The address from which to read the instruction for output 0/1. This address is 8-byte aligned, meaning address bits [2:0] are always 0x0.
<2:0>	—	RAZ	—	—	Reserved

NPI PCI Burst Size Register
NPI_PCI_BURST_SIZE

Controls the number of words the NPI will attempt to read/write to/from the PCI. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:14>	—	RAZ	—	—	Reserved
<13:7>	WR_BRST	R/W	0x10	0x40	The number of 8-byte words to write to PCI in any one write operation. A 0x0 is equal to 0x80. This value is used the packet reads and is clamped at a maximum of 0x70 for DMA write operations. DMA write values greater than 0x70 are equal to 0x70.
<6:0>	RD_BRST	R/W	0x11	0x40	Number of 8-byte words to read from PCI in any one read operation. Legal values are 0x1 to 0x7F, where a 0 is treated as a 1. For reading of packet data, value is limited to 0x40. This value does not control the size of a read caused by an IOBDMA from a core.

NPI D/I Buffer Sizes For Output Registers
NPI_BUFF_SIZE_OUTPUT0/1

The size in bytes of the data buffer and information buffer for outputs 0/1. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:23>	—	RAZ	—	—	Reserved
<22:16>	ISIZE	R/W	0x0	—	The number of bytes to move to the info-pointer from the front of the packet. Legal values are 0–120.
<15:0>	BSIZE	R/W	0x400	—	The size in bytes of the area pointed to by buffer pointer for output packet data.

NPI Output Control Register
NPI_OUTPUT_CONTROL

The address to start reading instructions from for Output 3. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved
<47:45>	SPR5	R/W	0x0	0x0	Spare.
<44>	P0_BMODE	R/W	0	0	When set to 1, the PCI_PKTS_SENT0 register is updated with the number of bytes in the packet sent; when 0, the register has a value of 1 added.
<43:32>	SPR4	R/W	0x0	0x0	Spare
<31:30>	O0_ES	R/W	0x0	—	Endian swap for output0 data. 0x0 = No swizzle 0x1 = Byte swizzle (per-quadword), 0x2 = Byte swizzle (per-longword) 0x3 = Longword swizzle
<29>	O0_NS	R/W	0x0	—	No-snoop mode enable for output0 data.
<28>	O0_RO	R/W	0x0	—	Relaxed ordering enable for output0 data.
<27:25>	SPR3	R/W	0x0	0x0	Spare.
<24>	O0_CSRM	R/W	0	1	When set to 1, the bits[63:60] of the address for write-packet data come from DPTR0[63:60]; and the ES, NS, and RO bits come from O0_ES, O0_NS, and O0_RO. When set to 0, the address bits [63:60] of the address the packet will be written to come from the O0_ES[1:0], O0_NS, and O0_RO respectively; and the ES, NS, and RO bits come from DPTR0[63:60] respectively.
<23:20>	SPR2	R/W	0x0	0x0	Spare.
<19:17>	SPR1	R/W	0x0	0	Spare.
<16>	IPTR_O0	R/W	0x0	1	Uses the info-pointer to store length and data for Output0.
<15:4>	SPR0	R/W	0x0	0x0	Spare.
<3:2>	ESR_SL0	R/W	0x0	—	Indicates the endian-swap mode for Slist0 reads.
<1>	NSR_SL0	R/W	0	—	Enables no-snoop mode for Slist0 reads.
<0>	ROR_SL0	R/W	0	—	Enables relaxed ordering for Slist0 reads.

NPI DMA Low-Priority Instruction-Buffer Starting-Address Register
NPI_LOWP_IBUFF_SADDR

The address to start reading Instructions from for LOWP. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	—	—	Reserved
<35:0>	SADDR	R/W	0x0	—	The starting address to read the first instruction.

NPI DMA High Priority Instruction Buffer Starting Address
NPI_HIGHP_IBUFF_SADDR

The address to start reading instructions from for HIGHP. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	—	—	Reserved
<35:0>	SADDR	R/W	0x0	—	The starting address to read the first instruction.

NPI Low-Priority Doorbell Register
NPI_LOWP_DBELL

The doorbell register for the low-priority DMA queue. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	DBELL	R/W	0x0	—	The value written to this register is added to the number of 8-byte words to be read and processed for the low-priority DMA queue.

NPI High Priority Doorbell Register
NPI_HIGHP_DBELL

The doorbell register for the high-priority DMA queue. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	DBELL	R/W	0x0	—	The value written to this register is added to the number of 8-byte words to be read and processed for the high-priority DMA queue.

NPI DMA Control Register
NPI_DMA_CONTROL

Controls operation of the DMA I/O of the NPI. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	—	—	Reserved
<35>	B0_LEND	R/W	0	0	When set to 1 and the NPI is in the mode to write 0x0 to L2C when a DMA is done, the address to be written to is treated as a little-endian address.
<34>	DWB_DENB	R/W	0	0	When set to 1, the NPI sends a value in the DWB field for a free-page operation for the memory that contained the data in CN50XX.
<33:25>	DWB_ICHK	R/W	0x0	0x0	When instruction chunks for DMA operations are freed, this value is used for the DWB field of the operation.
<24:22>	FPA_QUE	R/W	0x0	0x0	The FPA queue that the instruction-chunk page is returned to when used.
<21>	O_ADD1	R/W	0	1	When set to 1, 1 is added to the DMA counters; when set to 0, the number of bytes in the DMA transfer is added to the count register.
<20>	O_RO	R/W	0	0	Relaxed ordering mode for DMA.
<19>	O_NS	R/W	0	0	No-snoop mode for DMA.
<18:17>	O_ES	R/W	0x0	—	Endian-swap mode for DMA. 0x0 = No swizzle 0x1 = Byte swizzle (per-quadword), 0x2 = Byte swizzle (per-longword) 0x3 = Longword swizzle
<16>	O_MODE	R/W	0	1	Select PCI_POINTER MODE to be used. 1 = use pointer values for address, and register values for RO, ES, NS 0 = use register values for address, and pointer values for RO, ES, NS.
<15>	HP_ENB	R/W	0	1	Enables the high-priority DMA. When this bit is cleared to 0, the value in NPI_HIGHP_IBUFF_SADDR is reinitialized. When this bit is 0, the value in CSIZE is reloaded for the high-priority DMA engine.
<14>	LP_ENB	R/W	0	1	Enables the low-priority DMA. When this bit is cleared to 0, the value in NPI_LOWP_IBUFF_SADDR is reinitialized. When this bit is 0, the value in CSIZE is reloaded for the low-priority DMA engine.
<13:0>	CSIZE	R/W	0x0	—	The size in words of the DMA instruction chunk. This value should only be written once. After writing this value, a new value will not be recognized until the end of the DMA I-chunk is reached.

NPI Configuration For PCI Arbiter Register
NPI_PCI_INT_ARB_CFG

Controls operation of the internal PCI arbiter. This register should only be written when PRST# is asserted. NPI_PCI_INT_ARB_CFG[EN] should only be set when CN50XX is a host. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved
<4>	EN	R/W	0	—	Internal arbiter enable.
<3>	PARK_MOD	R/W	0	—	Bus park mode: 0 = park on last, 1 = park on device.
<2:0>	PARK_DEV	R/W	0x0	—	Bus park device: 0–3 = external device, 4 = CN50XX.

NPI Input Control Register
NPI_INPUT_CONTROL

Control for reads for the gather list and instructions. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:22>	—	RAZ	—	—	Reserved
<21:9>	PBP_DHI	R/W	0x0	—	When NPI_PORT32/33_INSTR_HDR[PBP] is set to 1, this field is used to calculate a DPTR.
<8>	D_NSR	R/W	0	—	Enables no-snoop mode for reading of gather data.
<7:6>	D_ESR	R/W	0x0	—	Endian-swap mode for reading of gather data. 0x0 = No swizzle 0x2 = Byte swizzle (per-longword) 0x1 = Byte swizzle (per-quadword), 0x3 = Longword swizzle
<5>	D_ROR	R/W	0	—	Enables relaxed ordering for reading of gather data.
<4>	USE_CSR	R/W	0	1	When set to 1, the CSR value is used for ROR, ESR, and NSR. When cleared to 0, the value in DPTR is used. In turn, the bits not used for ROR, ESR, and NSR are used for bits [63:60] of the address used to fetch packet data.
<3>	NSR	R/W	0	—	Enables no-snoop mode for reading of gather list and gather instruction.
<2:1>	ESR	R/W	0x0	—	Endian-swap mode for reading of gather list and gather instruction. 0x0 = No swizzle 0x2 = Byte swizzle (per-longword) 0x1 = Byte swizzle (per-quadword), 0x3 = Longword swizzle
<0>	ROR	R/W	0	—	Enables relaxed ordering for reading of gather list and gather instruction.

NPI Low Priority DMA Counts Register
NPI_DMA_LOWP_COUNTS

Values for determining the number of instructions for low-priority DMA in the NPI. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:39>	—	RAZ	—	—	Reserved
<38:32>	FCNT	RO	0x0	0x0	Number of words in the instruction FIFO.
<31:0>	DBELL	RO	0x0	0x0	Number of available words of instructions to read.

NPI High Priority DMA Counts Register
NPI_DMA_HIGHP_COUNTS

Values for determining the number of instructions for high-priority DMA in the NPI. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:39>	—	RAZ	—	—	Reserved
<38:32>	FCNT	RO	0x0	0x0	Number of words in the instruction FIFO.
<31:0>	DBELL	RO	0x0	0x0	Number of available words of instructions to read.

NPI Low Priority DMA Next Ichunk Address Register
NPI_DMA_LOWP_NADDR

The place NPI will read the next Ichunk data from. This is valid when state is 0. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:40>	—	RAZ	—	—	Reserved
<39:36>	STATE	RO	0x0	0x0	The DMA instruction engine state vector. Typical value is 0x0 (IDLE).
<35:0>	ADDR	RO	0x0	—	The next L2C address to read DMA instructions from for the low-priority DMA engine.

NPI High Priority DMA Next Ichunk Address Register
NPI_DMA_HIGHP_NADDR

The place NPI will read the next Ichunk data from. This is valid when state is 0. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:40>	—	RAZ	—	—	Reserved
<39:36>	STATE	RO	0x0	0x0	The DMA instruction engine state vector. Typical value is 0x0 (IDLE).
<35:0>	ADDR	RO	0x0	—	The next L2C address to read DMA instructions from for the high-priority DMA engine.

NPI Port Instruction Counts For Packets Out Registers
NPI_P0/1_PAIR_CNTS

Used to determine the number of instruction in the NPI and to be fetched for output-packets. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:37>	—	RAZ	—	—	Reserved
<36:32>	FCNT	RO	0x0	0x0	16 - number entries in the data/info pair FIFO.
<31:0>	AVAIL	RO	0x0	0x0	Doorbell count to be read.

NPI Port Data-Buffer Pair Next-Read Address Registers
NPI_P0/1_DBPAIR_ADDR

Contains the next address to read for the Port 0/1 data/buffer pair. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	—	RAZ	—	—	Reserved
<62:61>	STATE	RO	0x0	0x0	State. POS state-machine vactor.
<60:0>	NADDR	RO	0x0	—	Bits [63:3] of the next data/info pair to read, valid only when STATE = 0.

NPI Port Instruction Counts For Packets In Registers
NPI_P0/1_INSTR_CNTS

Used to determine the number of instruction in the NPI and to be fetched for input packets. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:38>	—	RAZ	—	—	Reserved
<37:32>	FCNT	RO	0x0	0x0	Number of entries in the instruction FIFO.
<31:0>	AVAIL	RO	0x0	0x0	Doorbell count to be read.

NPI Port Instruction Next Read Address Registers
NPI_P0/1_INSTR_ADDR

Contains the next address to read for Port 0/1 instructions. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:61>	STATE	RO	0x0	0x0	State.
<60:0>	NADDR	RO	0x0	—	Bits [63:3] of the next instruction to read, valid only when STATE = 0.

NPI Window Read Time-out Register
NPI_WIN_READ_TO

Provides the number of core-clock cycles to wait before timing out on a WINDOW-READ to the IOB. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31:0>	TIME	R/W	0x0	0x20000	Time to wait, in core clock cycles. A value of 0x0 causes no time-outs.

Debug Data Register
DBG_DATA

Value returned on the debug-data lines from the RSLs. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:31>	—	RAZ	—	—	Reserved
<30:28>	PLL_MUL	RO	—	—	Value of PLL_MUL<2:0> pins at time PLL_DCOK asserts.
<27:23>	—	RAZ	—	—	Reserved
<22:18>	C_MUL	RO	—	—	Core PLL multiplier sampled at assertion of PLL_DCOK.
<17>	DSEL_EXT	R/W	1	0	Allows changes in the external pins to set the debug select value.
<16:0>	DATA	RO	0x0	—	Value on the debug data lines.

Port Backpressure Control Register
NPI_PORT_BP_CONTROL

Enables port-level backpressure. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	—	—	Reserved
<5:4>	BP_ON	RO	0x0	0x0	Ports 33, 32 port-level backpressure applied.
<3:0>	ENB	R/W	0xF	0xF	Enables port-level back pressure from the IPD.

NPI Port Instruction Header Registers
NPI_PORT32/33_INSTR_HDR

Contains bits[62:42] of the instruction header for port 32/33. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:44>	—	RAZ	—	—	Reserved
<43>	PBP	R/W	0	—	Enable packet-by-packet mode.
<42:38>	—	R/W	0x0	—	Reserved
<37:36>	RPARMODE	R/W	0x0	—	Raw parse mode. Used when the packet is raw and PBP = 0.
<35>	—	R/W	0x0	—	Reserved
<34:28>	RSKP_LEN	R/W	0x8	—	Raw skip length. Used when the packet is raw and PBP = 0.
<27:22>	—	R/W	0x0	—	Reserved
<21>	USE_IHDR	R/W	0	—	Use instruction header. When set to 1, the instruction header is sent as part of the packet data, regardless of the value of bit [63] of the instruction header. USE_IHDR must be set whenever PBP is set.
<20:16>	—	R/W	0x0	—	Reserved
<15:14>	PAR_MODE	R/W	0x0	—	Parse mode. Used when USE_IHDR is set, packet is not raw, and PBP is not set.
<13>	—	R/W	0x0	—	Reserved
<12:6>	SKP_LEN	R/W	0x0	—	Skip length. Used when USE_IHDR is set, packet is not raw, and PBP is not set.
<5:0>	—	R/W	0x0	—	Reserved

NPI BIST Status Register
NPI_BIST_STATUS

Provides the results from the BIST runs of NPI memories. See Table 9–6 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description[1, 2]
<63:20>	—	RAZ	—	—	Reserved
<19>	CSR_BS	RO	0	0	BIST status for the CSR_FIFO
<18>	DIF_BS	RO	0	0	BIST status for the DIF_FIFO
<17>	RDP_BS	RO	0	0	BIST status for the RDP_FIFO
<16>	PCNC_BS	RO	0	0	BIST status for the PCN_CNT_FIFO
<15>	PCN_BS	RO	0	0	BIST status for the PCN_FIFO
<14>	RDN_BS	RO	0	0	BIST status for the RDN_FIFO
<13>	PCAC_BS	RO	0	0	BIST status for the PCA_CMD_FIFO
<12>	PCAD_BS	RO	0	0	BIST status for the PCA_DATA_FIFO
<11>	RDNL_BS	RO	0	0	BIST status for the RDN_LENGTH_FIFO
<10>	PGF_BS	RO	0	0	BIST status for the PGF_FIFO
<9>	PIG_BS	RO	0	0	BIST status for the PIG_FIFO
<8>	POF0_BS	RO	0	0	BIST status for the POF0_FIFO
<7>	POF1_BS	RO	0	0	BIST status for the POF1_FIFO
<6>	—	RAZ	—	—	Reserved
<5>	—	RAZ	—	—	Reserved
<4>	POS_BS	RO	0	0	BIST status for the POS_FIFO
<3>	NUS_BS	RO	0	0	BIST status for the NUS_FIFO
<2>	DOB_BS	RO	0	0	BIST status for the DOB_FIFO
<1>	PDF_BS	RO	0	0	BIST status for the PDF_FIFO
<0>	DPI_BS	RO	0	0	BIST status for the DPI_FIFO

1. Registers at address 0x1000 -> 0x17FF are PNI. Start at 0x100 into range. These are shifted by 2 to the left to make address.
2. Registers at address 0x1800 -> 0x18FF are CFG. These are shifted by 2 to the left to make address.

Chapter 10

Timer

This chapter contains the following subjects:

Overview

10.1 Timer Features

Figure 10–1 shows a conceptual illustration of the CN50XX timer.

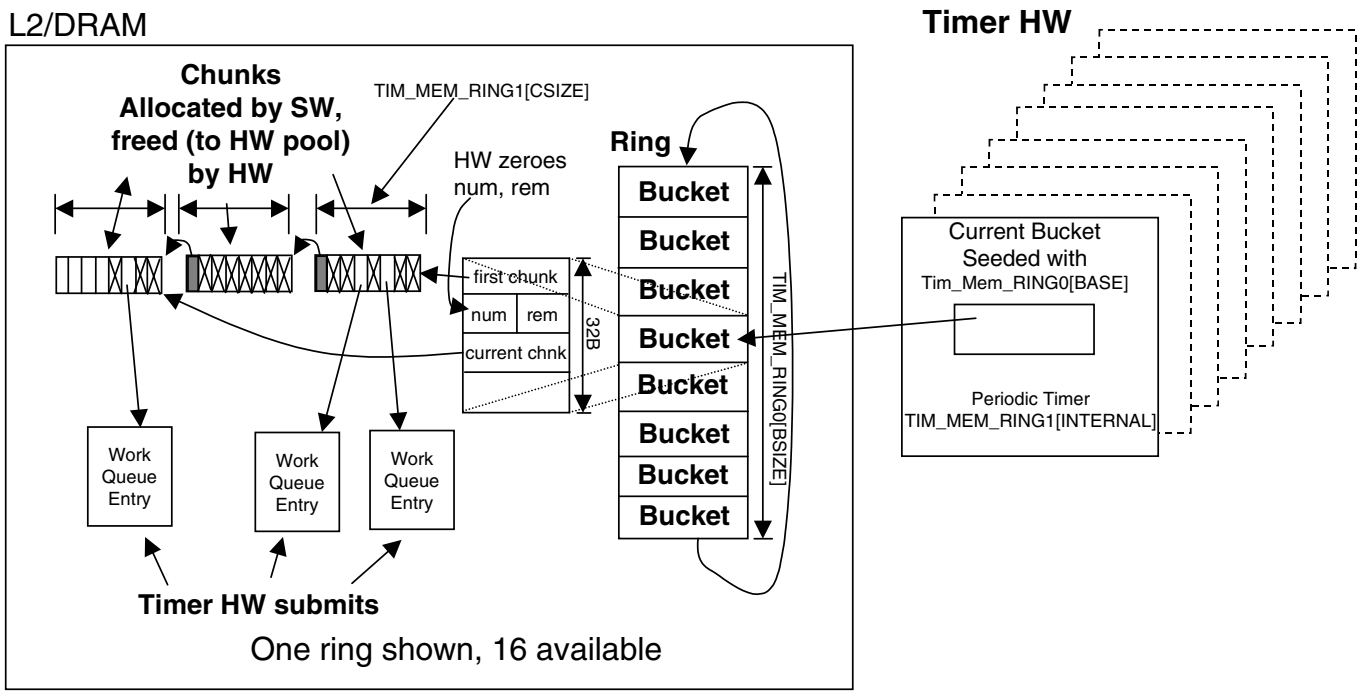


Figure 10–1 CN50XX Timer

The timer consists of 16 bucketed rings held in L2/DRAM.

- Software can program a separate ring per core.
- Each bucket corresponds to a different time slice.
- Each ring has a programmable number of buckets.
- Each ring has a programmable time interval between buckets.

At each periodic bucket time expiration, hardware processes the next bucket.

- Each timer entry within a bucket may be a work-queue entry pointer that hardware submits
- A bucket is a chunked list, and hardware frees chunks (to a hardware pool) after using them
- Hardware also resets the bucket data structure

The hardware traverses up to 80 million timer entries per second.

The software inserts work-queue entries into the appropriate bucket.

- Software allocates bucket chunks as needed
- Software can remove added entries from the bucket later by rewriting the entry to NULL (before the hardware processes the bucket)

10.2 Timer Support

The CN50XX support for timers is a mix of software and hardware.

Software creates bucketed timer-list entries and can later invalidate them. Each timer-list entry is a pointer to a work-queue entry that was created by software.

See Chapter 5Chapter 5. The work-queue entry can be as small as 16 bytes.

Upon expiration, hardware traverses the timer lists created previously by software. For all entries in the list that are still valid, hardware schedules the work. Hardware also initializes the list for subsequent adds. Because there are 16 timer lists, software can allocate one list for each core and avoid synchronization between cores.

See Figure 10–2. Each bucket corresponds to a timer expiration at a different time. The next time increment is the next sequential entry in the bucket array, wrapping around. The amount of time between buckets is also programmable.

The space that held the timer entries is dynamically allocated by the cores and freed by hardware as it traverses the list. The list is allocated in chunks.

The rings are configured via the TIM_MEM_RING0 and TIM_MEM_RING1 CSRs. Each ring has the following separate items:

- Enable bit (TIM_MEM_RING1[ENA])
- Time between bucket traversals (TIM_MEM_RING1[INTERVAL])
- Base pointer (i.e. pointer to the first bucket) (TIM_MEM_RING0[BASE])
- Number of buckets in the ring (minus 1) (TIM_MEM_RING0[BSIZE])
- Chunk size (i.e. number of work-queue entry pointers in each buffer chunk linked to the base bucket data structure) (TIM_MEM_RING1[CSIZE])
- Pool to free the chunks to (see Chapter 6) (TIM_MEM_RING1[CPOOL])

TIM_MEM_RING0[RID] and TIM_MEM_RING1[RID] select the ring when programming the above values. The following describes the data structures recognized by the timer hardware:

```
// must be 128-byte aligned
struct timer_entry_chunk {
   uint64                 entries[TIM_MEM_RING1[CSIZE]-1]; // if all zeroes, the entry is invalid
   struct timer_entry_chunk *next_ptr;
};

// must be 32-byte aligned
// this description assumes big-endian
struct |{
   struct   timer_entry_chunk *first_chunk;
   uint32   num_entries;                    // zeroed by HW after traversing list
   uint32   chunk_remainder;                // zeroed by HW after traversing list

   // the remaining 16 bytes are not touched by hardware
   struct   timer_entry_chunk *current_chunk;
   uint64   pad;
};
```

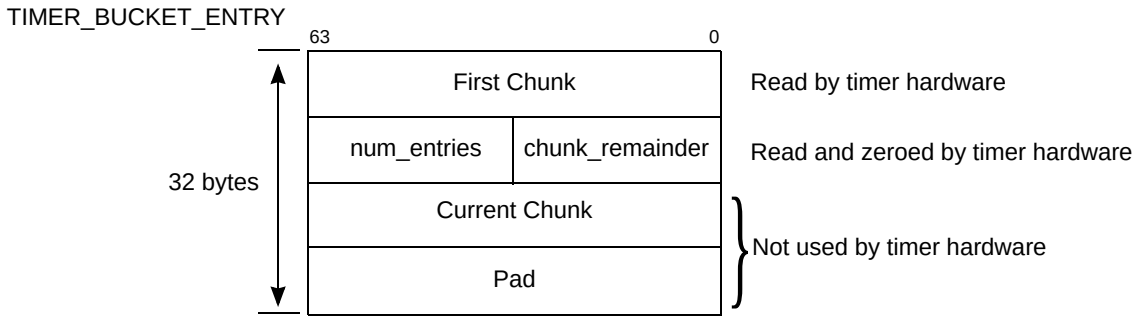


Figure 10–2 Bucket Data Structure

The following is a sample add routine:

```
// must be 32-byte aligned
TIMER_BUCKET_ENTRY buckets[TIM_MEM_RING0[BSIZE]+1];

Here is a possible routine to add a work queue pointer to a timer bucket list:

uint64 *add_bucket(uint64 wqptr, int bucket_id) {
  TIMER_BUCKET_ENTRY *e = &buckets[bucket_id];
  uint64 *res;
  uint32 remainder = e->chunk_remainder;
  e->num_entries++;
  if(remainder == 0) {
     TIMER_ENTRY_CHUNK *ptr = malloc_chunk();
     if(cur == NULL) e->first_chunk = ptr;
     else e->current_chunk->next_ptr = ptr;
     e->current_chunk = ptr;
     res = &(ptr->entries[0]);
     remainder = TIM_MEM_RING1[CSIZE] - 1;
  }
  else {
     res = &(e->current_chunk->entries[TIM_MEM_RING1[CSIZE] - remainder]);
     remainder --;
  }
  *res = wqptr;
  e->chunk_remainder = remainder;
  return(res);
}
```

At some later point, but not while or after the hardware traverses the list, software can invalidate a timer entry by writing it to all 0s. In that case, hardware does not create the work-queue entry.

Note that the timer hardware zeroes the num_entries and chunk_remainder fields while it traverses the list associated with a bucket. The timer hardware also reads the first_chunk field, but never references the current_chunk field.

10.3 Software Responsibilities

There are two specific race conditions that must be managed by software:

1. Software must not add to a bucket/list when the hardware might be processing the list. This is because hardware may miss the addition.

   In practice this means two things:

   - A timer entry that expires too close to the current time cannot be added.
   - A timer entry that expires too far from the current time cannot be added (i.e. you must not wrap around the buckets)

2. Software must not invalidate a timer entry when hardware might be processing the list that contains the timer entry.

This is because hardware may have already freed the memory that contains the timer entry. (In which case, hardware will already have scheduled it.) It is also possible that hardware has not freed the memory, but has already traversed past the relevant entry.

In practice this means that a timer entry that is going to expire “too soon” cannot be invalidated. Hardware schedules the work-queue entry anyway in this case, and software must have the appropriate data structures available to prevent it from performing any action.

A good way to prevent any action in this case is to store the expiration time of the timer in some data structure that is accessible from the work queue entry. If the stored timer expiration time is later than the current time, when hardware schedules the work queue entry to a Core, no action should be performed.

Note also that in addition to invalidation, another common timer operation is to reschedule a timer entry further into the future. This should be handled by invalidating the prior timer entry and creating a new one. If the expiration time of the current timer is “too close” for a simple invalidation, software may choose to wait and create the new entry after hardware schedules the prior one. In any case, software must not schedule the same work queue entry simultaneously. Waiting for hardware to schedule the prior entry will allow software to use only a single work queue entry in this case. Without the wait, the new entry will likely need a new work queue entry.

10.4 Timer Registers

The timer registers are shown in Table 10-1.

Table 10-1 Timer Registers

Register	Address	CSR Type[1]	Detailed Description
TIM_REG_FLAGS	0x0001180058000000	RSL	See page 455
TIM_REG_READ_IDX	0x0001180058000008	RSL	See page 455
TIM_REG_BIST_RESULT	0x0001180058000080	RSL	See page 455
TIM_REG_ERROR	0x0001180058000088	RSL	See page 456
TIM_REG_INT_MASK	0x0001180058000090	RSL	See page 456
TIM_MEM_RING0	0x0001180058001000	RSL	See page 456
TIM_MEM_RING1	0x0001180058001008	RSL	See page 457
TIM_MEM_DEBUG0	0x0001180058001100	RSL	See page 457
TIM_MEM_DEBUG1	0x0001180058001108	RSL	See page 457
TIM_MEM_DEBUG2	0x0001180058001110	RSL	See page 458

1. RSL-type registers are accessed indirectly across the I/O Bus.

Timer Flags Register
TIM_REG_FLAGS

See Table 10-1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	x	0	Must be 0x0 (MBZ).
<2>	RESET	RAZ	0	0	Reset one-shot pulse for free-running structures.
<1>	ENA_DWB	R/W	0	0	Enables non-zero don't-write-back operations. When set, this bit enables the use of don't-write-backs during the buffer freeing operations
<0>	ENA_TIM	R/W	0	0	Enables the timer section. When set, this bit places the timer into normal operation.

Timer Read Index Register
TIM_REG_READ_IDX

This register provides the read index during a CSR read operation to any of the CSRs that are physically stored as memories (the names of these CSRs begin with the prefix TIM_MEM_). IDX[7:0] is the read index, and INC[7:0] is an increment that is added to IDX[7:0] after any CSR read. The intended use is to initially write this CSR such that IDX = 0 and INC = 1. Then, the entire contents of a CSR memory can be read with consecutive CSR read commands. See Table 10-1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	X	0x0	MBZ.
<15:8>	INC	R/W	0x0	0x0	Increment to add to current index for next index.
<7:0>	IDX	R/W	0x0	0x0	Index to use for next memory CSR read.

Timer BIST Result Register
TIM_REG_BIST_RESULT

This register provides access to the internal BIST results. Each bit is the BIST result of an individual memory (per bit, 0 = pass and 1 = fail). See Table 10-1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	X	0	MBZ.
<3:2>	STA	RO	X	0	BIST result of the STA memories (0x0 = pass, any other value = fail)
<1>	NCB	RO	X	0	BIST result of the IOB memories (0 = pass, 1 = fail)
<0>	CTL	RO	X	0	BIST result of the CTL memories (0 = pass, 1 = fail)

Timer Error Register
TIM_REG_ERROR

This register indicates if a ring is in error (i.e. its interval has elapsed more than once without having been serviced). See Table 10-1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	X	0	MBZ.
<15:0>	MASK	R/W1C	0	0	Bit mask indicating which rings are in error.

Timer Interrupt Mask Register
TIM_REG_INT_MASK

This register provides mask bits to enable interrupts when an error is shown in TIM_REG_ERROR.

See Table 10-1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	X	0	MBZ.
<15:0>	MASK	R/W	0	0	Interrupt-enable bit mask corresponding to the error mask in TIM_REG_ERROR.

Timer Ring0 Register
TIM_MEM_RING0

This CSR, which provides configuration for the 16 rings, is actually a 16-entry memory. The TIM_REG_READ_IDX register must be configured before any CSR read operations to this address can be performed. See Table 10-1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:55>	—	RAZ	X	0	MBZ
<54:24>	BASE	R/W	0	0	Pointer to bucket 0. This field is a 32-byte-aligned 36-bit pointer (bits<35:0>). Only bits <35:5> are stored because bits <4:0> are always 0s.
<23:4>	BSIZE	R/W	0	0	Number of buckets – 1 (i.e. number of buckets = BSIZE + 1).
<3:0>	RID	R/W	0	0	Ring ID.

Timer Ring1 Register
TIM_MEM_RING1

This CSR, which provides configuration for the 16 rings, is actually a 16-entry memory. The TIM_REG_READ_IDX register must be configured before any CSR read operations to this address can be performed. See Table 10-1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:43>	—	RAZ	X	0x0	MBZ.
<42>	ENA	R/W	0	0	Ring timer enable.
<41:39>	CPOOL	R/W	0x0	0x0	Free list used to free chunks
<38:26>	CSIZE	R/W	0x0	0x0	Number of words per chunk. It is illegal to program CSIZE < 16.
<25:4>	INTERVAL	R/W	0x0	0x0	Timer interval – 1, measured in 1024 cycle ticks.
<3:0>	RID	R/W	0x0	0x0	Ring ID.

Timer Debug0 Register
TIM_MEM_DEBUG0

This CSR, which provides internal per-ring state intended for debug use only, is actually a 16-entry memory. The TIM_REG_READ_IDX register must be configured before any CSR read operations to this address can be performed. See Table 10-1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RO	X	0x0	MBZ.
<47>	ENA	RO	X	0	Ring timer enable.
<46>	—	RO	X	0	MBZ.
<45:24>	COUNT	RO	X	0x0	Current count.
<23:22>	—	RO	X	0x0	MBZ.
<21:0>	INTERVAL	RO	X	0x0	Timer interval – 1.

Timer Debug1 Register
TIM_MEM_DEBUG1

This CSR, which provides internal per-ring state intended for debug use only, is actually a 16-entry memory. The TIM_REG_READ_IDX register must be configured before any CSR read operations to this address can be performed. See Table 10-1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:51>	BUCKET	RO	X	0x0	Specifies bits <12:0> of the current bucket.
<50:20>	BASE	RO	X	0x0	Pointer bits <35:5> to bucket 0.
<19:0>	BSIZE	RO	X	0x0	Number of buckets – 1.

Timer Debug2 Register
TIM_MEM_DEBUG2

This CSR, which provides internal per-ring state intended for debug use only, is actually a 16-entry memory. The TIM_REG_READ_IDX register must be configured before any CSR read operations to this address can be performed. See Table 10-1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RO	X	0	MBZ.
<31:24>	—	RO	X	0	MBZ.
<23:21>	CPOOL	RO	X	0	Free list used to free chunks.
<20:8>	CSIZE	RO	X	0	Number of words per chunk.
<7:7>	—	RO	X	0	MBZ.
<6:0>	BUCKET	RO	X	0	Specifies bits <19:13> of the current bucket.

Chapter 11

Central Interrupt Unit (CIU)

This chapter contains the following subjects:

Overview

CN50XX's CIU implements the following functions:

- Central interrupt collection, both internal and external (CIU_INT*n*_SUM0/1, CIU_INT0/1_SUM4).
- Central interrupt selection and distribution to cores, and to PCI INTA when CN50XX is not a PCI host (CIU_INT*n*_EN0/1, CIU_INT0/1_EN4_0/1).
- Per-core mailbox registers (CIU_MBOX_SET*n*, CIU_MBOX_CLR*n*).
- Per-core watchdog timers (CIU_WDOG*n*, CIU_PP_POKE*n*)
- Four general timers (CIU_TIM*n*)
- Core availability and reset (CIU_FUSE and CIU_PP_RST)
- Core debug-mode observability (CIU_PP_DBG)
- Core debug-interrupt generation (CIU_DINT)
- Core nonmaskable interrupt generation (CIU_NMI)
- Chip soft-reset initiation (CIU_SOFT_RST, CIU_SOFT_BIST)
- Global-stop (GSTOP) bit (CIU_GSTOP). Used for multi-core debug. Refer to Section 4.12 for more details on GSTOP usage.

11.1 Central Interrupt Collection and Distribution

Figures 11–1, 11–2, and 11–3 show CN50XX's entire interrupt-distribution network, some of which resides in the CIU. Figure 11–1 shows that the CIU distributes a total of 7 interrupts – three per core and one more for PCI INTA/MSI generation. The three interrupts for each core become Cause[IP4,IP3,IP2] local to the core. With three separate interrupt destinations per core, different interrupts destined for the core can be prioritized differently by the core-interrupt handlers.

Notes:

1. The variables ***x*** and ***y*** represent the selected interrupts. ***x*** is a value between 0 and 3, or 32 (used with *_SUM0 registers) and ***y*** is 0 or 1 (used with *_SUM4 registers).
2. The variable ***z*** represents the GPIO pin number. It is a value between 0 and 15
3. The variable ***n*** represents the number of cores. It is either 0 or 1
4. The sections of CIU_INT(***x/y***)_SUM0/4 that are shown in white are latched. All other bits pass right through.

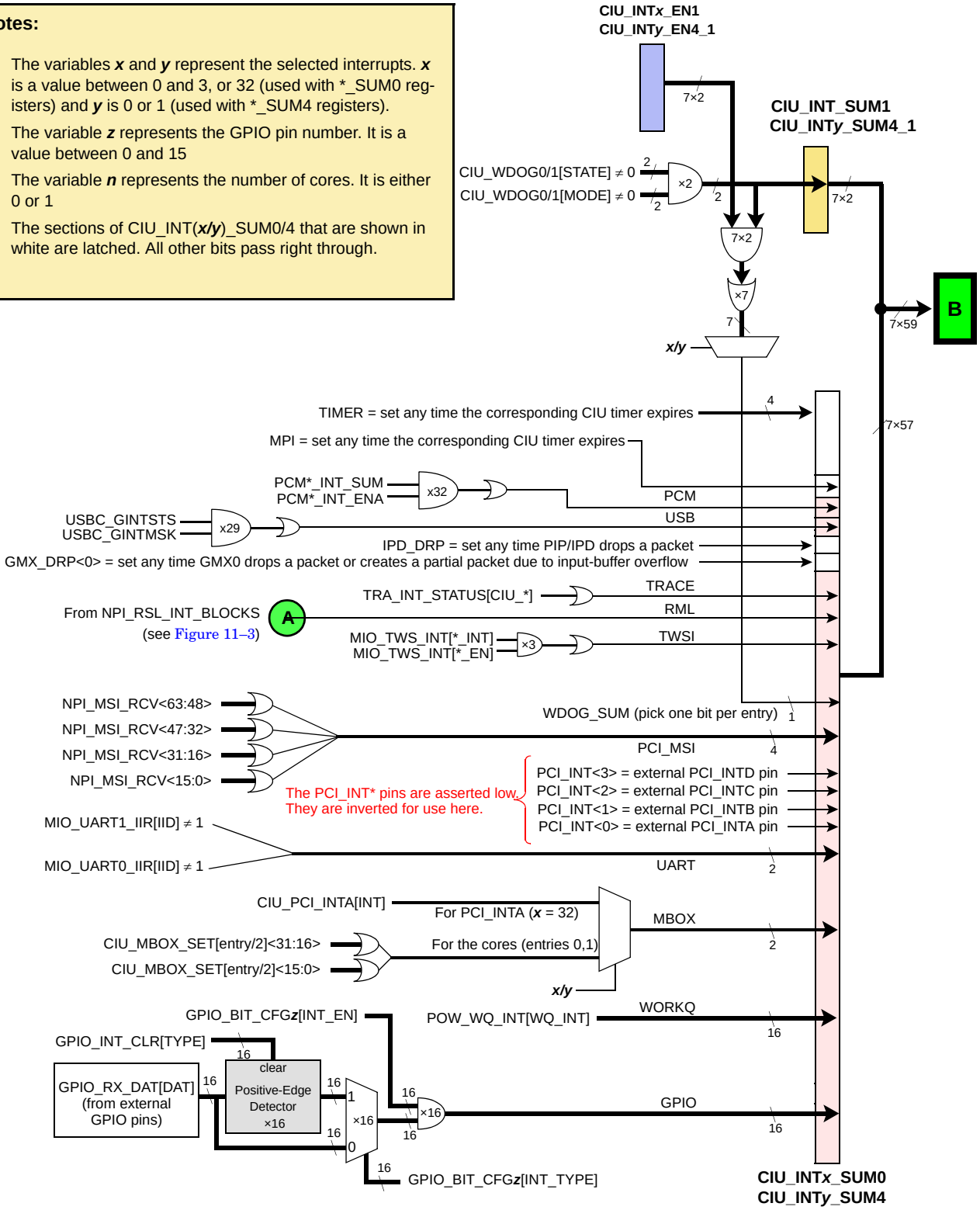


Figure 11–1 CN50XX Interrupt Distribution

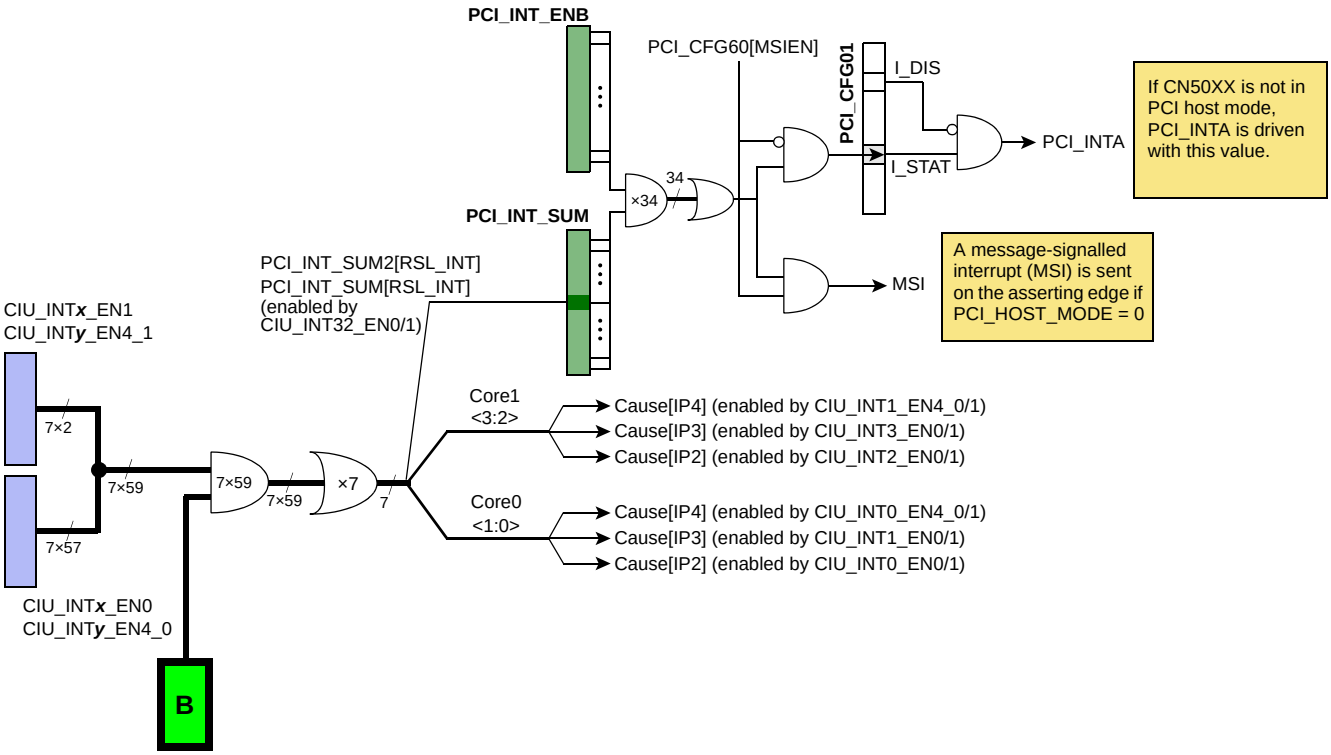


Figure 11–2 PCI Interrupt Distribution

As Figure 11–3 shows, there are more than 250 possible interrupt sources on CN50XX. Most of these sources are accumulated and flow into the 59-bit summary vector CIU_INT*n*_SUM0/1/4. The 59-bit summary directly contains the remainder of the interrupt sources; those direct interrupts can be more quickly discerned and processed. The corresponding 59 bits of interrupt enable vector in CIU_INT*n*_EN0/1 allow the summarized interrupts to reach their ultimate destination.

CN50XX configuration registers contain a different 59-bit enable (CIU_INT*n*_EN*, CIU_INT*n*_EN4_0/1) for each of the seven interrupt destinations, so each destination can include arbitrary and independent combinations of the 59 sources. Any interrupt in the 59-bit summary can be directed to any combination of the interrupt destinations.

There is a different summary vector for each interrupt destination. These summary vectors differ only in two fields:

- CIU_INT*n*_SUM0[MBOX].
- CIU_INT*n*_SUM0[WDOG_SUM].

All other fields in CIU_INT*n*_SUM0/1/4 are identical for the different destinations. CIU_INT*n*_SUM0[MBOX] is a summary of the mailbox that is specific to individual cores, as described below. (CIU_INT*n*_SUM0/4[MBOX] is CIU_PCI_INTA[INT] for the PCI destination.) CIU_INT*n*_SUM0[WDOG_SUM] is a convenient summary of the CIU_INT*n*_SUM1/CIU_INT*n*_EN4_1 result, so a single 64-bit access to CIU_INT*n*_SUM0/4 summarizes the available interrupts for the destination.

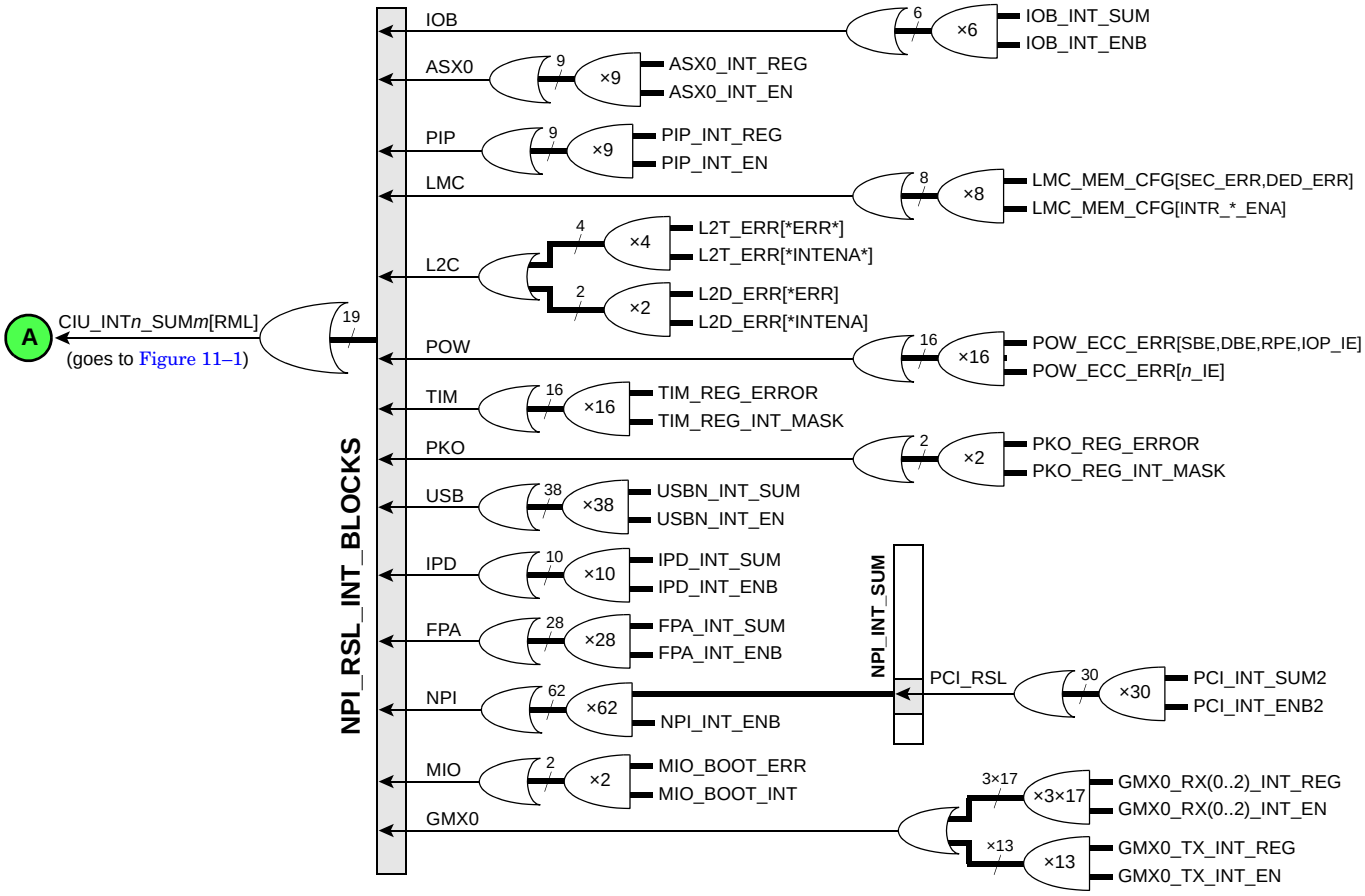


Figure 11–3 Input from NPI_RSL_INT_BLOCKS

Note that Figure 11–1 shows a simple depiction of the interrupt distribution logic. In the real implementation, there are delays to update interrupt destinations. The delay from when an interrupt source is updated to when the interrupt destination is updated may be as large as 60 cycles.

Except for CIU_INT*n*_SUM0/4[TIMER,IPD_DRP,GMX_DRP], CIU_INT*n*_SUM0/1/4 does not actually store the status bits for the interrupt, only summarizes them. Note that there is only one copy of the storage for CIU_INT*n*_SUM0/4[TIMER,IPD_DRP,GMX_DRP]. A write of 1 to any bit in these fields in any CIU_INT*n*_SUM0/4 clears the bit in all summaries.

Note that the majority of the interrupts enter via CIU_INT*n*_SUM0/4[RML] and NPI_RSL_INT_BLOCKS. There are many error checks distributed throughout CN50XX, and most of these errors set status bits that can cause interrupts via CIU_INT*n*_SUM0/4[RML] assertion. Figure 11–3 shows this.

CIU_INT*n*_SUM0/4[WDOG,TIMER,MBOX] are discussed in more detail in the following sections of this chapter. The remainder of the interrupts are discussed briefly here:

- CIU_INT*n*_SUM0/4[IPD_DRP,GMX_DRP<0>] are set whenever PIP/IPD or GMX0 drop a packet. As these are all the possible sources of packet drops, software can always be notified quickly of any packet drop.

- CIU_INT*n*_SUM0/4[TWSI] can assert on TWSI transmit arrival or completion. Refer to Chapter 17.
- CIU_INT*n*_SUM0/4[PCI_MSI] provides quick access to NPI_MSI_RCV, which can be used for PCI message-signalled interrupts. Refer to Section 9.8.
- CIU_INT*n*_SUM0/4[PCI_INT] provides interrupts via the external PCI interrupt input pins PCI_INTA_L/PCI_INTB_L/PCI_INTC_L/PCI_INTD_L. Note that PCI_INTA_L is an output, so probably should not be used here when CN50XX is not in PCI host mode (i.e. when PCI_HOST_MODE=0).
- CIU_INT*n*_SUM0/4[UART] are set when the corresponding UART indicates an interrupt. Refer to Chapter 16.
- CIU_INT*n*_SUM0/4[GPIO] provides either level-sensitive or edge-triggered interrupts for GPIO_RX_DAT[DAT]. GPIO_RX_DAT[DAT] is a 16-bit vector driven created from the GPIOs. Refer to Chapter 15.
- CIU_INT*n*_SUM0/4[WORKQ] provides per-group interrupts for POW. Refer to Section 5.6.
- CIU_PCI_INTA provides a direct mechanism for cores to send interrupts to the PCI bus.

11.2 Per-Core Mailbox Registers

Each core has a corresponding 32-bit mailbox register whose bits can be set/cleared. This CIU_MBOX_SET0/1/CIU_MBOX_CLR0/1 register is useful for inter-core direct interrupts, and a remote host can also direct an interrupt to a specific core with it.

Both CIU_MBOX_SET0/1[SET] and CIU_MBOX_CLR0/1[CLR] return the same copy of the mailbox bits for a core when they are read. Writes of 1s to CIU_MBOX_SET0/1[SET] cause corresponding mailbox bits to be set. Writes of 1s to CIU_MBOX_CLR0/1[CLR] cause corresponding mailbox bits to be cleared. With this, individual bits can be set and cleared simultaneously by multiple sources.

CIU_INT*n*_SUM0/4[MBOX] for a core summarizes the 32-bit mailbox register for the core. The core corresponding to summary *n* is n/2. CIU_INT*n*_SUM0/4[MBOX<1>] is set when any of <31:16> are set in the mailbox for the core. CIU_INT*n*_SUM0/4[MBOX<0>] is set when any of <15:0> are set in the mailbox for the core.

11.3 Per-Core Watchdog Timers

There are two watchdog timers in CN50XX, one for each of the two cores. The watchdogs can be accessed by any core or external PCI device, though a watchdog may generally only be used by its associated core. The CIU_WDOG0/1 registers configures each watchdog.

When not OFF, the watchdog timers count the time from the last “core poke” for the watchdog. A core poke indicates that the core is alive, and the watchdog should be reset. If a core poke does not arrive soon enough, a timeout occurs and the watchdog checks for another timeout. CIU_WDOG0/1[STATE] indicates the number of timeouts since the last core poke, and CIU_WDOG0/1[CNT]×256 is the cycles left before the next timeout. CIU_WDOG0/1[CNT] normally decrements once every 256 cycles when the watchdog is enabled. CIU_WDOG0/1[LEN]×65536 is the timeout in cycles. A core poke for a watchdog is a write of any value to the corresponding CIU_PP_POKE0/1 CSR.

Each of the two watchdog timers is in one of four modes, selected by CIU_WDOG0/1[MODE]:

- OFF (MODE = 0x0) – No interrupt/non-maskable interrupt/soft-reset is ever sourced by the watchdog.
- INTERRUPT ONLY (MODE = 0x1) – An interrupt asserts whenever there has been a timeout (i.e. whenever CIU_WDOG0/1[STATE] ≠ 0).
- INTERRUPT + NMI MODE (MODE = 0x2) – An interrupt asserts whenever there has been a timeout (i.e. whenever CIU_WDOG0/1[STATE] ≠ 0). An NMI pulse is generated for the associated core whenever a second timeout occurs (i.e. whenever CIU_WDOG0/1[STATE] is set to 2).
- INTERRUPT + NMI + SOFT-RESET (MODE = 0x3) – An interrupt asserts whenever there has been a timeout (i.e. whenever CIU_WDOG0/1[STATE] ≠ 0). An NMI pulse is generated for the associated core whenever a second timeout occurs (i.e. whenever CIU_WDOG0/1[STATE] is set to 2). A chip-wide soft-reset pulse is generated whenever a third timeout occurs (i.e. whenever CIU_WDOG0/1[STATE] would otherwise be set to 3).

The watchdogs are integrated with CN50XX’s debug features in the following way:

- When CIU_WDOG0/1[DSTOP] is set, the watchdog stops counting (i.e. CIU_WDOG0/1[CNT] does not decrement) whenever the corresponding core is in debug mode (i.e. whenever CIU_PP_DBG[PPDBG0/1] is set).
- When CIU_WDOG0/1[GSTOPEN] is set, the watchdog stops counting (i.e. CIU_WDOG0/1[CNT] does not decrement) whenever CN50XX is in global-stop mode (i.e. whenever CIU_GSTOP[GSTOP] is set).

Refer to Section 4.12 for more information regarding CN50XX debug.

There are three topics that deserve special mention with respect to the use of the CN50XX watchdog timers:

- delayed requests for new work
- long tag switch waits
- delayed Fetch&Add requests

These topics are discussed in the following sections.

Delayed Requests for New Work

The POW hardware can optionally delay a request for new work. When software selects this option, if work is not immediately available in one of the groups that the core is in, the hardware will hold onto the request until work comes available or the request times out. Without timeouts, a core could hang forever waiting for work, causing a watchdog expiration. With delayed work timeouts that are small relative to the watchdog expirations, as would be desired for this and other reasons, the software delayed work timeout handler can issue core-poke requests. The software may need to occasionally issue core-poke requests on timeouts when continuously waiting for work.

Refer to Chapter 5 and the POW_NW_TIM for more details.

Long Tag Switch Waits

Depending on software configuration, it may be possible for the delay of a tag switch wait to cause a watchdog expiration. In these situations, it may be necessary for the software handler that is polling for the tag-switch completion to occasionally issue core pokes. (In some cases, the preferred behavior may be to time out in this case, though.)

Refer to Chapter 5 for more information regarding tag switches.

Delayed Fetch&Add Requests

The hardware can optionally delay a Fetch&Add request. When software selects this option, the hardware either delays the operation until the prior tag switch completes, or times out the request. This is very similar to the delayed new work requests mentioned above. The solution could be the same also. But the solution may also be different since the software may not continuously resubmit the requests; it may instead fall into a tag switch wait.

Refer to Chapter 3 and the IOB_FAU_TIMEOUT for more details regarding FAU requests.

11.4 Four General Timers

CIU contains four general timers configured via CIU_TIM(0..3). Each timer can either be a one-shot timer or a periodic timer.

The three uses of the timers are:

- **Disabled**
  - Set CIU_TIM*n*[LEN] to zero.
- **One-shot**
  - Set CIU_TIM*n*[ONE_SHOT] to one.
  - Set CIU_TIM*n*[LEN] to the number of cycles.

  CIU_INT*_SUM0/4[TIMER*n*] is set when the timer expires, after which the timer becomes idle.

- **Periodic**
  - Set CIU_TIM*n*[ONE_SHOT] to zero.
  - Set CIU_TIM*n*[LEN] to the number of cycles–1 between periodic timeouts.

  CIU_INT*_SUM0/4[TIMER*n*] is set every time the timer expires (i.e. after exactly CIU_TIM*n*[LEN]+1 cycles each time).

After timer *n* generates a timer expiration (periodic or one-shot), CIU_INT*_SUM0/4[TIMER*n*] should be cleared by software so that the next expiration from timer *n* can cause a new interrupt.

11.5 Core Availability and Reset

The number of cores available on CN50XX is the number of bits set in CIU_FUSE.

CIU_PP_RST indicates the cores currently in reset on CN50XX. (When a bit is set, the corresponding core in CN50XX is in reset.) Except for the EJTAG TAP control register processor reset bit (ECR[PrRst]) local to each core, CIU_PP_RST is the sole reset source for CN50XX cores. Refer to the MIPS specifications and Chapter 4.

Note that the reset value of CIU_PP_RST depends on the external pin PCI_BOOT.

- When the external pin PCI_BOOT is asserted, both cores are held in reset after any chip hard or soft reset (i.e. CIU_PP_RST resets to 0x3).
- When the external pin PCI_BOOT is deasserted, core 0 is not held in reset after any chip hard or soft reset, but the other core is (i.e. CIU_PP_RST resets to 0x2).

Cores are taken out of reset by writing a 0 to the corresponding bit in CIU_PP_RST. This can be done either by a core or by a remote PCI host.

Cores can be put into reset, but CN50XX's stability cannot be guaranteed if the core is in the middle of any CMB, or I/O transaction at the time of the reset assertion for the core.

11.6 Core Debug-Mode Observability

CIU_PP_DBG returns the current debug mode state (i.e. Debug[DM]) of the cores. Refer to the MIPS specifications, Chapter 4, and Section 4.12.

11.7 Core Debug-Interrupt Generation

Writes of 1s to CIU_DINT cause the corresponding cores to enter debug mode, if they are not already in debug mode. CIU_DINT is one of the sources for core debug interrupts. Also included are the EJTAG TAP control register break bit (ECR[EjtagBrk]) local to each core, and MCD interrupts from the cores. Refer to the MIPS specifications, Chapter 4 and Section 4.12 for further documentation.

11.8 Core Non-Maskable Interrupt Generation

Writes of 1s to CIU_NMI cause NMI pulses to be sent to the corresponding cores. Except for CIU watchdog timers, described in Section 11.3, CIU_NMI is the sole source of core nonmaskable interrupts. Refer to the MIPS specifications and Chapter 4 for more details regarding non-maskable interrupts to the cores.

11.9 Chip Soft-Reset Initiation

To initiate a chip-wide soft reset, core or remote host software should do the following:

1. Write a 1 to CIU_SOFT_BIST[SOFT_BIST] to enable BIST during the soft reset
2. Read CIU_SOFT_RST, then write a 1 to CIU_SOFT_RST[SOFT_RST] to initiate the soft reset.

The effects of a soft reset are very similar to a chip reset initiated by the external reset pin. CHIP_RESET_L is the external reset pin when PCI_HOST_MODE=1, while PCI_RESET_L is the external reset pin when PCI_HOST_MODE=0.

One difference is that the Status[SR] bit in the cores are set so that boot software can tell that the reset was initiated from an internally-generated soft reset rather than an externally-generated reset.

WHEN PCI_HOST_MODE=1

All CN50XX CSRs and external interfaces are reset.

WHEN PCI_HOST_MODE=0

Except for the PCICONFIG, PCI, and PCI_NCB registers, all the CN50XX CSRs are reset. All interfaces other than the PCI interface are reset.

A remote host should not access CN50XX or any other CN50XX resources following a soft reset until a read of PCI_CTL_STATUS_2[ERST_N] returns a 1.

11.10 CIU Registers

The CIU registers are listed in Table 11–1.

Table 11–1 CIU Registers

Registers	Address	CSR Type[1]	Detailed Description
CIU_INT0_SUM0 CIU_INT1_SUM0 CIU_INT2_SUM0 CIU_INT3_SUM0 CIU_INT32_SUM0	0x0001070000000000 0x0001070000000008 0x0001070000000010 0x0001070000000018 0x0001070000000100	NCB	See page 470
CIU_INT_SUM1	0x0001070000000108	NCB	See page 470
CIU_INT0_EN0 CIU_INT1_EN0 CIU_INT2_EN0 CIU_INT3_EN0 CIU_INT32_EN0	0x0001070000000200 0x0001070000000210 0x0001070000000220 0x0001070000000230 0x0001070000000400	NCB	See page 471
CIU_INT0_EN1 CIU_INT1_EN1 CIU_INT2_EN1 CIU_INT3_EN1 CIU_INT32_EN1	0x0001070000000208 0x0001070000000218 0x0001070000000228 0x0001070000000238 0x0001070000000408	NCB	See page 471
CIU_TIM0 ... CIU_TIM3	0x0001070000000480 ... 0x0001070000000498	NCB	See page 474
CIU_WDOG0 CIU_WDOG1	0x0001070000000500 0x0001070000000508	NCB	See page 474
CIU_PP_POKE0 CIU_PP_POKE1	0x0001070000000580 0x0001070000000588	NCB	See page 474
CIU_MBOX_SET0 CIU_MBOX_SET1	0x0001070000000600 0x0001070000000608	NCB	See page 475
CIU_MBOX_CLR0 CIU_MBOX_CLR1	0x0001070000000680 0x0001070000000688	NCB	See page 475
CIU_PP_RST	0x0001070000000700	NCB	See page 475
CIU_PP_DBG	0x0001070000000708	NCB	See page 475
CIU_GSTOP	0x0001070000000710	NCB	See page 475
CIU_NMI	0x0001070000000718	NCB	See page 476
CIU_DINT	0x0001070000000720	NCB	See page 476
CIU_FUSE	0x0001070000000728	NCB	See page 476
CIU_BIST	0x0001070000000730	NCB	See page 476
CIU_SOFT_BIST	0x0001070000000738	NCB	See page 476
CIU_SOFT_RST	0x0001070000000740	NCB	See page 477
CIU_SOFT_PRST	0x0001070000000748	NCB	See page 477
CIU_PCI_INTA	0x0001070000000750	NCB	See page 477
CIU_INT0_SUM4 CIU_INT1_SUM4	0x0001070000000C00 0x0001070000000C08	NCB	See page 472
CIU_INT0_EN4_0 CIU_INT1_EN4_0	0x0001070000000C80 0x0001070000000C90	NCB	See page 473
CIU_INT0_EN4_1 CIU_INT1_EN4_1	0x0001070000000C88 0x0001070000000C98	NCB	See page 473

1. NCB-type registers are accessed directly across the I/O bus, and RSL-type registers are accessed indirectly across the I/O bus.

CIU Interrupt Summary 0 Registers
CIU_INT(0..3,32)_SUM0

This register indicates where an interrupt has occurred. The five registers are apportioned in the following manner:

CIU_INT0_SUM0 is used for core0/IP2
CIU_INT1_SUM0 is used for core0/IP3
CIU_INT2_SUM0 is used for core1/IP2
CIU_INT3_SUM0 is used for core1/IP3
CIU_INT32_SUM0 is used for PCI/INTA

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:59>	—	RAZ	—	—	Reserved.
<58>	MPI	R/W1C	0	0	MPI/SPI Interrupt
<57>	PCM	RO	0	0	PCM/TDM Interrupt
<56>	USB	RO	0	0	USB Interrupt
<55:52>	TIMER	R/W1C	0	0	General timer interrupts
<51>	—	RAZ	—	—	Reserved.
<50>	IPD_DRP	R/W1C	0	0	IPD QOS packet drop interrupt
<49>	—	RAZ	—	—	Reserved.
<48>	GMX_DRP	R/W1C	0	0	GMX packet drop interrupt
<47>	—	RAZ	—	—	Reserved.
<46>	RML	RO	0	0	RML interrupt
<45>	TWSI	RO	0	0	TWSI interrupt
<44>	WDOG_SUM	RO	0	0	Watchdog summary ● Cores use CIU_INT*n*_SUM0 where *n* = 0–3. – Even INTx registers report WDOG to IP2. – Odd INTx registers report WDOG to IP3 ● PCI uses the CIU_INT32_SUM0.
<43:40>	PCI_MSI	RO	0x0	0x0	PCI MSI <43> is the OR of <63:48> <42> is the OR of <47:32> <41> is the OR of <31:16> <40> is the OR of <15:0>
<39:36>	PCI_INT	RO	0x0	0x0	PCI INTA/B/C/D
<35:34>	UART	RO	0x0	0x0	Two UART interrupts
<33:32>	MBOX	RO	0x0	0x0	Two mailbox interrupts for entries 0-3. <33> is the OR of <31:16> <32> is the OR of <15:0> Two PCI internal interrupts for entry 32 CIU_PCI_INTA
<31:16>	GPIO	RO	0x0	0x0	Sixteen GPIO interrupts
<15:0>	WORKQ	RO	0x0	0x0	Sixteen work-queue interrupts. One bit/group. A copy of the R/W1C bit in the POW.

CIU Interrupt Summary 1 Register
CIU_INT_SUM1

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1:0>	WDOG	RO	0x0	0x0	Two watchdog interrupts.

CIU Interrupt Enable 0 Registers
CIU_INT(0..3,32)_EN0

This register provide enable bits for interrupt bits in CIU_INT*n*_SUM0. The five registers are apportioned in the following manner:

CIU_INT0_EN0 is used for core0/IP2
CIU_INT1_EN0 is used for core0/IP3
CIU_INT2_EN0 is used for core1/IP2
CIU_INT3_EN0 is used for core1/IP3
CIU_INT32_EN0 is used for PCI/INTA

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:59>	—	RAZ	—	—	Reserved.
<58>	MPI	R/W	0	0	MPI/SPI Interrupt
<57>	PCM	R/W	0	0	PCM/TDM Interrupt
<56>	USB	R/W	0	0	USB Interrupt
<55:52>	TIMER	R/W	0x0	0x0	General timer interrupts
<51>	—	RAZ	—	—	Reserved.
<50>	IPD_DRP	R/W	0	0	IPD QOS packet drop
<49>	—	RAZ	—	—	Reserved.
<48>	GMX_DRP	R/W	0	0	GMX packet drop
<47>	—	RAZ	—	—	Reserved.
<46>	RML	R/W	0	0	RML Interrupt
<45>	TWSI	R/W	0	0	TWSI Interrupt
<44>	—	R/W	0	0	Reserved.
<43:40>	PCI_MSI	R/W	0x0	0x0	PCI MSI.
<39:36>	PCI_INT	R/W	0x0	0x0	PCI INTA/B/C/D
<35:34>	UART	R/W	0x0	0x0	Two UART interrupts
<33:32>	MBOX	R/W	0x0	0x0	Two mailbox/PCI interrupts
<31:16>	GPIO	R/W	0x0	0x0	Sixteen GPIO interrupts
<15:0>	WORKQ	R/W	0x0	0x0	Sixteen work-queue interrupts. one bit per group.

CIU Interrupt Enable 1 Registers
CIU_INT(0..3,32)_EN1

This register provide interrupt-enable bits for watchdog vectors in CIU_INT*n*_SUM1. The five registers are apportioned in the following manner:

CIU_INT0_EN1 is used for core0/IP2
CIU_INT1_EN1 is used for core0/IP3
CIU_INT2_EN1 is used for core1/IP2
CIU_INT3_EN1 is used for core1/IP3
CIU_INT32_EN1 is used for PCI/INTA

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1:0>	WDOG	RO	0x0	0x0	Watchdog summary interrupt-enable vectors.

CIU Interrupt Summary IP4 Registers
CIU_INT0/1_SUM4

Indicates where an interrupt has occurred. The registers are apportioned in the following manner:

CIU_INT0_SUM4 is used for core0/IP4

CIU_INT1_SUM4 is used for core1/IP4

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:59>	—	RAZ	—	—	Reserved.
<58>	MPI	R/W1C	0	0	MPI/SPI interface interrupt.
<57>	PCM	RO	0	0	PCM/TDM interface interrupt.
<56>	USB	RO	—	—	USB interface interrupt.
<55:52>	TIMER	R/W1C	0	0	General timer interrupts.
<51>	—	RAZ	—	—	Reserved.
<50>	IPD_DRP	R/W1C	0	0	IPD QOS packet drop interrupt.
<49>	—	RAZ	—	—	Reserved.
<48>	GMX_DRP	R/W1C	0	0	GMX packet drop interrupt.
<47>	—	RAZ	—	—	Reserved.
<46>	RML	RO	0	0	RML interrupt.
<45>	TWSI	RO	0	0	TWSI interrupt.
<44>	WDOG_SUM	RO	0	0	Watchdog summary. These registers report WDOG to IP4.
<43:40>	PCI_MSI	RO	0x0	0x0	PCI MSI. <43> is the OR of <63:48> <42> is the OR of <47:32> <41> is the OR of <31:16> <40> is the OR of <15:0>
<39:36>	PCI_INT	RO	0x0	0x0	PCI INTA/B/C/D.
<35:34>	UART	RO	0x0	0x0	Two UART interrupts.
<33:32>	MBOX	RO	0x0	0x0	Two mailbox interrupts for entries 0-31. <33> is the OR of <31:16> <32> is the OR of <15:0> Two PCI internal interrupts for entry 32 CIU_PCI_INTA.
<31:16>	GPIO	RO	0x0	0x0	Sixteen GPIO interrupts.
<15:0>	WORKQ	RO	0x0	0x0	Sixteen work-queue interrupts. One bit/group. A copy of the R/W1C bit in the POW.

CIU Interrupt Enable IP4 Registers 0
CIU_INT0/1_EN4_0

Provide enable bits for interrupts. The registers are apportioned in the following manner:

CIU_INT0_EN4_0 is used for core0/IP4

CIU_INT1_EN4_0 is used for core1/IP4

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:59>	—	RAZ	—	—	Reserved.
<58>	MPI	R/W	0	0	MPI/SPI interface interrupt.
<57>	PCM	R/W	0	0	PCM/TDM interface interrupt.
<56>	USB	R/W	0	0	USB interface interrupt.
<55:52>	TIMER	R/W	0x0	0x0	General timer interrupts.
<51>	—	R/W	0	0	Reserved.
<50>	IPD_DRP	R/W	0	0	IPD QOS packet drop.
<49>	—	R/W	0	0	Reserved.
<48>	GMX_DRP	R/W	0	0	GMX packet drop.
<47>	—	RAZ	0	0	Reserved.
<46>	RML	R/W	0	0	RML Interrupt.
<45>	TWSI	R/W	0	0	TWSI Interrupt.
<44>	—	RAZ	—	—	Reserved.
<43:40>	PCI_MSI	R/W	0x0	0x0	PCI MSI.
<39:36>	PCI_INT	R/W	0x0	0x0	PCI INTA/B/C/D.
<35:34>	UART	R/W	0x0	0x0	Two UART interrupts.
<33:32>	MBOX	R/W	0x0	0x0	Two mailbox/PCI interrupts.
<31:16>	GPIO	R/W	0x0	0x0	Sixteen GPIO interrupts.
<15:0>	WORKQ	R/W	0x0	0x0	Sixteen work-queue interrupts.

CIU Interrupt Enable IP4 Registers 1
CIU_INT0/1_EN4_1

Provide enable bits for watchdog vectors. The registers are apportioned in the following manner:

CIU_INT0_EN4_1 is used for core0/IP4

CIU_INT1_EN4_1 is used for core1/IP4

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1:0>	WDOG	R/W	0	0	Watchdog summary interrupt-enable vector

CIU General Timer Registers

CIU_TIM(0..3)

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:37>	—	RAZ	—	—	Reserved.
<36>	ONE_SHOT	R/W	0	0	One-shot mode when LEN ≠ 0x0: 1 = timer is in **one-shot** mode, 0 = timer is in **periodic** mode.
<35:0>	LEN	R/W	0x0	0x0	Time-out length in core-clock cycles – 1. The timer disabled when LEN = 0x0.

CIU Watchdog Registers

CIU_WDOG0/1

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:46>	—	RAZ	—	—	Reserved.
<45>	GSTOPEN	R/W	0	0	Global-stop enable.
<44>	DSTOP	R/W	0	0	Debug-stop enable.
<43:20>	CNT	RO	0	0	Number of 256-cycle intervals until next watchdog expiration. Cleared on write to associated CIU_PP_POKE*n* register.
<19:4>	LEN	R/W	0	0	Watchdog time-expiration length. This field represents the most-significant bits of a 24-bit decrementer that decrements every 256 cycles. Must be set greater than 0x0.
<3:2>	STATE	RO	0	0	Watchdog state. The number of watchdog time expirations since last core poke. Cleared on write to associated CIU_PP_POKE*n* register.
<1:0>	MODE	R/W	0	0	Watchdog mode: 0 = Off 1 = Interrupt only 2 = Interrupt + NMI 3 = Interrupt + NMI + soft reset

CIU Cores Poke Registers

CIU_PP_POKE0/1

Write operations to these registers cause the following:

- clears any pending interrupts generated by the associated watchdog
- resets the CIU_WDOG*n*[STATE] field
- sets CIU_WDOG[CNT] to be (CIU_WDOG[LEN] << 8)

Read operations to these registers return the associated CIU_WDOG register. See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	—	RAZ	—	—	Reserved

CIU Mailbox Set Registers
CIU_MBOX_SET0/1

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	SET	R/W1	0x0	0x0	Writing a 1 to a bit sets the corresponding bit in mailbox storage. Reading this register returns the contents of mailbox storage.

CIU Mailbox Clear Registers
CIU_MBOX_CLR0/1

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	CLR	R/W1C	0x0	0x0	Writing a 1 to a bit clears the corresponding bit in mailbox storage. Reading this register returns the contents of mailbox storage.

CIU Cores Reset Register
CIU_PP_RST

This register holds the reset control for each core. A value of 1 holds the corresponding core in reset, 0 releases it from reset. The reset value is 0x3 when PCI_BOOT is asserted, or 0x2 when PCI_BOOT is clear.

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1>	RST	R/W	1	0	Core reset for core 1.
<0>	RST0	R/W	—	0	Core reset for core 0.

CIU Cores Debug Register
CIU_PP_DBG

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1:0>	PPDBG	RO	0x0	0x0	Debug[DM] value for each core whether the cores are in debug mode or not.

CIU Global-Stop Register
CIU_GSTOP

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	GSTOP	R/W	0	0	Set global-stop mode.

CIU NonMaskable Interrupt Register
CIU_NMI

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1:0>	NMI	WO	0x0	0x0	Writing a 1 to either bit sends an NMI pulse to the corresponding core vector.

CIU Debug Interrupt Register
CIU_DINT

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1:0>	DINT	WO	0x0	0x0	Each bit set sends a DINT pulse to the corresponding core.

CIU Fuse Register
CIU_FUSE

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1:0>	FUSE	RO	—	—	Each bit set indicates a physical core is present.

CIU BIST Register
CIU_BIST

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	—	—	Reserved.
<3:0>	BIST	RO	0x0	0x0	BIST results. Hardware sets a bit for each memory that fails BIST.

CIU Soft BIST Register
CIU_SOFT_BIST

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	SOFT_BIST	R/W	0	0	BIST on soft reset enable. When set, BIST is run when soft reset is asserted.

CIU Cores Soft Reset Register
CIU_SOFT_RST

This register allows the software to reset the CN50XX cores. When resetting from a remote PCI device, always read this register first (and wait for the result), then set SOFT_RST. See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	SOFT_RST	WO	0	0	When set, resets the CN50XX core.

CIU PCI Soft Reset Register
CIU_SOFT_PRST

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved.
<2>	HOST64	R/W	0	0	PCI-X host-mode device capability. Must be 0.
<1>	NPI	R/W	0	0	When set and SOFT_PRST = 1, resets the NPI and PNI logic
<0>	SOFT_PRST	R/W	1	0	When set and CN50XX is configured as a host, resets the PCI bus (i.e. asserts PCI_RST_L). Refer to Section 9.11.

CIU PCI Interrupt Register
CIU_PCI_INTA

See Table 11–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1:0>	INT	R/W	0x0	0x0	PCI interrupt. These bits are observed in CIU_INT32_SUM0<33:32>

Chapter 12

Boot Bus

This section contains the following subjects:

Overview

The CN50XX boot-bus hardware attaches to nonvolatile devices, like FLASHes, compact flashes, and ROMs, and can service the initial boot address on CN50XX. The bus protocol is flexible, so it can interface to many other types of devices as well. The big-endian boot bus contains:

- eight chip-selects (four are shared with GPIO signals: GPI_GPIO<11:8>)
- 32 address/data lines
- programmable 8/16-bit data width
  - With the 8-bit option, BOOT_AD<31:24> is the data bus
  - With the 16-bit option, BOOT_AD<31:24> is the most-significant byte and BOOT_AD<23:16> is the least-significant byte of the data bus

Figure 12–1 shows the boot-bus hardware.

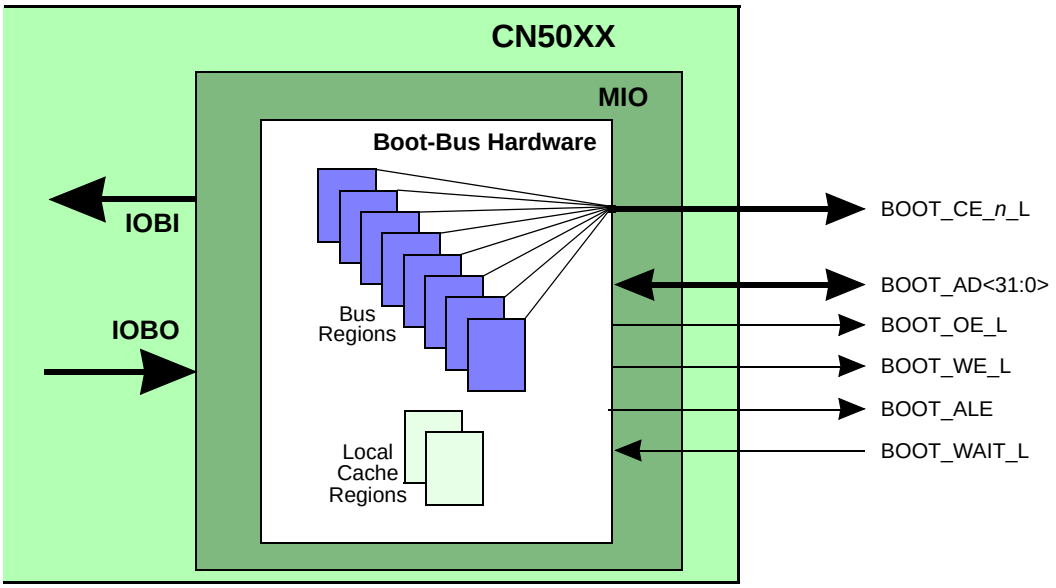


Figure 12–1 Boot-Bus Hardware

The boot-bus hardware can configure up to eight regions that correspond to the eight chip-selects. Each region/chip select has different bus configuration and timing parameters. When CN50XX self-boots (i.e. when the external pin PCI_BOOT = 0), the boot bus must provide the initial instructions to boot the chip on chip-select 0.

The boot-bus hardware also contains two local, 128-byte cache regions. These cache regions must be used to service core-boot and debug-exception vector references when there is no device attached to the boot bus that will service them. The local cache regions can also accelerate reads to commonly accessed areas of the boot bus.

A remote PCI host can also use the boot bus. The boot-bus hardware services all PCI expansion-ROM references, as well as all references to the boot-bus physical addresses via the PCI windowed addressing mechanism. (The remote PCI host accesses the PCI_WIN_WR_ADDR, PCI_WIN_RD_ADDR, PCI_WIN_WR_DATA, PCI_WIN_WR_MASK, and PCI_WIN_RD_DATA CSRs to use the PCI windowed-addressing mechanism.)

12.1 Boot-Bus Addresses

The boot bus consumes a 4GB portion of the CN50XX physical-address (PA) space range from 0x1 0000 0000 0000 to 0x1 0000 FFFF FFFF.

NOTE: CN50XX cores also convert all of their physical addresses in the range 0x0 0000 1000 0000 to 0x0 0000 1FFF FFFF to CN50XX boot-bus physical addresses in the range 0x1 0000 1000 0000 to 0x1 0000 1FFF FFFF.

The boot-bus physical addresses shown in Table 12–1 are particularly important, as they are exception vectors for the cores.

Table 12–1 Exception Vectors

Vector Address (PA)	Comment
0x1 0000 1FC0 0000	A core vectors to this physical address after all resets, soft resets, or nonmaskable interrupt (NMI) exceptions.
0x1 0000 1FC0 0480	If the EJTAG TAP register field ECR[ProbEn] is not set, a core vectors to this physical address after all debug exceptions (i.e. whenever it enters debug mode).

Other core exceptions can also vector to boot-bus addresses in certain situations. The other exception vectors can be avoided by configuration if desired (by clearing the core COP0 register bit Status[BEV]), but the vector addresses in Table 12–1 cannot be avoided by core configuration.

If a system does not have any devices attached to the boot bus, the two local cache regions in the boot-bus hardware must service the (instruction fetch) reads that immediately follow the exceptions listed in Table 12–1. In such a system, CN50XX cannot self-boot (i.e. the external PCI_BOOT pin must be set to 1). Each local cache region can hold up to 32 instructions at these vector addresses. If more instructions are needed to service these exceptions in these systems, the exception handler must jump to L2/DRAM within these first 32 instructions.

The following boot-bus address range is also of particular interest regarding PCI expansion ROM references.

Table 12–2 PCI Expansion-ROM Address Range

Address (PA) Range	Comment
0x1 0000 1FC1 0000 to 0x1 0000 1FC1 FFFF	This is the 64KB physical-address range used for PCI expansion ROM references. The PCI hardware converts expansion-ROM reads by a remote PCI host into CN50XX boot-bus accesses in this range.

12.2 Boot-Bus Address Matching and Regions

The boot-bus hardware only services boot-bus references by matching their PA with programmed regions. For read operations, the hardware first checks for a match with the local cache regions. If the read PA matches one of the two local cache regions, the hardware services the read with the data in the cache inside the unit. Otherwise (i.e. for any write operations or for a read operation that does not hit in a local cache region), the boot-bus hardware checks for a match of the PA with the bus regions. On a match, the hardware uses that region (and its corresponding chip select and timing parameters) to complete the reference.

Each of the two local cache regions are enabled when the corresponding MIO_BOOT_LOC_CFG*n*[EN] is set. Each local cache region is 128 bytes when enabled, and matches when PA<31:7> of the read PA equals MIO_BOOT_LOC_CFG*n*[BASE].

The storage for each local-cache region is held in the boot-bus logic, and must be filled by software before the local-cache region is enabled. Software can fill local-cache region 0 by performing the following steps:

1. Set MIO_BOOT_LOC_ADR to 0x0 (set to 0x80 for local-cache region 1).
2. Write MIO_BOOT_LOC_DAT 16 times with eight bytes each time.

Each of the eight bus regions are enabled when the corresponding MIO_BOOT_REG_CFG*n*[EN] is set. The size in bytes of each enabled bus region is:

$$(\text{MIO\_BOOT\_REG\_CFG}n\text{[SIZE]} + 1) \times 2^{16}$$

A boot-bus read or write operation matches an enabled bus region when both of the following conditions are met:

$$\text{PA<31:16>} \geq \text{MIO\_BOOT\_REG\_CFG}n\text{[BASE]}, \quad \text{AND}$$

$$\text{PA<31:16>} \leq \text{MIO\_BOOT\_REG\_CFG}n\text{[BASE]} + \text{MIO\_BOOT\_REG\_CFG}n\text{[SIZE]}$$

When a boot-bus reference matches an enabled bus region, the chip-select for the region asserts to complete the reference. The boot-bus address used to access the external device is:

$$\text{BOOT\_A<27:0>} = \text{PA<31:0>} - (\text{MIO\_BOOT\_REG\_CFG}n\text{[BASE]} \ll 16)$$

NOTE: Depending on the configuration, it is possible that some of the 28 internal address bits will not appear on BOOT_AD<27:0>.

Software must ensure the following conditions:

- Software must not allow more than one local-cache region to match any PA.
- Software must not allow more than one bus region to match any PA.

It is legal for a PA to match both a local cache region and a bus region, though. When this happens, a read takes the data from the local-cache region.

12.3 Boot-Bus Reset Configuration and Booting

The reset configuration for the bus regions must be used when CN50XX self-boots (i.e. when PCI_BOOT = 0), as the core-reset vector is used immediately out of reset in this case. After reset, bus region 0 is set up this way:

- it is the only valid region
- it is maximum size
- it maps CN50XX PA = 0x1 0000 1FC0 0000 to boot address = 0x0 on the device attached to chip-select 0.
- it uses an 8-bit data width, unless an external pullup resistor is connected to BOOT_AD[14], then it uses a 16-bit data width.
- it operates in standard, nonmultiplexed mode, unless an external pullup resistor is connected to BOOT_AD[15], then it operates in multiplexed mode.

This means that when CN50XX self-boots, the device attached to chip-select 0 must contain the boot code for the CN50XX starting at boot address = 0. When the CN50XX self-boots, core 0 executes this code immediately following chip-reset deassertion. The reset timing parameters for region 0 are set to conservative values to accommodate a wide range of devices. If desired, the timing parameters for region 0 can be changed for faster operation while booting.

The reset configuration for the bus regions typically must also be used when PCI expansion-ROM references occur in the system, since PCI expansion-ROM references from a remote host occur early in the PCI-bus boot sequence. The boot-bus regions will typically not have changed from their reset values at the time that the reads occur, so all PCI expansion-ROM reads map to the device attached to chip select 0.

- To fully support a PCI expansion ROM via CN50XX, a device typically must be attached to chip-select 0, and must contain the expansion-ROM data starting at boot address = 0x10000 in the device.
- If a device is not physically attached to boot-bus chip-select 0, any PCI expansion-ROM read operations return 0 (as do any other references to the region), and it appears that a PCI expansion ROM is not present to remote PCI host software.
- If a device is attached to chip-select 0 and PCI expansion-ROM read operations to CN50XX occur, the remote PCI-host configuration software still thinks that an expansion ROM is not present as long as read operations near physical address 0x1 0000 1FC1 0000 (i.e. boot address = ~0x10000 in the device) return 0s.

  When a system has a memory device attached to chip-select 0, but wants it to appear to a remote PCI host that the PCI expansion ROM is not present, the system must set the boot addresses = ~0x10000 to 0 on the device attached to chip-select 0.

12.4 Boot-Bus Region Timing

The mode and timing of each boot-bus region can be configured independently (using MIO_BOOT_REG_TIM*n*), so the characteristics of each device on the bus can be completely different. The different chip-selects insulate the different devices on the bus from each other. CN50XX is the default driver of the BOOT_DAT data bus.

Most individual devices only require a single set of modes and timing parameters. The boot bus has an OR mode, enabled by setting MIO_BOOT_REG_CFG*n*[OR], that makes CN50XX support two different modes/timings for a device, though the device need only connect to a single BOOT_CE_*n*_L wire. When MIO_BOOT_REG_CFG*n*[OR] is set, BOOT_CE_*n*_L asserts whenever either region *n* or region *n–1* matches.

- If region *n* matches, the mode and timing for region *n* are used.
- If region *n–1* matches, the mode and timing for region *n–1* are used.

In either case, BOOT_CE_*n*_L asserts. This can be useful for devices like the compact flash. The compact flash has both attribute and common memories, each with very different characteristics. MIO_BOOT_REG_CFG*n*[OR] is not needed if the different memories can be accessed at different times, however (with a MIO_BOOT_REG_TIM*n* reconfiguration between them).

Table 12–3 shows the configuration of the two mode bits that determine bus operation.

Table 12–3 Bus Operation

MIO_BOOT_REG_TIM*n*		
<63>[PAGEM]	<62>[WAITM]	Comment
0	0	Static-timed, page-mode disabled
0	1	Dynamic-timed (via BOOT_WAIT_L)
1	0	Static-timed, page-mode enabled
1	1	Illegal

Table 12–4 describes the time intervals used in the timing diagrams that follow.

Table 12–4 Boot-Bus Timing Parameters

Parameter	Field	Description
T_{ADR}	MIO_BOOT_REG_TIM*n*[ADR]	The time that the boot address is valid before BOOT_CE_*n*_L asserts in nonmultiplexed mode, or the time between the deassertion of BOOT_ALE and the assertion of BOOT_CE_*n*_L in multiplexed mode. Also affects the minimum idle time between bus references in nonmultiplexed mode.
T_{CE}	MIO_BOOT_REG_TIM*n*[CE]	The time that BOOT_CE_*n*_L is asserted before BOOT_OE_L (read operation) or BOOT_WE_L (write operation) asserts.
T_{OE}	{MIO_BOOT_REG_CFG*n*[OE_EXT], MIO_BOOT_REG_TIM*n*[OE]}	The time that BOOT_OE_L is asserted for a read operation.
T_{RH}	MIO_BOOT_REG_TIM*n*[RD_HLD]	The time that BOOT_CE_*n*_L remains asserted after BOOT_OE_L deasserts for a read operation.
T_{PAUSE}	MIO_BOOT_REG_TIM*n*[PAUSE]	The time that the boot address retains its value after BOOT_CE_*n*_L deasserts. Also affects the minimum idle time between bus references.
T_{WE}	{MIO_BOOT_REG_CFG*n*[WE_EXT], MIO_BOOT_REG_TIM*n*[WE]}	The time that BOOT_WE_L is asserted for a write operation.
T_{WH}	MIO_BOOT_REG_TIM*n*[WR_HLD]	The time that BOOT_CE_*n*_L remains asserted after BOOT_WE_L deasserts for a write operation.
T_{PAGE}	MIO_BOOT_REG_TIM*n*[PAGE]	The time that the boot address must be valid before resampling BOOT_DAT on a page-mode read operation.
T_{WAIT}	MIO_BOOT_REG_TIM*n*[WAIT]	The time that BOOT_WAIT_L must be deasserted before BOOT_OE_L (read operation) or BOOT_WE_L (write operation) deasserts.
T_{ALE}	MIO_BOOT_REG_TIM*n*[ALE]	The time that BOOT_ALE is asserted in multiplexed mode.

The timing configuration parameters, with the exception of MIO_BOOT_REG_TIM*n*[WAIT], specify the time interval in internal core-clock cycles:

time_interval = **(TM × config_parameter + 1)** core-clock cycles

where TM = MIO_BOOT_REG_CFG*n*[TIM_MULT]

The wait time after the deassertion of BOOT_WAIT_L is specified in internal core-clock cycles:

time_interval = **MIO_BOOT_REG_TIM*n*[WAIT]** core-clock cycles

Table 12–5 describes the fixed time intervals used in the timing diagrams.

Table 12–5 Boot-Bus Fixed-Timing Parameters

Parameter	Value	Description
T_{DS}	6 ns – (MIO_BOOT_REG_CFG*n*[RD_DLY] core-clock cycles)	Data setup time with respect to the rising edge of BOOT_OE_L or the transition of BOOT_AD<2:0> (in page mode).
T_{DH}	(MIO_BOOT_REG_CFG*n*[RD_DLY] core-clock cycles) –1 ns	Data hold time with respect to the rising edge of BOOT_OE_L or the transition of BOOT_AD<2:0> (in page mode).
T_{WS}	6 ns	Wait setup time with respect to the expiration of T_{OE} or T_{WE}.

In the timing sequences in the following sections, the examples can have the following properties:

- multiplexed (ALE) or nonmultiplexed (not ALE)
- eight-bit data width (8W) or 16-bit data width (16W)

12.4.1 Static-Timed Read Sequences

Static-timed read operations can be performed in the following modes:

- eight-bit, nonmultiplexed ($\overline{\text{ALE}}$, 8W)
- eight-bit, multiplexed (ALE, 8W)
- 16-bit, nonmultiplexed ($\overline{\text{ALE}}$, 16W)
- 16-bit, multiplexed (ALE, 16W)

Static-Timed Read: Not ALE, 8W

Figure 12–2 shows a boot-bus single-byte read transaction in eight-bit, nonmultiplexed mode. All reads to the device are single-byte when MIO_BOOT_REG_TIM*n*[PAGEM] = 0 and MIO_BOOT_REG_CFG*n*[WIDTH] = 0.

The boot address (BOOT_AD<23:0>) first becomes valid, then BOOT_CE_*n*_L asserts, then BOOT_OE_L asserts.

CN50XX stops driving boot data (BOOT_AD<31:24>) to 0s at the same time that BOOT_OE_L asserts. CN50XX samples the boot data as it deasserts BOOT_OE_L. After that, BOOT_CE_*n*_L deasserts, then the boot address returns to 0s. CN50XX drives the boot data to 0s after it deasserts BOOT_CE_*n*_L.

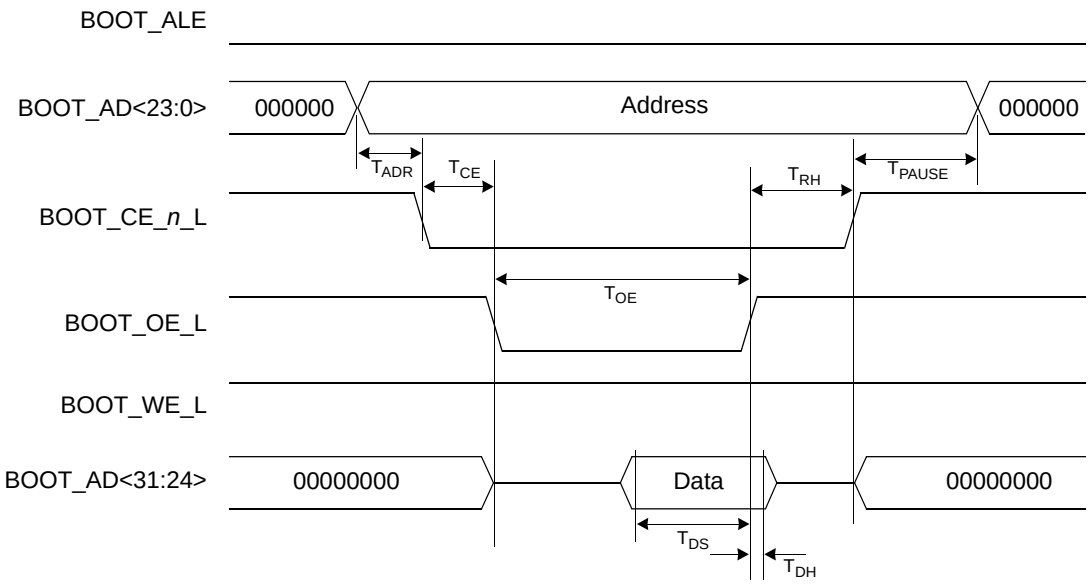


Figure 12–2 Static-Timed Read Sequence (not ALE, 8W)

Static-Timed Read: ALE, 8W

Figure 12–3 shows a boot-bus single-byte read transaction in eight-bit, multiplexed mode. All reads to the device are single-byte when MIO_BOOT_REG_TIM*n*[PAGEM] = 0 and MIO_BOOT_REG_CFG*n*[WIDTH] = 0.

BOOT_ALE asserts and the boot address (BOOT_AD<31:0>) becomes valid first, then BOOT_ALE deasserts, then BOOT_CE_*n*_L asserts, then BOOT_OE_L asserts.

CN50XX stops driving the upper boot address bits (BOOT_AD<31:24>) at the same time that BOOT_OE_L asserts. CN50XX samples the boot data as it deasserts BOOT_OE_L. After that, BOOT_CE_*n*_L deasserts, then the boot address returns to 0s. CN50XX drives the boot data after it deasserts BOOT_CE_*n*_L.

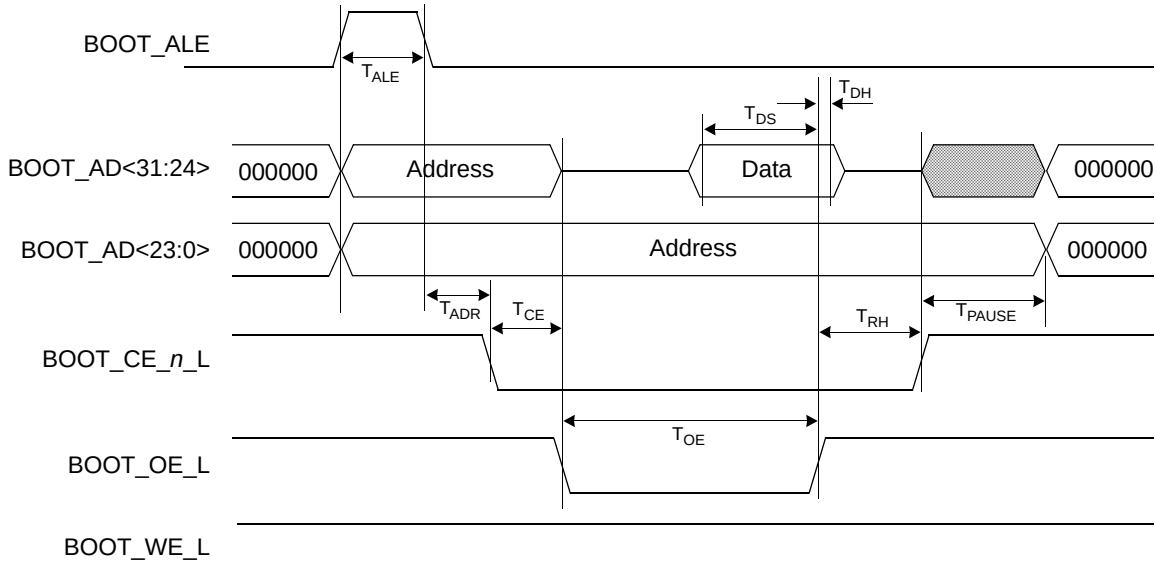


Figure 12–3 Static-Timed Read Sequence (ALE, 8W)

Static-Timed Read: Not ALE, 16W

Figure 12–4 shows a boot-bus two-byte read transaction in 16-bit, nonmultiplexed mode. All reads to the device are two-byte when MIO_BOOT_REG_TIM*n*[PAGEM] = 0 and MIO_BOOT_REG_CFG*n*[WIDTH] = 1.

The boot address (BOOT_AD<15:0>) first becomes valid, then BOOT_CE_*n*_L asserts, then BOOT_OE_L asserts.

CN50XX stops driving boot data (BOOT_AD<31:16>) to 0s at the same time that BOOT_OE_L asserts. CN50XX samples the boot data as it deasserts BOOT_OE_L. After that, BOOT_CE_*n*_L deasserts, then the boot address returns to 0s. CN50XX drives the boot data to 0s after it deasserts BOOT_CE_*n*_L.

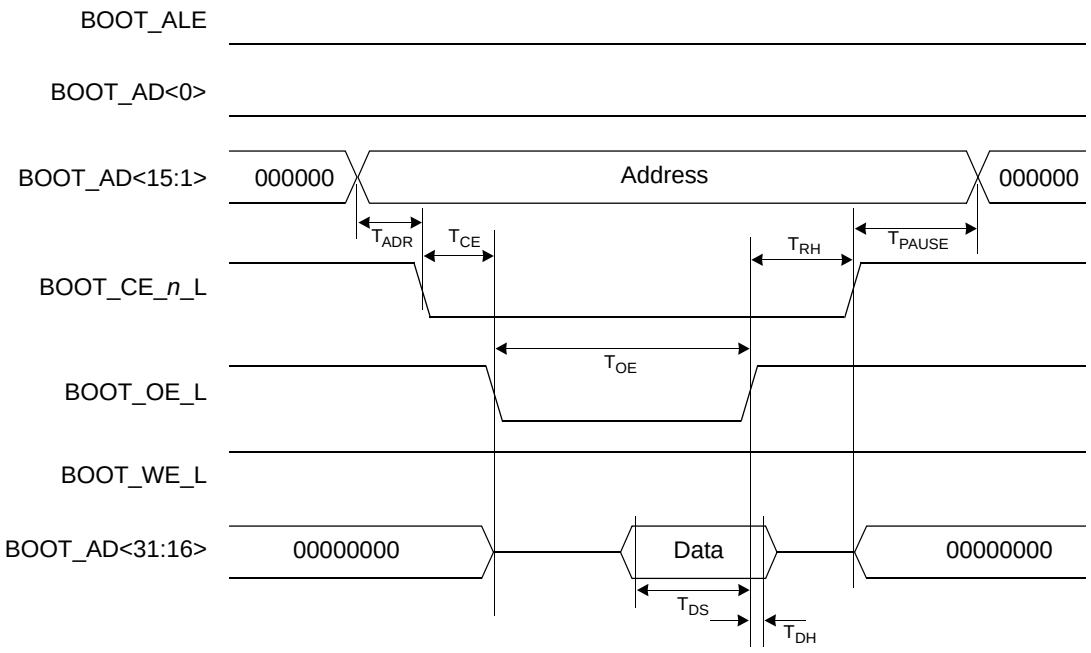


Figure 12–4 Static-Timed Read Sequence (not ALE, 16W)

Static-Timed Read: ALE, 16W

Figure 12–5 shows a boot-bus two-byte read transaction in 16-bit, multiplexed mode. All reads to the device are two-byte when MIO_BOOT_REG_TIM*n*[PAGEM] = 0 and MIO_BOOT_REG_CFG*n*[WIDTH] = 1.

BOOT_ALE asserts and the boot address (BOOT_AD<31:0>) becomes valid first, then BOOT_ALE deasserts, then BOOT_CE_*n*_L asserts, then BOOT_OE_L asserts.

CN50XX stops driving the upper boot address bits (BOOT_AD<31:16>) at the same time that BOOT_OE_L asserts. CN50XX samples the boot data as it deasserts BOOT_OE_L. After that, BOOT_CE_*n*_L deasserts, then the boot address returns to 0s. CN50XX drives the boot data after it deasserts BOOT_CE_*n*_L.

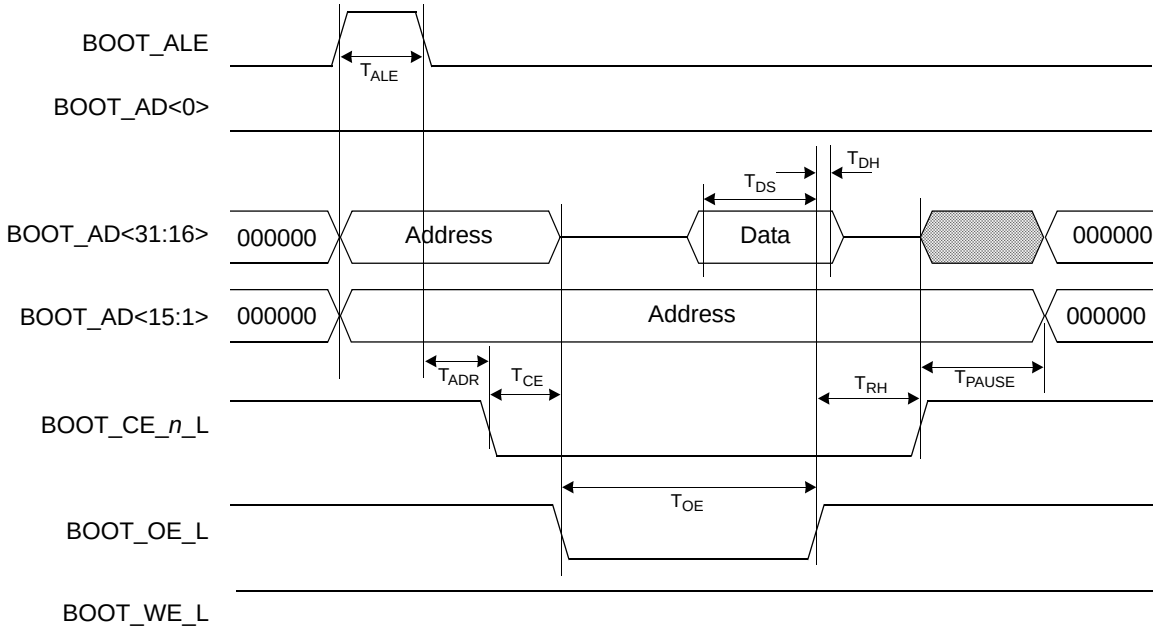


Figure 12–5 Static-Timed Read Sequence (ALE, 16W)

12.4.2 Static-Timed Write Sequences

Static-timed write operations can be performed in the following modes:

- eight-bit, nonmultiplexed
- eight-bit, multiplexed
- 16-bit, nonmultiplexed
- 16-bit, multiplexed

Static-Timed Write: Not ALE, 8W

Figure 12–6 shows a boot-bus write transaction in eight-bit, nonmultiplexed mode. All boot-bus write transactions are single-byte when MIO_BOOT_REG_CFG*n*[WIDTH] = 0. The boot address (BOOT_AD<23:0>) and boot data (BOOT_AD<31:24>) become valid first, then BOOT_CE_*n*_L asserts and BOOT_WE_L asserts. After a time delay, BOOT_WE_L deasserts, BOOT_CE_*n*_L deasserts, and finally the boot address and boot data return to 0s.

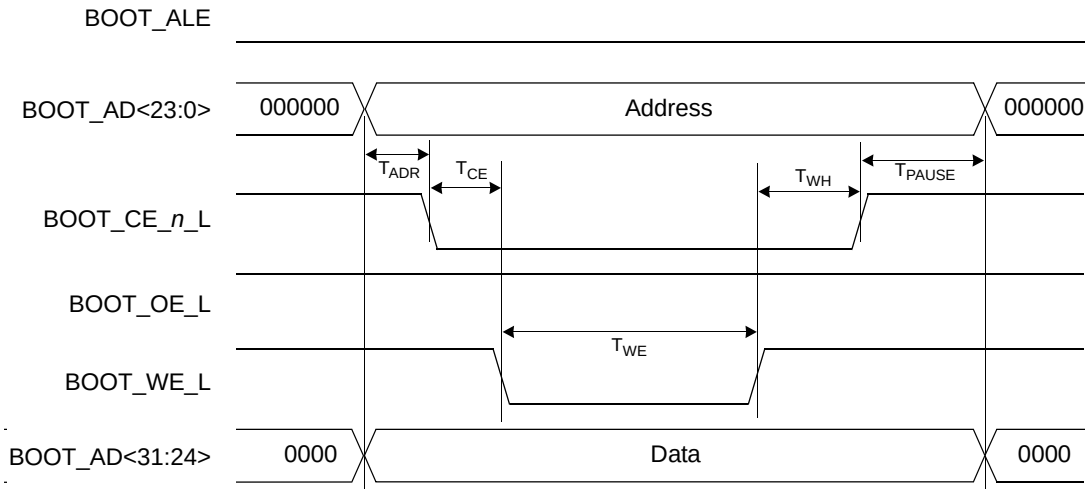


Figure 12–6 Static-Timed Write Sequence (not ALE, 8W)

Static-Timed Write: ALE, 8W

Figure 12–7 shows a boot-bus write transaction in eight-bit, multiplexed mode. All boot-bus write transactions are single-byte when MIO_BOOT_REG_CFG*n*[WIDTH] = 0.

BOOT_ALE asserts and the boot address (BOOT_AD<31:0>) becomes valid. Then BOOT_CE_*n*_L asserts and boot data (BOOT_AD<31:24>) becomes valid, and BOOT_WE_L asserts. After a time delay, BOOT_WE_L deasserts, BOOT_CE_*n*_L deasserts, and finally the boot address and boot data return to 0s.

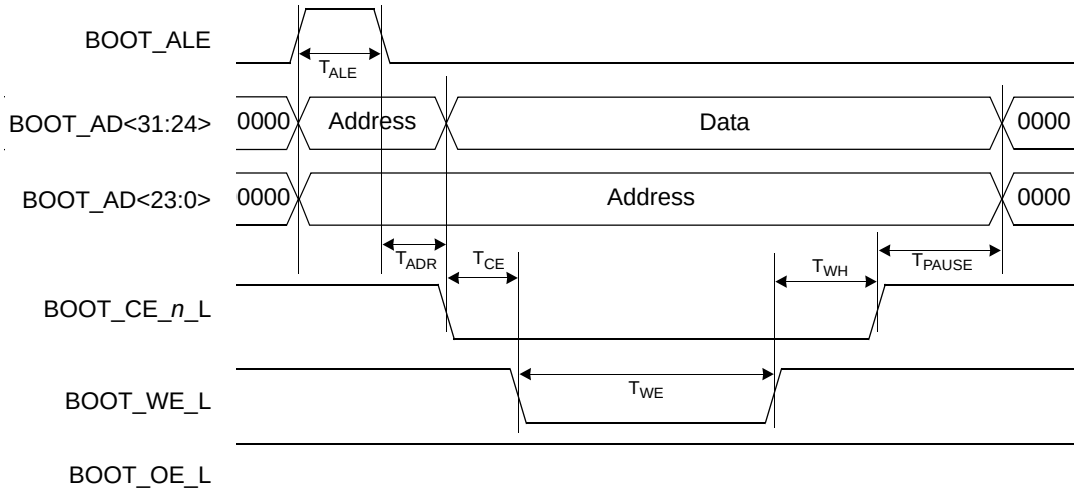


Figure 12–7 Static-Timed Write Sequence (ALE, 8W)

Static-Timed Write: Not ALE, 16W

Figure 12–8 shows a boot-bus write transaction in 16-bit, nonmultiplexed mode. All boot-bus write transactions are two-byte when MIO_BOOT_REG_CFG*n*[WIDTH] = 1. The boot address (BOOT_AD<15:0>) and boot data (BOOT_AD<31:16>) become valid first, then BOOT_CE_*n*_L asserts and BOOT_WE_L asserts. After a time delay, BOOT_WE_L deasserts, BOOT_CE_*n*_L deasserts, and finally the boot address and boot data return to 0s.

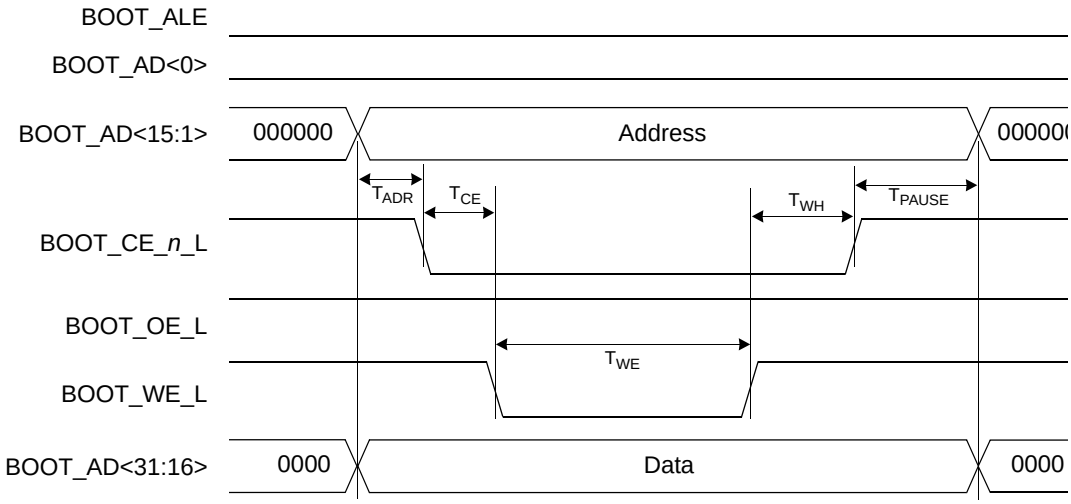


Figure 12–8 Static-Timed Write Sequence (not ALE, 16W)

Static-Timed Write: ALE, 16W

Figure 12–9 shows a boot-bus write transaction in 16-bit, multiplexed mode. All boot-bus write transactions are two-byte when MIO_BOOT_REG_CFG*n*[WIDTH] = 1.

BOOT_ALE asserts and the boot address (BOOT_AD<31:0>) becomes valid. Then BOOT_CE_*n*_L asserts and boot data (BOOT_AD<31:16>) becomes valid, and BOOT_WE_L asserts. After a time delay, BOOT_WE_L deasserts, BOOT_CE_*n*_L deasserts, and finally the boot address and boot data return to 0s.

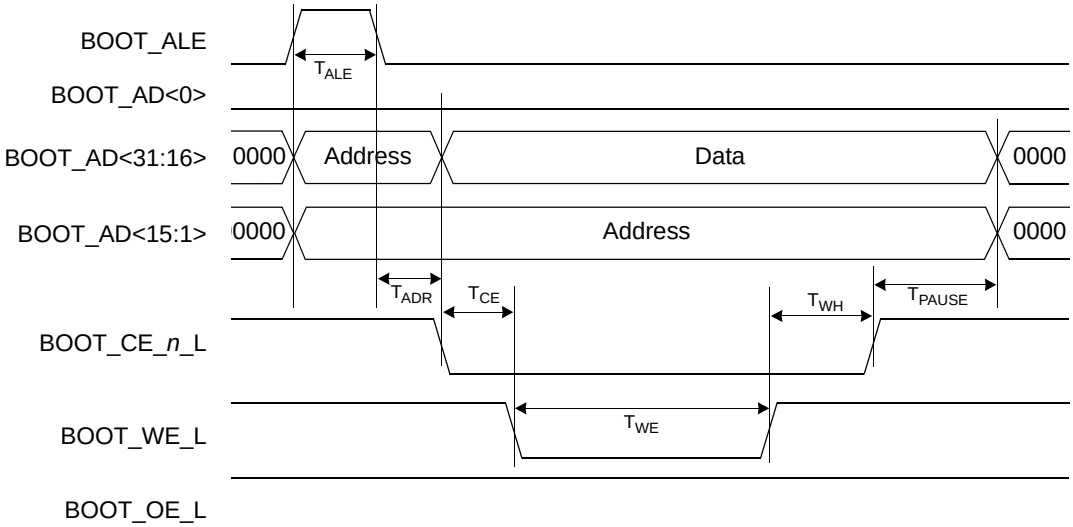


Figure 12–9 Static-Timed Write Sequence (ALE, 16W)

12.4.3 Static-Timed Page-Read Sequences

Static-timed page-read operations can be performed in the following modes:

- eight-bit, nonmultiplexed
- eight-bit, multiplexed
- 16-bit, nonmultiplexed
- 16-bit, multiplexed

Static-Timed Page-Read: Not ALE, 8W

Figure 12–10 shows a boot-bus page read transaction with a four-byte length in eight-bit, nonmultiplexed mode. When MIO_BOOT_REG_TIM*n*[PAGEM] = 1 and MIO_BOOT_REG_CFG*n*[WIDTH] = 0, 64-bit, 32-bit, and 16-bit read transactions are page-mode read transactions. This example shows a page of four bytes. Two- and eight-byte pages are also possible.

The transaction is the same as the single-byte transaction in Figure 12–2 up to the point when T_{OE} would expire. Instead, at that point, BOOT_OE_L remains asserted and the boot address transitions through the next three sequential addresses (bits of lesser significance in the word since the bus is big-endian).

CN50XX samples the boot data (BOOT_AD<31:24>) for the previous address as it transitions to the next address. BOOT_OE_L deasserts as boot data is sampled the last time, and the remainder of the transaction is identical to the single-byte case.

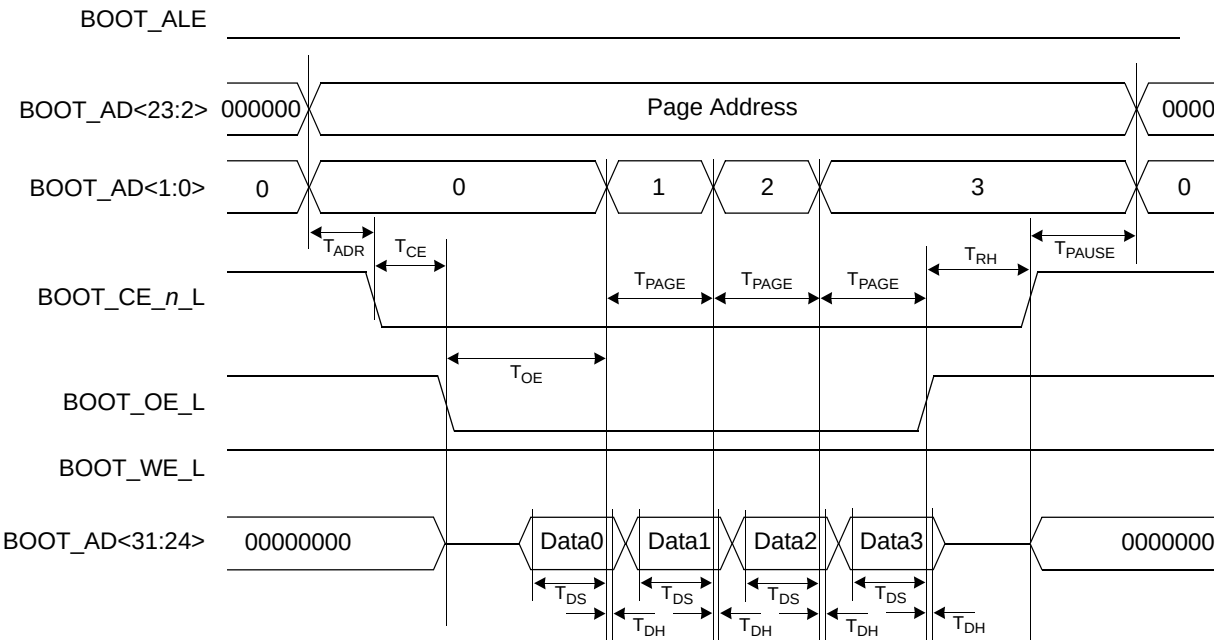


Figure 12–10 Static-Timed Page-Read Sequence (Four-Byte Page) (not ALE, 8W)

Static-Timed Page-Read: ALE, 8W

Figure 12–11 shows a boot-bus page read transaction with a four-byte length in eight-bit, multiplexed mode. When MIO_BOOT_REG_TIM*n*[PAGEM] = 1 and MIO_BOOT_REG_CFG*n*[WIDTH] = 0, 64-bit, 32-bit, and 16-bit read transactions are page-mode read transactions. This example shows a page of four bytes. Two- and eight-byte pages are also possible.

The transaction is the same as the single-byte transaction in Figure 12–3 up to the point when T_{OE} would expire. Instead, at that point, BOOT_OE_L remains asserted and the boot address transitions through the next three sequential addresses (bits of lesser significance in the word since the bus is big-endian).

CN50XX samples the boot data (BOOT_AD<31:24>) for the previous address as it transitions to the next address. BOOT_OE_L deasserts as boot data is sampled the last time, and the remainder of the transaction is identical to the single-byte case.

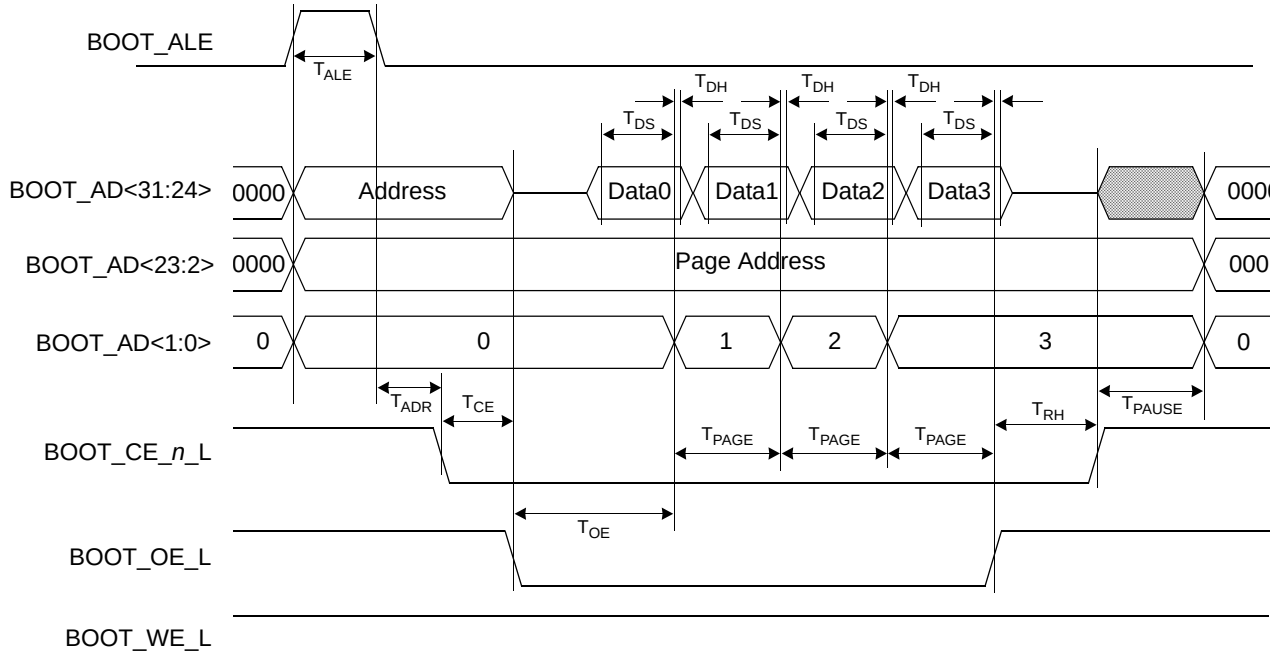


Figure 12–11 Static-Timed Page-Read Sequence (Four-Byte Page) (ALE, 8W)

Static-Timed Page-Read: Not ALE, 16W

Figure 12–12 shows a boot-bus page read transaction with an eight-byte length in 16-bit, nonmultiplexed mode. When MIO_BOOT_REG_TIM*n*[PAGEM] = 1 and MIO_BOOT_REG_CFG*n*[WIDTH] = 1, 64-bit and 32-bit read transactions are page-mode read transactions. This example shows a page of eight bytes. Four-byte pages are also possible.

The transaction is the same as the two-byte transaction in Figure 12–4 up to the point when T_{OE} would expire. Instead, at that point, BOOT_OE_L remains asserted and the boot address transitions through the next three sequential addresses (bits of lesser significance in the word since the bus is big-endian).

CN50XX samples the boot data (BOOT_AD<31:16>) for the previous address as it transitions to the next address. BOOT_OE_L deasserts as boot data is sampled the last time, and the remainder of the transaction is identical to the two-byte case.

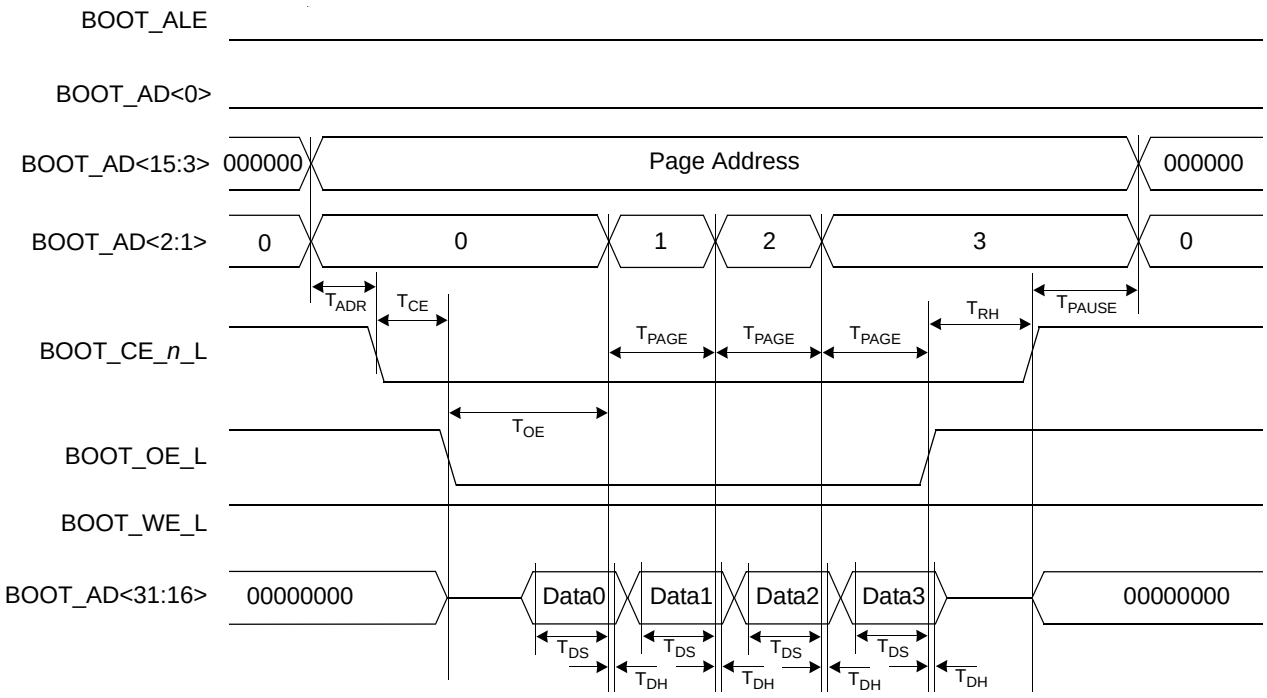


Figure 12–12 Static-Timed Page-Read Sequence (Eight-Byte Page) (not ALE, 16W)

Static-Timed Page-Read: ALE, 16W

Figure 12–12 shows a boot-bus page read transaction with an eight-byte length in 16-bit, multiplexed mode. When MIO_BOOT_REG_TIM*n*[PAGEM] = 1 and MIO_BOOT_REG_CFG*n*[WIDTH] = 1, 64-bit and 32-bit read transactions are page-mode read transactions. This example shows a page of eight bytes. Four-byte pages are also possible.

The transaction is the same as the two-byte transaction in Figure 12–5 up to the point when T_{OE} would expire. Instead, at that point, BOOT_OE_L remains asserted and the boot address transitions through the next three sequential addresses (bits of lesser significance in the word since the bus is big-endian).

CN50XX samples the boot data (BOOT_AD<31:16>) for the previous address as it transitions to the next address. BOOT_OE_L deasserts as boot data is sampled the last time, and the remainder of the transaction is identical to the two-byte case.

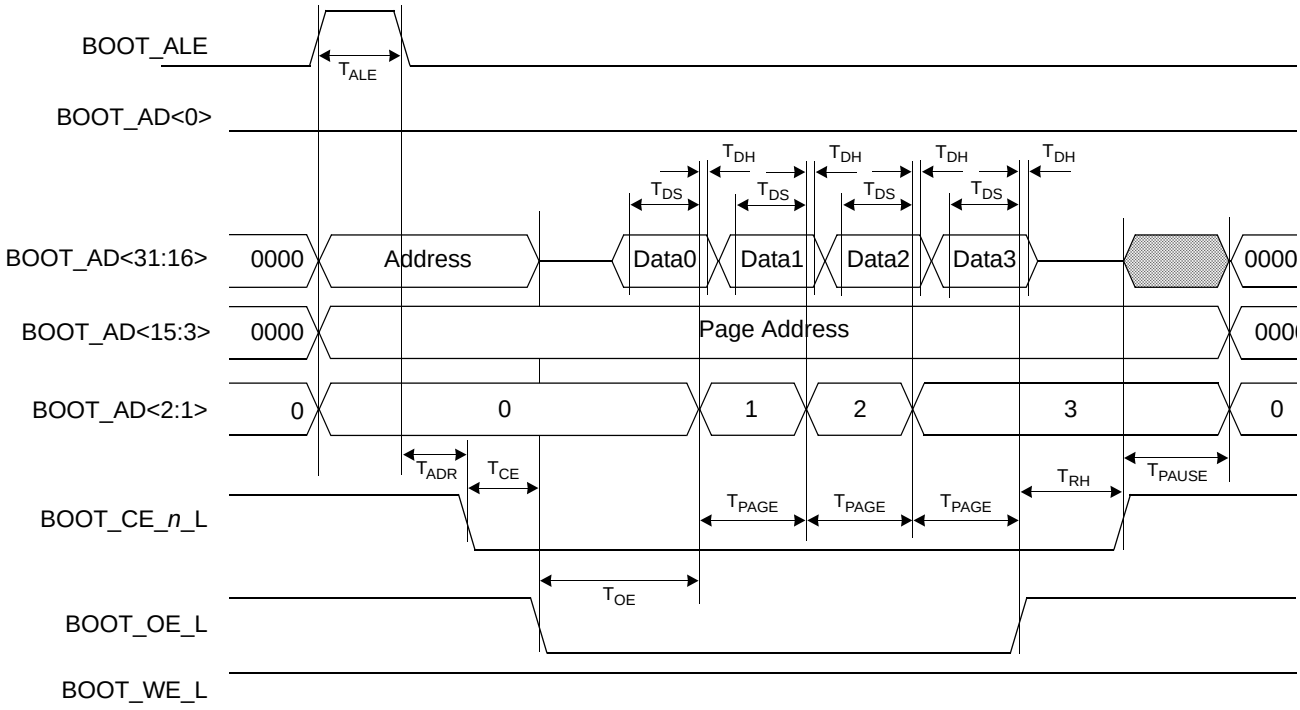


Figure 12–13 Static-Timed Page-Read Sequence (Eight-Byte Page) (ALE, 16W)

12.4.4 Dynamic-Timed Sequences

Dynamic-timed read and write operations can be performed in the following modes:

- eight-bit, nonmultiplexed
- eight-bit, multiplexed
- 16-bit, nonmultiplexed
- 16-bit, multiplexed

The following examples are shown:

- eight-bit, nonmultiplexed read operation
- eight-bit, nonmultiplexed write operation

Dynamic-Timed Read Sequence (not ALE, 8W)

Figure 12–14 shows a dynamic-timed read transaction (i.e. when MIO_BOOT_REG_TIM*n*[WAITM] = 1). This example is in eight-bit, non-multiplexed mode, but multiplexed and 16-bit dynamic-timed read sequences are also possible. This transaction is almost identical to the single-byte read transaction shown in Figure 12–2, except for the delay in the deassertion of BOOT_OE_L caused by the target device asserting BOOT_WAIT_L. Note that BOOT_WAIT_L must be asserted before T_{OE} would normally expire, otherwise the transaction will not be extended. After T_{OE} has expired and BOOT_WAIT_L has been deasserted for T_{WAIT} cycles, BOOT_OE_L deasserts and the transaction continues as a normal single byte read. Note that if BOOT_WAIT_L is still asserted 2^{15} cycles after T_{OE} would normally expire, the transaction completes as normal (though presumably with bad read data) and MIO_BOOT_ERR[WAIT_ERR] is set.

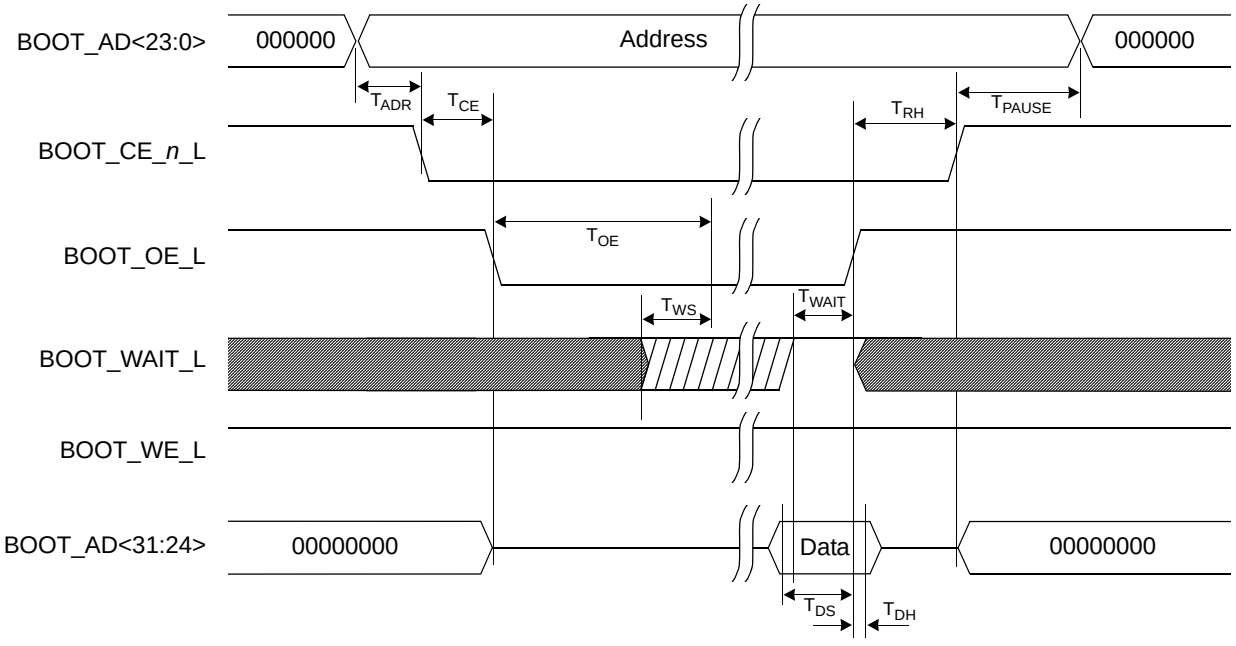


Figure 12–14 Dynamic-Timed Read Sequence (not ALE, 8W)

Dynamic-Timed Write Sequence (not ALE, 8W)

Figure 12–15 shows a dynamic-timed write transaction (i.e. when MIO_BOOT_REG_TIM*n*[WAITM] = 1). This example is in eight-bit, non-multiplexed mode, but multiplexed and 16-bit dynamic-timed write sequences are also possible. This transaction is almost identical to the byte-write transaction shown in Figure 12–7, except for the delay in the deassertion of BOOT_WE_L caused by the target device asserting BOOT_WAIT_L. Note that BOOT_WAIT_L must be asserted before T_{WE} would normally expire, otherwise the transaction will not be extended. After T_{WE} has expired and BOOT_WAIT_L has been deasserted for T_{WAIT} cycles, BOOT_WE_L deasserts and the transaction continues as a normal byte write. Note that if BOOT_WAIT_L is still asserted 2^{15} cycles after T_{WE} would normally expire, the transaction completes as normal (though presumably the write will not complete in the target) and MIO_BOOT_ERR[WAIT_ERR] is set.

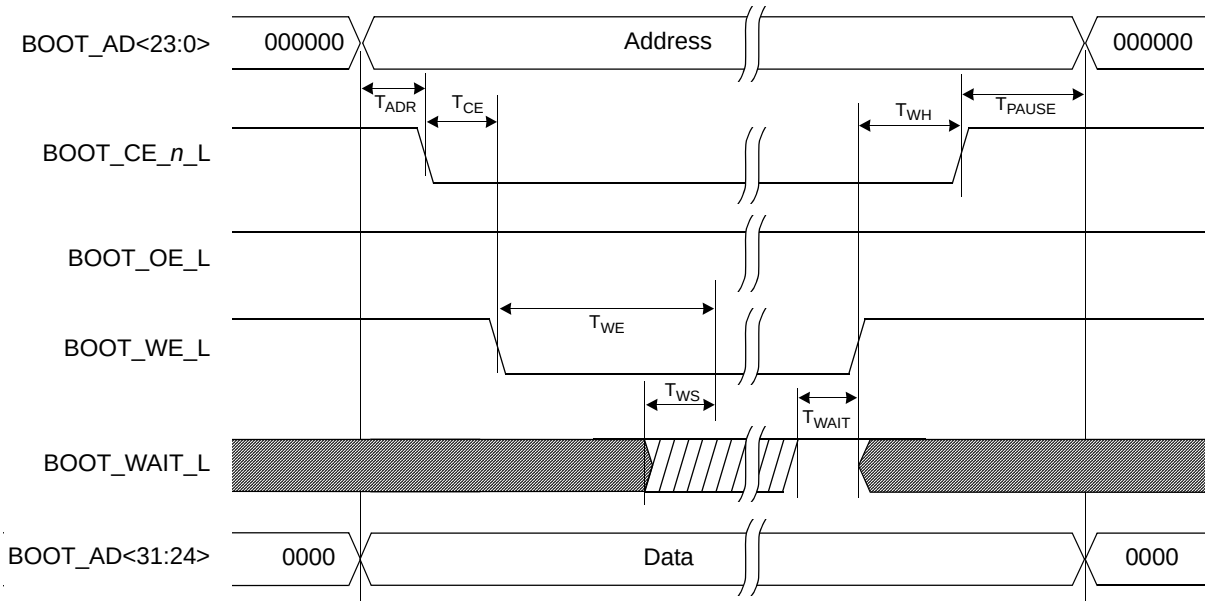


Figure 12–15 Dynamic-Timed Write Sequence (not ALE, 8W)

12.5 Boot-Bus Request Queuing

The boot-bus logic has a FIFO that holds load/store operations destined for the boot bus. When

FIFO size ≥ MIO_BOOT_THR[FIF_THR],

the boot-bus logic applies backpressure, and no more CSR references (destined anywhere in CN50XX) can complete. Each load/store operation takes one FIFO entry. It can take a long time for this queue to drain, depending on the timing of the devices on the bus. Overflow of this FIFO should preferably be avoided when fast response from CN50XX is required. Load operations cannot cause the FIFO to overflow, since each core has at most one load outstanding, but a single core can overflow the FIFO with IOBDMA or store operations.

MIO_BOOT_THR[FIF_CNT] contains the current FIFO size. Software can avoid the CN50XX performance effects of overflowing this FIFO by monitoring the current size of the FIFO after a series of IOBDMAs/store operations.

12.6 Boot-Bus Connections

Figure 12–16 shows a sample connection from the boot bus to a 16-bit-wide, 256MB flash memory, using BOOT_CE_0_L. Note the pullup resistors on BOOT_AD<15, 14> to enable 16-bit and multiplexed booting.

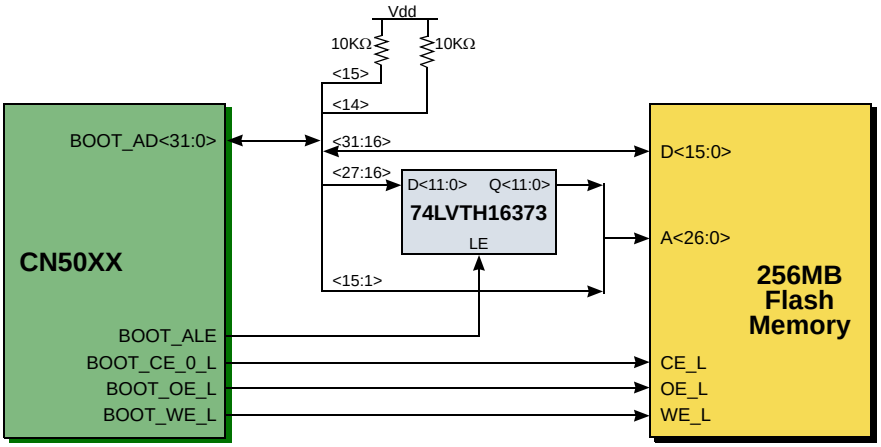


Figure 12–16 Sample Boot-Bus Connection 1 (ALE, 16W)

Figure 12–17 shows a sample connection from the boot bus to an 8-bit-wide, 16MB flash memory, nonmultiplexed, using BOOT_CE_0_L.

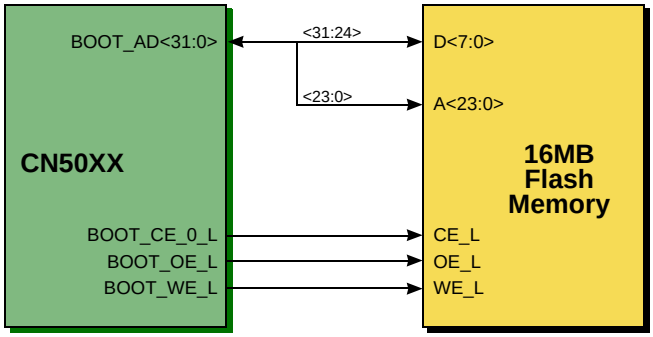


Figure 12–17 Sample Boot-Bus Connection 2 (not ALE, 8W)

Figure 12–18 shows a sample connection from the boot bus to an 16-bit-wide, compact flash memory, nonmultiplexed, using BOOT_CE_3_L.

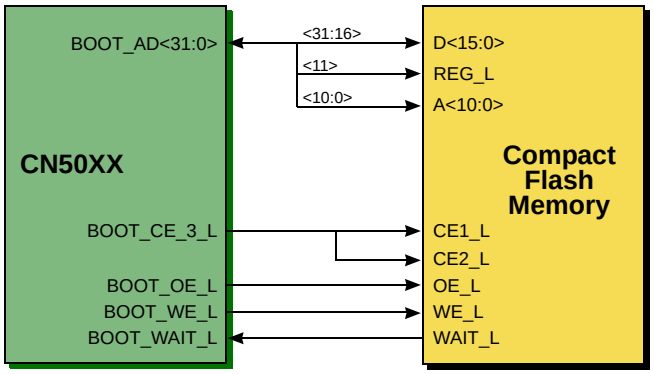


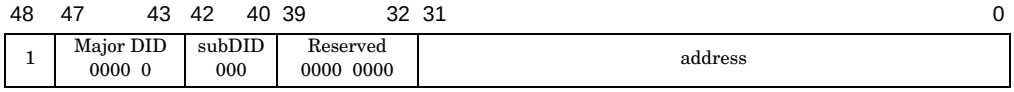
Figure 12–18 Sample Boot-Bus Connection 3 (not ALE, 16W)

12.7 Boot-Bus Operations

The boot bus performs load, IOBDMA, and store operations, which are described in the following subsections.

12.7.1 Load Operations

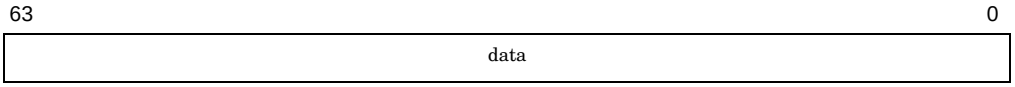
Load Address Field



- **address -** The address to be matched to local cache and bus regions.
  - for 64-bit load operations: address<2:0> must be 000.
  - for 32-bit load operations: address<1:0> must be 00.
  - for 16-bit load operations: address<0> must be 0.

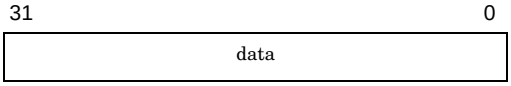
Load Result Field

64-bit operation result



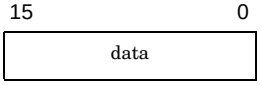
- **data -** data is the result of the boot-bus load, in big-endian format.

32-bit operation result



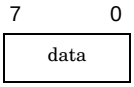
- **data -** data is the result of the boot-bus load operation, in big-endian format.

16-bit operation result



- **data -** data is the result of the boot-bus load operation, in big-endian format.

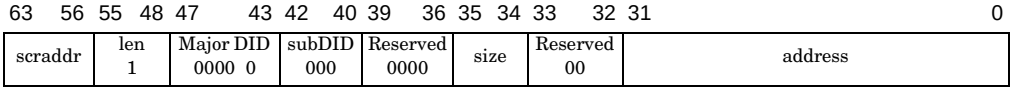
8-bit operation result



- **data -** data is the result of the boot-bus load operation.

12.7.2 IOBDMA Operations

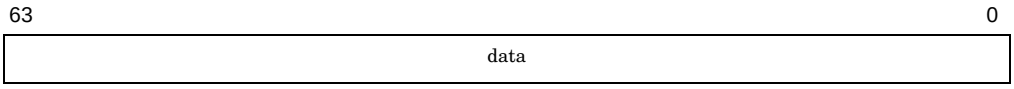
IOBDMA Address Field



- **scraddr -** Defined in "cnMIPS™ Core" on page 147.
- **len -** Must be 1. Defined in "cnMIPS™ Core" on page 147.
- **size** - Indicates the size of the operation
  00 = 8-byte operation
  01 = 4-byte operation
  10 = 2-byte operation
  11 = 1-byte operation
- **address -** The address to be matched to local cache and bus regions.
  - for 64-bit IOBDMA operations: address<2:0> must be 000.
  - for 32-bit IOBDMA operations: address<1:0> must be 00.
  - for 16-bit IOBDMA operations: address<0> must be 0.

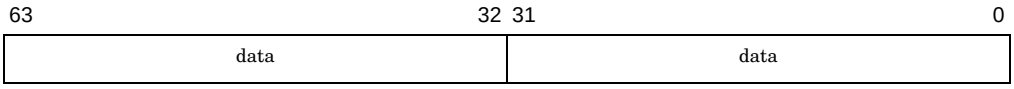
IOBDMA Result Field

8-byte IOBDMA operation result



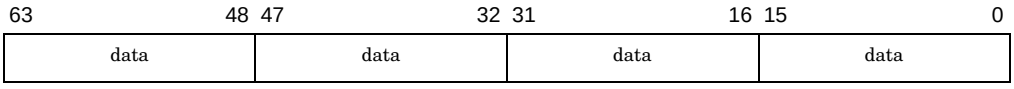
- **data -** data is the result of the boot-bus read operations totalling 8 bytes, in big-endian format.

4-byte IOBDMA operation result



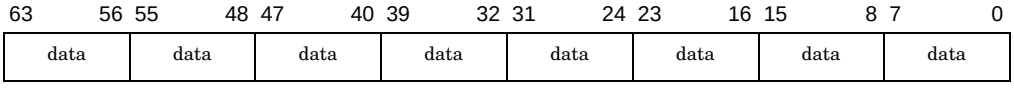
- **data -** data is the result of the boot-bus read operations totalling 4 bytes, in big-endian format, duplicated in <63:32> and <31:0>.

2-byte IOBDMA operation result



- **data -** data is the result of the boot-bus read operations totalling 2 bytes, in big-endian format, duplicated in <63:48>, <47:32>, <31:16>, and <15:0>.

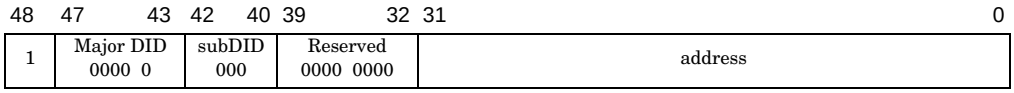
1-byte IOBDMA operation result



- **data -** data is the result of the boot-bus read operations totalling 8 bytes, in big-endian format, duplicated in all bytes.

12.7.3 Store Operations

Store Address Field



- **address -** The address to be matched to local cache and bus regions.
  - for 64-bit store operations: address<2:0> must be 000.
  - for 32-bit store operations: address<1:0> must be 00.
  - for 16-bit store operations: address<0> must be 0.

The boot-bus hardware writes the store data onto the boot bus in big-endian format.

12.8 Boot-Bus Registers

The boot-bus registers are listed in Table 12–6.

Table 12–6 Boot-Bus Registers

Register	Address	CSR Type[1]	Detailed Description
MIO_BOOT_REG_CFG0	0x0001180000000000	RSL	See page 503
MIO_BOOT_REG_CFG1 ... MIO_BOOT_REG_CFG7	0x0001180000000008 ... 0x0001180000000038	RSL	See page 504
MIO_BOOT_REG_TIM0	0x0001180000000040	RSL	See page 505
MIO_BOOT_REG_TIM1 ... MIO_BOOT_REG_TIM7	0x0001180000000048 ... 0x0001180000000078	RSL	See page 505
MIO_BOOT_LOC_CFG0	0x0001180000000080	RSL	See page 506
MIO_BOOT_LOC_CFG1	0x0001180000000088	RSL	See page 506
MIO_BOOT_LOC_ADR	0x0001180000000090	RSL	See page 506
MIO_BOOT_LOC_DAT	0x0001180000000098	RSL	See page 506
MIO_BOOT_ERR	0x00011800000000A0	RSL	See page 507
MIO_BOOT_INT	0x00011800000000A8	RSL	See page 507
MIO_BOOT_THR	0x00011800000000B0	RSL	See page 507
MIO_BOOT_COMP	0x00011800000000B8	RSL	See page 508
MIO_BOOT_BIST_STAT	0x00011800000000F8	RSL	See page 508

1. RSL-type registers are accessed indirectly across the I/O bus.

MIO Boot Region 0 Configuration Register MIO_BOOT_REG_CFG0

The region 0 configuration register contains configuration parameters for boot region 0. See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:42>	—	RAZ	—	—	Reserved.
<41:40>	TIM_MULT	R/W	0x0	—	Region 0 timing multiplier, specifies the timing multiplier for region 0. The timing multiplier applies to all timing parameters, except for WAIT and RD_DLY, which simply count core-clock cycles. 00 = 4×, 01 = 1×, 10 = 2×, 11 = 8×.
<39:37>	RD_DLY	R/W	0x0	—	Region 0 read sample delay, specifies the read sample delay in core-clock cycles for region 0. For read operations, the data bus is normally sampled on the same core-clock edge that drives BOOT_OE_N to the inactive state (or the core-clock edge that toggles the lower address bits in page mode). This parameter can delay that sampling edge by up to 7 core-clock cycles. **NOTE:** The number of core-clock cycles counted by the PAGE and RD_HLD timing parameters must be greater than RD_DLY.
<36>	SAM	R/W	0	—	Region 0 strobe and mode. When asserted, this field internally combines the output-enable and write-enable strobes into a single strobe that is then driven onto both BOOT_OE_and BOOT_WE_L. This is useful for parts that use a single strobe along with a read/write bit that can be driven from an address signal.
<35:34>	WE_EXT	R/W	0x0	—	Region 0 write-enable count extension.
<33:32>	OE_EXT	R/W	0x0	—	Region 0 output-enable count extension.
<31>	EN	R/W	1	1	Region 0 enable.
<30>	OR	R/W	0	0	Reserved (since there is no previous region).
<29>	ALE[1]	R/W	0	—	Region 0 address-latch enable. 0 = address/data bus not multiplexed 1 = address/data bus multiplexed
<28>	WIDTH[2]	R/W	0	—	Region 0 data-bus width: 0 = 8-bit, 1 = 16-bit.
<27:16>	SIZE	R/W	0xFFF	—	Region 0 size. Region size is specified in 64K blocks and in “block–1” notation (i.e. 0 = 1 64K block, 1 = 2 64K blocks, etc.).
<15:0>	BASE	R/W	0x1FC0	0x1FC0	Region 0 base address. Specifies address bits [31:16] of the first 64K block of the region.

1. The reset value is the value of BOOT_AD15 at the deassertion of reset. BOOT_AD15 has an internal pulldown resistor, so you must place an external pullup resistor on it to enable multiplexing out of reset for region 0.
2. The reset value is the value of BOOT_AD14 at the deassertion of reset. BOOT_AD14 has an internal pulldown resistor, so you must place an external pullup resistor on it to enable a 16-bit bus out of reset for region 0.

MIO Boot Regions 1..7 Configuration Registers
MIO_BOOT_REG_CFG(1..7)

The region *n* configuration register (one register for each of seven regions) contains configuration parameters for boot region *n*. See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:42>	—	RAZ	—	—	Reserved.
<41:40>	TIM_MULT	R/W	0x0	—	Region 1–7 timing multiplier, specifies the timing multiplier for region 0. The timing multiplier applies to all timing parameters, except for WAIT and RD_DLY, which simply count core-clock cycles. 00 = 4×, 01 = 1×, 10 = 2×, 11 = 8×.
<39:37>	RD_DLY	R/W	0x0	—	Region 1–7 read sample delay, specifies the read sample delay in core-clock cycles for region 0. For read operations, the data bus is normally sampled on the same core-clock edge that drives BOOT_OE_N to the inactive state (or the core-clock edge that toggles the lower address bits in page mode). This parameter can delay that sampling edge by up to 7 core-clock cycles. **NOTE:** The number of core-clock cycles counted by the PAGE and RD_HLD timing parameters must be greater than RD_DLY.
<36>	SAM	R/W	0	—	Region 1–7 strobe and mode. When asserted, this field internally combines the output-enable and write-enable strobes into a single strobe that is then driven onto both BOOT_OE_and BOOT_WE_L. This is useful for parts that use a single strobe along with a read/write bit that can be driven from an address signal.
<35:34>	WE_EXT	R/W	0x0	—	Region 1–7 write-enable count extension.
<33:32>	OE_EXT	R/W	0x0	—	Region 1–7 output-enable count extension.
<31>	EN	R/W	0	1	Region 1–7 enable.
<30>	OR	R/W	0	0	Region 1–7 OR bit. Asserts the given region's chip enable when there is an address hit in the previous region. This is useful for CF cards because it allows the use of 2 separate timing configurations for common memory and attribute memory.
<29>	ALE	R/W	0	—	Region 1–7 address-latch enable. 0 = address/data bus not multiplexed 1 = address/data bus multiplexed
<28>	WIDTH	R/W	0	—	Region 1–7 data-bus width: 0 = 8-bit, 1 = 16-bit.
<27:16>	SIZE	R/W	0x0	—	Region 1–7 size. Region size is specified in 64K blocks and in "block–1" notation (i.e. 0 = 1 64K block, 1 = 2 64K blocks, etc.).
<15:0>	BASE	R/W	0x0	—	Region 1–7 base address. Specifies address bits [31:16] of the first 64K block of the region.

MIO Boot Timing Register for Region 0
MIO_BOOT_REG_TIM0

The region 0 timing register contains page-mode, wait-mode, and timing parameters for region 0. Note that [OE], [WE], [PAGE], and [ALE] must be non-zero to ensure legal transitions on the corresponding boot bus outputs. The [WAIT] field must be non-zero if [WAITM] is set. See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	PAGEM	R/W	0	—	Region 0 page-mode enable. (See Table 12–4.)
<62>	WAITM	R/W	0	—	Region 0 wait-mode enable. (See Table 12–4.)
<61:60>	PAGES	R/W	0x0	—	Region 0 page size.
<59:54>	ALE	R/W	4	—	Region 0 ALE count.
<53:48>	PAGE	R/W	0x3F	—	Region 0 page count.
<47:42>	WAIT	R/W	0x3F	—	Region 0 wait count, must be nonzero when WAITM is set to 1.
<41:36>	PAUSE	R/W	0x11	—	Region 0 pause count.
<35:30>	WR_HLD	R/W	0x3F	—	Region 0 write hold count.
<29:24>	RD_HLD	R/W	0x5	—	Region 0 read hold count.
<23:18>	WE	R/W	0x3F	—	Region 0 write enable count.
<17:12>	OE	R/W	0x3F	—	Region 0 output enable count.
<11:6>	CE	R/W	0x5	—	Region 0 chip enable count.
<5:0>	ADR	R/W	0x8	—	Region 0 address count.

MIO Boot Timing Registers for Regions 1-7
MIO_BOOT_REG_TIM(1..7)

The region *n* timing register (one register for each of seven regions) contains page-mode, wait-mode, and timing parameters for region *n*. Note that [OE], [WE], [PAGE], and [ALE] must be non-zero to ensure legal transitions on the corresponding boot bus outputs. The [WAIT] field must be non-zero if [WAITM] is set. See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	PAGEM	R/W	0	—	Region 1–7 page-mode enable. (See Table 12–4.)
<62>	WAITM	R/W	0	—	Region 1–7 wait-mode enable. (See Table 12–4.)
<61:60>	PAGES	R/W	0x0	—	Region 1–7 page size.
<59:54>	ALE	R/W	0x3F	—	Region 1–7 ALE count.
<53:48>	PAGE	R/W	0x3F	—	Region 1–7 page count
<47:42>	WAIT	R/W	0x3F	—	Region 1–7 wait count, must be nonzero when WAITM is set to 1.
<41:36>	PAUSE	R/W	0x3F	—	Region 1–7 pause count
<35:30>	WR_HLD	R/W	0x3F	—	Region 1–7 write-hold count
<29:24>	RD_HLD	R/W	0x3F	—	Region 1–7 read-hold count
<23:18>	WE	R/W	0x3F	—	Region 1–7 write-enable count
<17:12>	OE	R/W	0x3F	—	Region 1–7 output-enable count
<11:6>	CE	R/W	0x3F	—	Region 1–7 chip-enable count
<5:0>	ADR	R/W	0x3F	—	Region 1–7 address count

MIO Boot Local-Region Configuration Registers
MIO_BOOT_LOC_CFG0/1

The local-region configuration register (one for each of two regions) contains local-region enable and local-region base-address parameters. Each local region is 128 bytes organized as 16 entries × 8 bytes.

See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31>	EN	R/W	0	—	Local region 0/1 enable
<30:28>	—	RAZ	—	—	Reserved
<27:3>	BASE	R/W	0x0	—	Local region 0/1 base address, specifying address bits [31:7] of the region.
<2:0>	—	RAZ	—	—	Reserved

MIO Boot Local-Memory Address Register
MIO_BOOT_LOC_ADR

The local-region memory-address register specifies the address for reading or writing the local memory. This address post-increments following an access to the MIO boot local-memory data register.

- Local-memory region 0 is addresses 0x00–0x78.
- Local-memory region 1 is addresses 0x80–0xF8.

See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved
<7:3>	ADR	R/W	0x0	—	Local memory address
<2:0>	—	RAZ	—	—	Reserved

MIO Boot Local-Memory Data Register
MIO_BOOT_LOC_DAT

This is a pseudo-register that reads/writes the local memory at the address specified by the MIO boot local-memory address register when accessed.

See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	DATA	R/W	—	—	Local memory data

MIO Boot-Error Register
MIO_BOOT_ERR

The boot-error register contains the address decode error and wait mode error bits. See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved
<1>	WAIT_ERR	R/W1C	0	0	Wait mode error. This bit is set when wait mode is enabled and the external wait signal is not deasserted after 16K ECLK (core clock) cycles.
<0>	ADR_ERR	R/W1C	0	0	Address decode error. This bit is set when a boot-bus access does not hit in any of the eight remote regions or two local regions.

MIO Boot-Interrupt Register
MIO_BOOT_INT

The boot-interrupt register contains the interrupt-enable bits for address-decode errors and wait-mode errors. See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved
<1>	WAIT_INT	R/W	0	—	Wait-mode error interrupt enable.
<0>	ADR_INT	R/W	0	—	Address-decode error interrupt enable.

MIO Boot-Threshold Register
MIO_BOOT_THR

The boot-threshold register contains MIO boot-threshold values and should never be written.

See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:14>	—	RAZ	—	—	Reserved.
<13:8>	FIF_CNT	RO	0x0	—	FIFO count. Contains the current IOB FIFO count.
<7:6>	—	RAZ	—	—	Reserved.
<5:0>	FIF_THR	RO	0x1A	0x1A	IOB busy threshold. Should always read 0x1A (the only legal value).

MIO Boot Compensation Register
MIO_BOOT_COMP

See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:10>	—	RAZ	—	—	Reserved.
<9:5>	PCTL	R/W	0x1F	—	Boot bus PCTL. This controls the pull-up drive strength of the boot-bus drivers. • 0x1F = Set full strength for boot-bus drivers. • 0x0F = Set 20Ω output impedance for boot-bus drivers. • 0x08 = Set 50Ω output impedance for boot-bus drivers.
<4:0>	NCTL	R/W	0x1F	—	Boot bus NCTL. This controls the pull-down drive strength of the boot-bus drivers. • 0x1F = Set full strength for boot-bus drivers. • 0x19 = Set 20Ω output impedance for boot-bus drivers. • 0x06 = Set 50Ω output impedance for boot-bus drivers.

MIO Boot BIST Status Register
MIO_BOOT_BIST_STAT

The boot BIST status register contains the BIST status for the MIO boot memories: 0 = pass, 1 = fail. See Table 12–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	—	—	Reserved.
<5>	PCM_1	RO	0	0	PCM memory 1 BIST status.
<4>	PCM_0	RO	0	0	PCM memory 0 BIST status.
<3>	NCBO_1	RO	0	0	IOB output-FIFO 1 BIST status.
<2>	NCBO_0	RO	0	0	IOB output-FIFO 0 BIST status.
<1>	LOC	RO	0	0	Local memory BIST status.
<0>	NCBI	RO	0	0	IOB input-FIFO BIST status.

Chapter 13

CN50XX Packet Interface

This chapter contains the following subjects:

Overview

13.1 Packet Interface Introduction

The CN50XX has a packet interface that can be configured in gigabit media-independent interface (GMII) mode, media-independent interface (MII) mode, or reduced gigabit media-independent interface (RGMII) mode. The packet interface supports 0–3 independent links or ports. The link configuration is chosen by the GMX0_INF_MODE register as shown in Table 13–1.

Table 13–1 Packet Interface Configuration

GMX0_INF_MODE			Configuration
[EN]	[TYPE]	[P0MII]	
0	X	X	All links are disabled.
Port 0 Configure			
1	X	0	Port 0 is RGMII
1	X	1	Port 0 is MII
Ports 1 and 2[1] Configure			
1	0	X	Ports 1 and 2 are configured as RGMII ports.
1	1	X	Port 1: GMII/MII; Port 2: disabled. GMII or MII port is selected by GMX_PRT1_CFG[SPEED].

1. Port 2 is in CN3010 only.

The GMII and MII links are fully compliant with the IEEE 802.3 specification and the RGMII links are fully compliant with both the IEEE 802.3 and HP RGMII 1.3 specifications.

Figure 13–1 shows the physical difference between the GMII, MII, and RGMII links. Figure 13–2 shows the different operating modes.

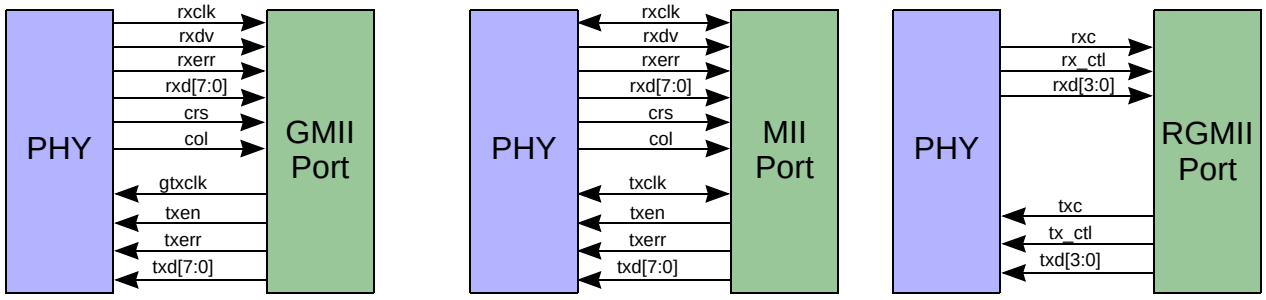


Figure 13–1 GMII, MII, and RGMII Links

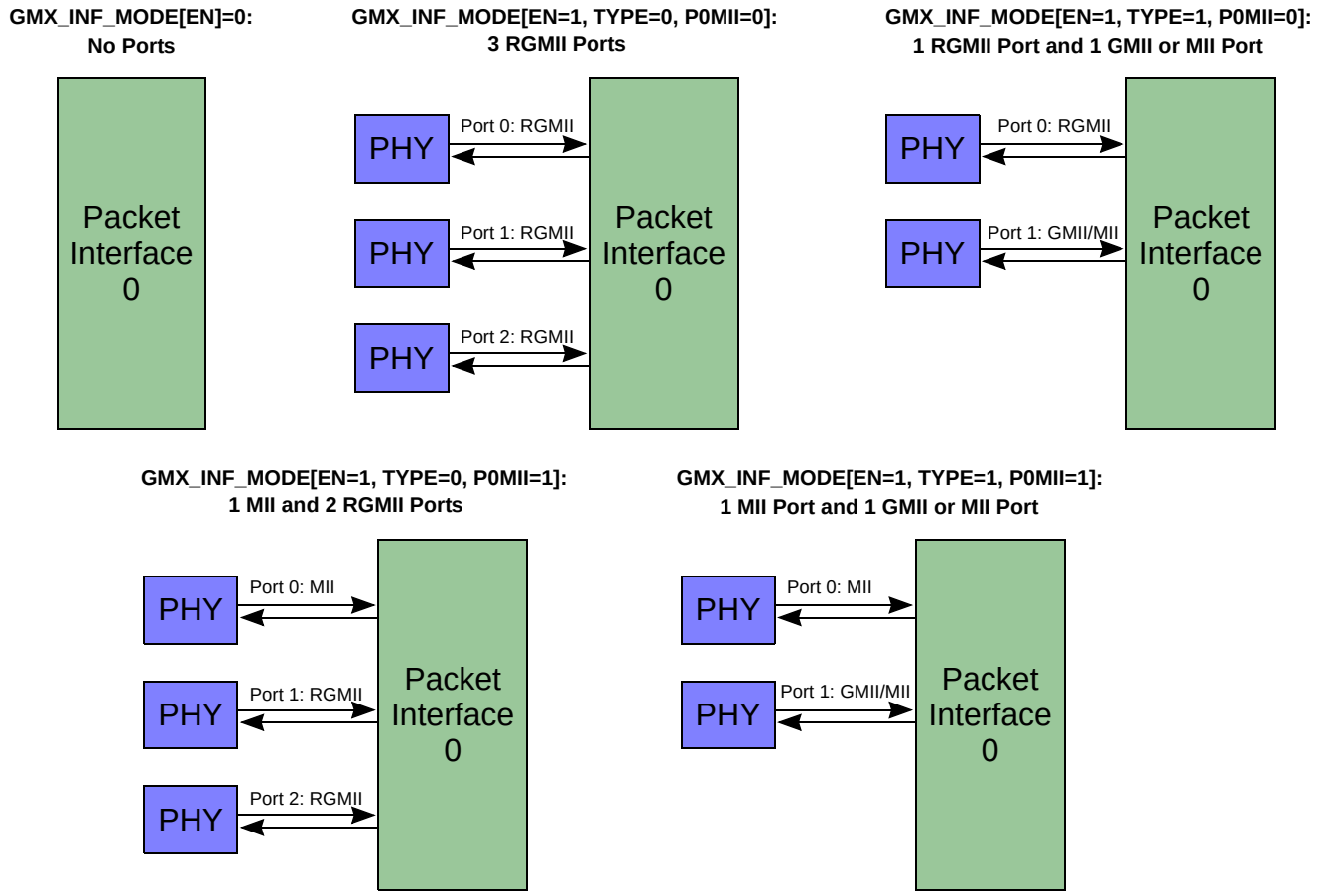


Figure 13–2 Packet Interface Operating Modes

Each enabled RGMII link can operate in 10Mbs, 100Mbs, or 1000Mbs speeds and in half- or full-duplex modes. The GMII link, if present, must be configured in 1000Mbs. If 10/100 Mbs is required, the link operates in MII mode. The GMII/MII link can be switched dynamically between modes when the media changes by using GMX_TX_CLK_MSK.

The packet interface has a 3KB receive buffer and a 3KB transmit staging buffer. The receive buffer can be partitioned among the physically allocated links based on GMX_RX_PRTS. The transmit buffer is fixed at 1KB per link.

In the output case, PKO has much buffering to add to the 3KB transmit staging buffer. More importantly, the CN50XX hardware buffers packet data in L2/DRAM, and this buffering can be many times larger than the RGMII transmit staging buffer and the PKO buffering combined.

In the input case, it is possible for PIP/IPD to exhaust all of the available L2/DRAM buffering, leaving the RGMII 4KB input buffer as the only buffering option for input packets. But most CN50XX applications use per-port backpressure and/or packet drop to prevent buffer exhaustion (refer to Sections 7.6 and 7.7). The CN50XX can backpressure (via PAUSE packets in full-duplex mode or forced collisions in half-duplex mode) to avoid overflow of the 4KB input buffer when its usage exceeds programmable on/off thresholds.

The following sections discuss the general operation of the CN50XX packet interface, regardless if the link is configured as RGMII or GMII.

13.2 RGMII Features

This section discusses some notable features provided by the CN50XX architecture outside the IEEE 802.3 and HP RGMII 1.3 specifications.

13.2.1 Flow Control

The following subsections discuss how the CN50XX receives and transmits flow control.

13.2.1.1 Receive Flow Control

The CN50XX supports flow control for both half- and full-duplex modes of operation for RGMII links only. GMII links must operate in full-duplex mode.

When the receive interface in full-duplex mode, the CN50XX can be configured to receive PAUSE control packets as defined in ANNEX 31A of the 802.3 specification based on the GMX0_RX*n*_FRM_CTL[CTL_BCK/CTL_DRP] fields. Common modes of operation are:

CTL_BCK = 1, CTL_DRP = 1	CN50XX drops pause packets, inspects the packets, and sets the transmit defer counter to the TIME parameter in the packets. CN50XX then defers the transmission of new packets until the timer expires.
CTL_BCK = 0, CTL_DRP = 0	CN50XX processes pause packets like normal packets, so software will see them. Software can optionally implement a backpressure scheme. Software can also set/reset the defer counter by writing to GMX0_TX*n*_SOFT_PAUSE[TIME]. Like pause frames, software must periodically write to GMX0_TX*n*_SOFT_PAUSE[TIME] if the flow condition extends beyond a TIME interval.
CTL_BCK = 0, CTL_DRP = 1	All pause frames are dropped and completely ignored by both hardware and software.
CTL_BCK = 1, CTL_DRP = 0	CN50XX sends pause packets to software, and sets the transmit-defer counter to the TIME parameter in the packet. CN50XX then defers the transmission of new outbound packets until the timer expires. As software can see all pause frames, it can optionally implement a backpressure scheme. Software can also set/reset the defer counter by writing to GMX0_TX*n*_SOFT_PAUSE[TIME]. Like pause frames, software must periodically write to GMX0_TX*n*_SOFT_PAUSE[TIME] if the flow condition extends beyond a TIME interval.

Note that the transmit defer counter is not set if FCS checking is enabled (i.e. GMX0_RX0_FRM_CHK[FCSERR] = 1) and the pause packet failed FCS/CRC check.

Both CTL_BCK and CTL_DRP should be set to 0s in half-duplex mode (pause packets should not be present in half-duplex mode).

13.2.1.2 Transmit Flow Control

The CN50XX controls the flow of packets from the transmitting device under the conditions specified in Table 13–2.

Table 13–2 Flow Control for Transmitting Device

Condition	Description
Receive FIFO Flow Control	The receive FIFO can control the flow as it fills up. Each port has a dedicated receive FIFO with dedicated controls. Figure 13–3 shows the relative FIFO depths for each programmable parameter. CN50XX begins to flow control to port when the FIFO accumulates GMX0_RX_BP_ON*n* data. Flow control is removed once the FIFO drops below GMX0_RX_BP_OFF*n* data.
Per-Port Flow Control	PIP/IPD can also communicate per-port flow control. This flow control is exhibited the same way as receive FIFO flow control. Refer to Sections 7.6 and 7.7 for more detailed information.
Software Flow Control	Software can use GMX0_TX_OVR_BP[EN/BP] to force or ignore the flow-control state as desired. If GMX0_TX_OVR_BP[EN*n*] = 1 for a port, the flow-control state will be GMX0_TX_OVR_BP[BP*n*]. This feature is intended for use only in system testing.

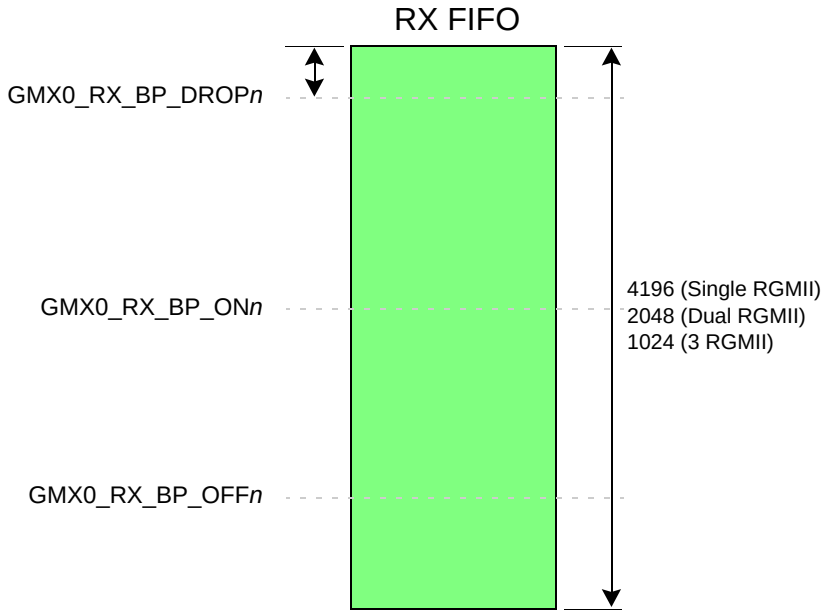


Figure 13–3 Receive FIFO Parameters

- In half-duplex mode when flow control is active for the port, CN50XX forces collisions by sending JAM packets.
- In full-duplex mode, CN50XX can generate PAUSE packets.
  - Each link can configure the PAUSE packet's SMAC (GMX0_SMAC*n*) and PAUSE TIME (GMX0_TX*n*_PAUSE_PKT_TIME).
  - Each link can configure how often the PAUSE packets are sent (GMX0_TX*n*_PAUSE_PKT_INTERVAL) under the flow control condition.

  CN50XX also supports a programmable DMAC (MX0_TX_PAUSE_PKT_DMAC) and TYPE (GMX0_TX_PAUSE_PKT_TYPE) shared by all RGMII links on an interface.

13.2.2 Receive Preamble

CN50XX can be configured to check or ignore PREAMBLE/SFD data that marks the beginning of each frame based on GMX0_RX*n*_FRM_CTL[PRE_CHK]. This may be useful for system configuration using multiple CN50XXs or other devices that choose to communicate over directly over RGMII and do not require a PHY device. If PREAMBLE data is present, it can be stripped or forwarded as part of the packet based on GMX0_RX*n*_FRM_CTL[PRE_STRP].

13.2.3 Receive Packet Dropping

There are several mechanisms within CN50XX that will drop packets:

- preamble errors
- receive-FIFO packet dropping
- PAUSE packet drops
- DMAC filters
- PIP/IPD per-QOS admission control
- Receive collisions

These mechanisms are described in the following subsections.

13.2.3.1 PREAMBLE Errors

If preamble checking is enabled (GMX0_RX*n*_FRM_CTL[PRE_CHK] = 1) and the packet does not send a valid PREAMBLE followed by SFD, the packet and all subsequent data until the next RGMII idle cycle is dropped. Normally only a single packet is dropped, but if the error occurs at the beginning of a burst, then multiple packets are dropped.

13.2.3.2 Receive-FIFO Packet Dropping

When the receive FIFO exceeds the GMX0_RX_BP_DROP*n* (as shown in Figure 13–3), the packet cannot be buffered and must be dropped. If there is room in the FIFO for a partial portion of the packet, the beginning of the packet is buffered but is sent down the IOBI bus with the PARTIAL opcode (indicating a drop). As described is Figure 7–9, partial packets can be recognized by software when WORD2[RE] = 1 and WORD2[OPCODE] = 1 (partial error). All other packets are completely dropped and are not sent on the IOBI bus.

Both partial drops and complete drops set the CIU_INT*n*_SUM0[GMX_DRP*m*] bit in CIU and can raise an interrupt based on CIU_INT*n*_EN0[GMX_DRP*m*].

13.2.3.3 PAUSE Packet Drops

OCTEON drops any incoming pause packet, when GMX0_RX*n*_FRM_CTL[CTL_DRP] = 1. See Section 13.2.1 for additional details.

13.2.3.4 DMAC Filter

Each RGMII link can store up to eight full MAC addresses to match against (GMX0_RX*n*_ADR_CAM_EN and GMX_RX_ADR_CAM*n*). CN50XX can be configured to accept or drop packets based on this address match (GMX0_RX*n*_ADR_CTL[CAM_MODE]). In addition, CN50XX can be configured to accept or reject multicast or broadcast packets (GMX0_RX*n*_ADR_CTL[MCST/BCST]).

The full algorithm is shown below:

```
bool dmac_addr_filter(uint8 prt, uint48 dmac) {
   ASSERT(prt >= 0 && prt <= 3);
   if (is_bcst(dmac))                                          //broadcast accept
      return (GMX_RX{prt}_ADR_CTL[BCST] ? ACCEPT : REJECT);
   if (is_mcst(dmac) & GMX_RX{prt}_ADR_CTL[MCST] == 1)   //multicast reject
      return REJECT;
   if (is_mcst(dmac) & GMX_RX{prt}_ADR_CTL[MCST] == 2)   //multicast accept
      return ACCEPT;

   cam_en  = 0;
   cam_hit = 0;

   for (i=0; i<8; i++) {
      if (GMX_RX{prt}_ADR_CAM_EN[EN<i>] == 0)
         continue;
      cam_en++;
      uint48 unswizzled_mac_adr = 0x0;
      for (j=5; j>=0; j--) {
         cam_byte = GMX_RX{prt}_ADR_CAM{j}[ADR<i*8+7:i*8>];
         unswizzled_mac_adr = (unswizzled_mac_adr << 8) | cam_byte;
      }
      if (unswizzled_mac_adr == dmac) {
         cam_hit = 1;
         break;
      }
   }

   if (cam_hit)
      return (GMX_RX{prt}_ADR_CTL[CAM_MODE] ? ACCEPT : REJECT);
   else
      return (GMX_RX{prt}_ADR_CTL[CAM_MODE] ? REJECT : ACCEPT);
}
```

13.2.3.5 PIP/IPD Per-QOS Admission Control

Section 7.7 discusses QOS admission. Packets that fail the admission test are dropped.

13.2.3.6 Receive Collisions

When operating in half-duplex mode (GMX0/1_PRT(0..3)_CFG[DUPLEX] = 0), the receiver drops any packets in which the slottime was not satisified. Note that late collisions are not normally dropped since they satisfiy the slottime. Late collisions can be dropped due to any of the other reasons above.

13.2.4 Receive-Packet Inspection

The following subsections describe the receive packets.

13.2.4.1 Receive-Packet Formats

Figure 13–4 shows the supported receive-packet formats, which differ from those described in Section 7.2. The packet interface supports all six of the L2 header types as defined in Figure 7–4.

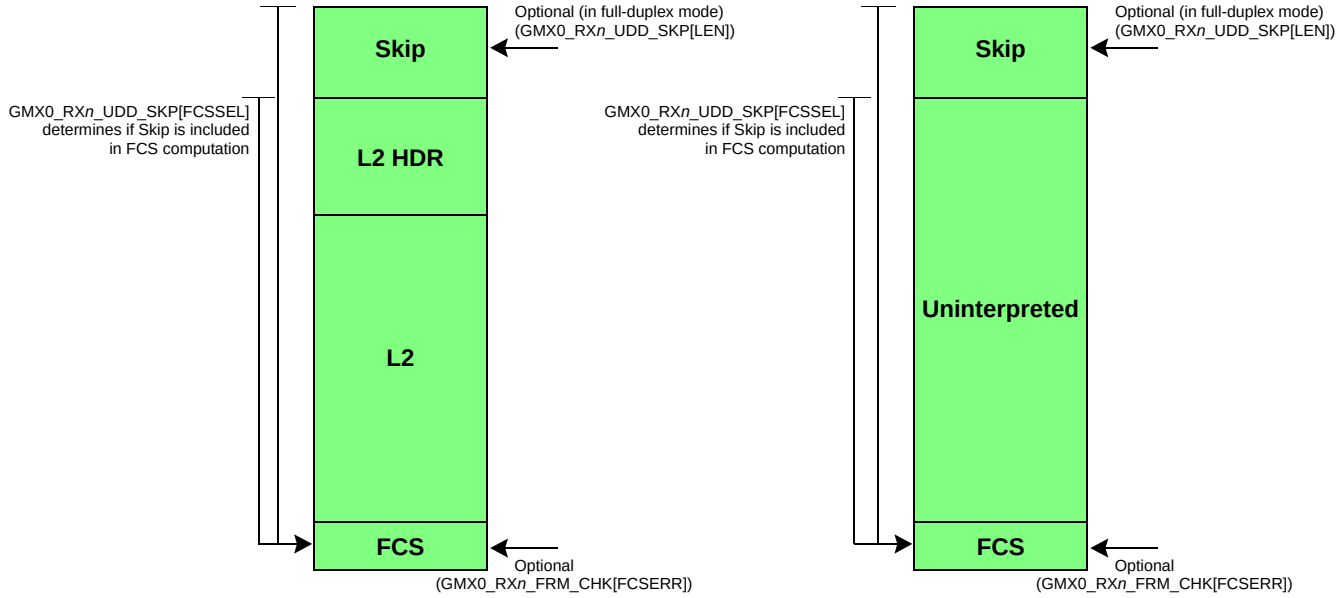


Figure 13–4 Packet Formats

The RGMII receiver inspects the packet to extract the L2 parameters that are required for various tasks and checks. The packet parser must be able to determine where the L2 header is in the packet to extract correct data.

13.2.4.2 Receive User-Defined Data

Normally, the L2 header will be either be at the start of the packet or after the optional preamble/SFD bytes. In full-duplex mode only, CN50XX supports user-defined data to come between the optional preamble/SFD and the start of the L2 header. CN50XX can support 0-64 bytes of UDD data based on GMX0_RX*n*_UDD_SKP[LEN]. Note that this is a static per-port configuration that cannot change on a per-packet basis. CN50XX can also be configured to compute the FCS over the UDD data or just the L2 header and L2 payload based on GMX0_RX*n*_UDD_SKP[FCSSEL].

Systems have two choices when skip bytes are sent over RGMII channels (followed by an ordinary ethernet L2 header):

1. If the skip amount is constant for all packets, set GMX0_RX*n*_UDD_SKP[LEN] to the total SKIP (Skip I + PKT_INST_HDR + Skip II, refer to Section 7.2.)
2. If the amount of total skip data before the L2 HDR is variable, the L2 HDR inspection must be disabled (see Section 13.2.4.4 for more details).

13.2.4.3 Receive Packets without L2 Headers

Some systems may not use the standard ethernet L2 HDR header or may develop custom headers. Normal L2 HDR inspection can be disabled on a per-port basis. If the L2 HDR is not present in the packet, the L2 HDR inspection must be disabled. See Section 13.2.4.4 for more details.

13.2.4.4 Disabling L2 Header Inspection

If the L2 header is not present or is not at a fixed offset in the packet, L2 header inspection should be disabled as follows:

– **DMAC filtering** – the L2 DMAC field cannot be extracted from the packet. The DMAC filter can be put into promiscuous mode by setting the following parameters:

GMX0_RX*n*_ADR_CTL[CAM_MODE] = 0

GMX0_RX*n*_ADR_CTL[MCST] = 0

GMX0_RX*n*_ADR_CTL[BCST] = 1

GMX0_RX*n*_ADR_CAM_EN[EN] = 0.

– **Pause-packet recognition** – the L2 DMAC and type fields cannot be extracted from the packet. Pause-packet recognition can be disabled by setting the following parameters:

GMX0_RX*n*_FRM_CTL[CTL_BCK] = 0

GMX0_RX*n*_FRM_CTL[CTL_DRP] = 0

13.2.5 Receive Link Status

The RGMII protocol allows the RGMII link status to be on the data lines during IDLE or NOP RGMII cycles. The PHY can communicate the link status, speed, and duplex value. The current status can be read by software through the GMX0_RX*n*_INBND register and the hardware can be configured to interrupt upon changes.CN50XX

13.2.6 Packet Transmission

The transmit FIFO is 3KB and is partitioned into logical FIFOs, one per output port. Each logical FIFO is fixed a 1KB.

The RGMII transmit state machine begins packet transmission when the FIFO has either an EOP or the FIFO length exceeds a programmed number of bytes (GMX0_TX*n*_THRESH). The threshold value should be large enough to avoid underflow and small enough to avoid excessive latencies. The recommended value for this register is the reset value of the register that works across the widest number of configurations. In half-duplex mode, if the packet suffers a collision, it is automatically retried GMX0_TX_COL_ATTEMPT times. If the packet still collides, CN50XX discards the packet.

13.2.7 Transmit-Packet Options

The transmit machine can prepend/append various parts of a standard ethernet frame, as shown in Table 13–3.

Table 13–3 Transmit-Packet Options

Field	Control Register[Field]	Comment
PREAMBLE/SFD	GMX0_TX*n*_APPEND[PREAMBLE]	Prepend the standard PREAMBLE and SFD at the start of each packet.
PAD	GMX0_TX*n*_APPEND[PAD]/ GMX0_TX*n*_MIN_PKT[MIN_SIZE]	Append 0s in order to pad out a packet to the minimum packet size.
FCS	GMX0_TX*n*_APPEND[FCS]	Compute and append the standard FCS at the end of the packet.

13.2.8 Collisions

CN50XX fully supports half-duplex CDMA/CS operation in 10/100/1000Mbs modes. In the event of a detected collision, the CN50XX takes the following steps:

- the RGMII transmit engine bursts the JAM pattern, backs off the bus, and retransmits the packet later.
- CN50XX carrier extends to meet the higher slottime of 1000Mbs operation.
- CN50XX attempts retransmission GMX0_TX_COL_ATTEMPT[LIMIT] times before giving up. If the retransmission limit is reached, it is considered an excessive collision (refer to XSCOL in Table 13–5). The packet is drained and the transmit machines resets the collision count for the next packet.

Late collisions (i.e. collisions that occur after the slottime is satisfied) are considered an error and the packet is not retransmitted.

13.2.9 Bursts

CN50XX supports packet bursting on half-duplex links in 1000Mbs mode. The RGMII transmit continues sending packets (if packets are available to send) up until the burst limit (GMX0_TX*n*_BURST[BURST]). The IFG cycles are filled with carrier extends.

13.3 Errors/Exceptions

CN50XX implements several error and exception checks on the RGMII interfaces. The following subsections describe the exception checkers and how the exception is communicated to software.

13.3.1 Receive Error/Exception Checks

Exceptions can be enabled by setting the appropriate bit in GMX0_RX*n*_FRM_CHK. If the bit is set, the exception is logged in GMX0_RX*n*_INT_REG and can optionally raise an interrupt based on the corresponding bits in GMX0_RX*n*_INT_EN. In addition, packets sent into CN50XX arrive with a receive error (WQE WORD2[RE] = 1) and the opcode is set the exception.

ASX0_INT_REG contains internal checks that should never assert in normal operation. Exceptions in ASX0_INT_REG can optionally raise an interrupt based on the corresponding bits in ASX0_INT_EN.

The exceptions, their causes, and how they are handled are shown in Table 13–4.

Table 13–4 Receive Errors/Exceptions

Exception	Cause	Notification
CAREXT	A packet was received with one or more carrier extend errors.	GMX0_RX*n*_FRM_CHK[CAREXT] enables the check. If enabled, GMX0_RX*n*_INT_REG[CAREXT] is set to 1 and WQE WORD2[OPCODE] is set to 0x9.
JABBER	A packet was received with length > GMX0_RX*n*_JABBER bytes.	The packet is unconditionally truncated at GMX0_RX*n*_JABBER bytes. GMX0_RX*n*_FRM_CHK[JABBER] enables the check. If enabled, GMX0_RX*n*_INT_REG[JABBER] is set to 1 and WQE WORD2[OPCODE] is set to 0x2.
FCSERR	A packet was received that failed the FCS/CRC check.	GMX0_RX*n*_FRM_CHK[FCSERR] enables the check. If enabled, GMX0_RX*n*_INT_REG[FCSERR] is set to 1 and WQE WORD2[OPCODE] is set to 0x3, 0x5, 0x6, or 0x7 depending on the length and alignment checks.
ALNERR	A packet was received that fails the FCS check (if enabled) and was not an integer number of bytes. Only applies to 10/100Mbs mode. If the FCS check is disabled, all packets that are not an integer number of bytes are considered ALNERRs.	GMX0_RX*n*_FRM_CHK[ALNERR] enables the alignment check and GMX0_RX*n*_FRM_CHK[FCSERR] enables the FCS check. If enabled, GMX0_RX*n*_INT_REG[ALNERR] is set to 1 and WQE WORD2[OPCODE] is set to 0x5.
RCVERR	A packet was received with one more cycles of data-reception error indicated on the RGMII bus.	GMX0_RX*n*_FRM_CHK[RCVERR] enables the check. If enabled, GMX0_RX*n*_INT_REG[RCVERR] is set to 1 and WQE WORD2[OPCODE] is set to 0xB.
SKPERR	A packet was received in which the length < GMX0_RX*n*_UDD_SKP[LEN] bytes. This means that there was not enough packet data in order to get passed the Skip field and into the L2 HDR.	GMX0_RX*n*_FRM_CHK[SKPERR] enables the check. If enabled, GMX0_RX*n*_INT_REG[SKPERR] is set to 1 and WQE WORD2[OPCODE] is set to 0xC.
NIBERR	A packet received had a stutter error (data not repeated on both RGMII clock edges. 10/100Mbs only).	GMX0_RX*n*_FRM_CHK[NIBERR] enables the check. If enabled, GMX0_RX*n*_INT_REG[NIBERR] is set to 1 and WQE WORD2[OPCODE] is set to 0xD.

Table 13–4 Receive Errors/Exceptions (Continued)

Exception	Cause	Notification
OVRERR	Indicates that RGMII/GMII data arrived too quickly. This can only occur when the RGMII RXC or the GMII RXCLK and the CN50XX core-clock frequencies are mismatched. core-clock rate ≥ 2 × RXC. This should never happen during normal operation.	This check is always on and sets GMX0_RX*n*_INT_REG[OVRERR] to 1.
PCTERR	A packet with a bad preamble was received.	GMX0_RX*n*_FRM_CTL[PRE_CHK/PRE_FREE] enables the preamble checker. If enabled, GMX0_RX*n*_INT_REG[PCTERR] is set to 1 and the packet is dropped.
RSVERR	CN50XX detected a reserved opcode on the RGMII interface.	This check is always on and sets GMX0_RX*n*_INT_REG[RSVERR] to 1.
FALERR	CN50XX detected a false carrier on the RGMII interface.	This check is always on and sets GMX0_RX*n*_INT_REG[FALERR] to 1.
COLDET	In half-duplex mode, CN50XX detected a collision. This can and will occur under normal operation in half-duplex mode.	This check is always on and sets GMX0_RX*n*_INT_REG[COLDET] to 1.
IFGERR	CN50XX detected an interframe gap violation. This exception does not necessarily indicate a system failure.	This check is always on and sets GMX0_RX*n*_INT_REG[IFGERR] to 1.
OVRFLW	Receive FIFO overflow occurred.	This check is always on sets ASX0_INT_EN[OVRFLW*n*] to 1.

13.3.2 Transmit Error/Exception Checks

Transmit exceptions are logged in two registers.

- GMX0_TX*n*_INT_REG
- ASX0_INT_REG

GMX0_TX*n*_INT_REG contains the most common exceptions and can optionally raise an interrupt based on the corresponding bits in GMX0_TX*n*_INT_EN.

ASX0_INT_REG contains internal checks that should never assert in normal operation. Exceptions in ASX0_INT_REG can optionally raise an interrupt based on the corresponding bits in ASX0_INT_EN.

The exceptions, their causes, and how they are handled are shown in Table 13–5.

Table 13–5 Transmit Errors/Exceptions

Exception	Cause	Notification
PKO_NXA	The RGMII transmitter received a request to send data out a port not enabled by GMX0_TX_PRTS[PRTS].	CN50XX asserts GMX0_TX*n*_INT_REG[PKO_NXA*n*] and logs the offending port in GMX0_NXA_ADR.
UNDFLW	In the unlikely event that PKO cannot keep the RGMII TX FIFO full, the RGMII packet transfer will underflow. This should be detected by the receiving device as an FCS error. Internally, the packet is drained and lost.	CN50XX asserts GMX0_TX*n*_INT_REG[UNDFLW*n*].
XSCOL	An excessive collision occurs in half-duplex mode if CN50XX detects a collision during each of GMX0_TX*n*_COL_ATTEMPT[LIMIT] attempts to send a packet.	CN50XX asserts GMX0_TX*n*_INT_REG[XSCOL*n*] and the packet is drained and lost.
XSDEF	An excessive deferral occurs in half-duplex mode if CN50XX detects that a packet cannot send on the transmit interface for maxDeferTime (as defined by IEEE 802.3-2002 specification) time.	CN50XX asserts GMX0_TX*n*_INT_REG[XSDEF*n*].
OUT_COL NCB_OVR OUT_OVR LOSTSTAT STATOVR INB_NXA	The transmitter has several internal error checks. These checks never assert in a correctly configured system. Any assertion indicates an internal problem has occurred.	CN50XX asserts a bit in GMX0_BAD_REG.
TXPSH	Internal overflow check on the transmit FIFO. This cannot occur normal operation.	CN50XX asserts ASX0_INT_REG[TXPSH*n*].
TXPOP	Internal underflow check on the transmit FIFO. This cannot occur normal operation.	CN50XX asserts ASX0_INT_REG[TXPOP*n*].

13.3.3 Transmit Error Propagation

If PKO detects an error when reading packet data, the error is passed to the RMGII transmitter. If GMX0_TX_CORRUPT[CORRUPT*n*] is set, CN50XX corrupts the FCS of the packet in order to notify the receiving device that the packet is corrupted.

Software can detect that PKO detected an error through the PKO_REG_ERROR[PARITY] bit.

13.4 Link

Software must be able to detect the link status and mode and set the GMX configuration registers accordingly.

13.4.1 Link Status

Software can detect the link status and mode of operation by either reading the appropriate registers in the PHY using the SMI interface (see Chapter 18). If the link is operating in RGMII mode and PHY supports optional inband status, then software can use the GMX0_RX*n*_INBND[STATUS,SPEED,DUPLEX] fields. Note, inband status is not available when the link is operating in GMII mode and the GMX0_RX*n*_INBND should not be used.

13.4.2 Link Status Changes

Software can detect a change in the link status by several methods.

- Software can poll the appropriate registers in the PHY looking for a status change.
- If the PHY supports link-status-change interrupts, the board can route the PHY interrupt to CN50XX.
- If the link is in RGMII mode and the PHY supports the optional RGMII in-band link status, the current link status is reported through the GMX0/1_RX*n*_INBND[STATUS,SPEED,DUPLEX] registers. If this feature is supported by the PHY, software can do either of the following:
  - poll GMX0/1_RX*n*_INT_REG[PHY_DUPX/PHY_SPD/PHY_LINK]
  - set GMX0/1_RX*n*_INT_EN[PHY_DUPX/PHY_SPD/PHY_LINK] and wait for the interrupt.

When a link-status change is detected, software can only update the GMX configuration CSRs as follows:

- Software detects a change to the RGMII/GMII/MII operating parameters (duplex, speed, link status).
- Software clears GMX_PRT_CFG[EN].
- Software should read back GMX_PRT_CFG in order to flush the write operation. This causes GMX to finish working on any in-flight packet in both the RX and TX directions.

- Software should then wait a **max_packet_time** before changing parameters, where **max_packet_time** can be defined as:

$$\text{max\_packet\_time } (\mu s) = \frac{\text{max\_packet\_size (bytes)} \times 8}{\text{port\_speed (Mbps)}}$$

- Software can reprogram any appropriate CSRs (including all fields in GMX_PRT_CFG except the [EN] field).
- Software reenables the port by setting GMX_PRT_CFG[EN]

13.4.3 Configuration Based on Mode

GMX has several configuration registers that should be set depending on the link status. The following table lists the recommended settings for each mode.

Table 13–6 Recommended Configuration Settings

CSR/Field	RGMII			GMII	MII	
	1000	100	10	1000	100	10
GMX0_PRT*n*_CFG[SPEED]	1	0	0	1	0	0
GMX0_PRT*n*_CFG[SLOTTIME]	1	0	0	1	0	0
GMX0_TX*n*_CLK[CLK_CNT]	1	5	50	1	1	1
GMX0_TX*n*_SLOT[SLOT]	512	64	64	512	64	64
GMX0_TX*n*_BURST (half-duplex only)	8192	0	0	8192	0	0

13.5 Statistics

CN50XX supports a rich statistics gathering, described in Table 13–7, in order to support RX RMON and TX RMON. Refer to the CSR descriptions in Section 13.8 for a complete list of statistics.

Table 13–7 CN50XX Statistics Gathering

CSR Field	Statistic
RX	
GMX0_RX*n*_STATS_PKTS[CNT]	32-bit count of all good packets
GMX0_RX*n*_STATS_OCTS[CNT]	48-bit count of all bytes from good packets
GMX0_RX*n*_STATS_PKTS_CTL[CNT]	32-bit count of all control/PAUSE packets
GMX0_RX*n*_STATS_OCTS_CTL[CNT]	48-bit count of all bytes from control/PAUSE packets
GMX0_RX*n*_STATS_PKTS_DMAC[CNT]	32-bit count of all DMAC filtered packets
GMX0_RX*n*_STATS_OCTS_DMAC[CNT	48-bit count of all bytes from DMAC filtered packets
GMX0_RX*n*_STATS_PKTS_DRP[CNT]	32-bit count of all packets dropped due to RX FIFO full
GMX0_RX*n*_STATS_OCTS_DRP[CNT]	48-bit count of all bytes from packets dropped due to RX FIFO full
GMX0_RX*n*_STATS_PKTS_BAD[CNT]	32-bit count of all bad packets
TX	
GMX0_TX*n*_STAT0[XSCOL]	32-bit count of packets dropped due to excessive collisions
GMX0_TX*n*_STAT0[XSDEF]	32-bit count of packets dropped due to excessive deferral
GMX0_TX*n*_STAT1[MCOL]	32-bit count of packets sent that experienced multiple collisions before successful transmission
GMX0_TX*n*_STAT1[SCOL]	32-bit count of packets sent that experienced a single collision before successful transmission

Table 13–7 CN50XX Statistics Gathering

CSR Field	Statistic
RX	
GMX0_TX*n*_STAT2[OCTS]	48-bit count of all bytes sent
GMX0_TX*n*_STAT3[PKTS]	32-bit count of all packets sent
GMX0_TX*n*_STAT4[HIST0]	32-bit count of packets sent with an octet count < 64
GMX0_TX*n*_STAT4[HIST1]	32-bit count of packets sent with an octet count == 64
GMX0_TX*n*_STAT5[HIST2]	32-bit count of packets sent with an octet count of 65-127
GMX0_TX*n*_STAT5[HIST3]	32-bit count of packets sent with an octet count of 128-255
GMX0_TX*n*_STAT6[HIST4]	32-bit count of packets sent with an octet count of 256-511
GMX0_TX*n*_STAT6[HIST5]	32-bit count of packets sent with an octet count of 512-1023
GMX0_TX*n*_STAT7[HIST6]	32-bit count of packets sent with an octet count of 1024-1518
GMX0_TX*n*_STAT7[HIST7]	32-bit count of packets sent with an octet count of > 1518
GMX0_TX*n*_STAT8[BCST]	32-bit count of packets sent to a multicast DMAC
GMX0_TX*n*_STAT8[MCST]	32-bit count of packets sent to the broadcast DMAC
GMX0_TX*n*_STAT9[CTL]	32-bit count of control/PAUSE packets sent
GMX0_TX*n*_STAT9[UNDFLW]	32-bit count of packets sent that experienced a transmit underflow and were truncated

13.6 Loopback

CN50XX supports both internal and external loopback paths to aid in system development and bringup.

- When ASX0_PRT_LOOP[EXP_LOOP*n*] is set to 1, the receive datapath is fed back into the transmit FIFO as shown in Figure 13–5. This allows CN50XX to reflect all received packets without involving the core.
- When ASX0_PRT_LOOP[INT_LOOP*n*] is set to 1, the transmit FIFO output is fed back into the receive FIFO as shown if Figure 13–5. This allows the RGMII interface to send packets into the CN50XX core.

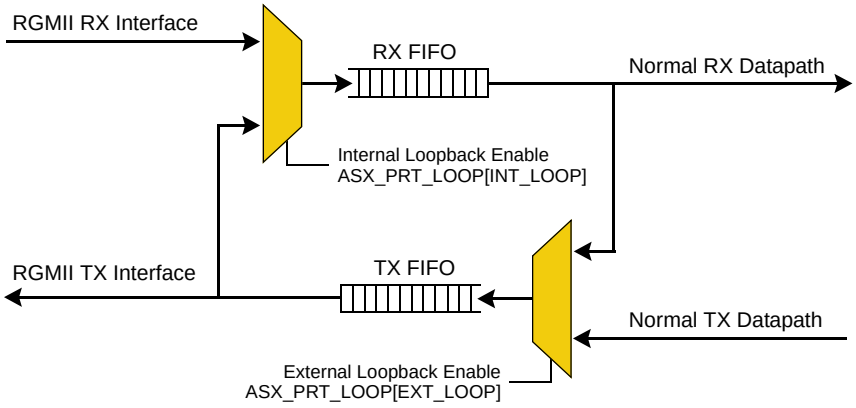


Figure 13–5 RGMII Loopback

13.7 Initialization

Perform the following steps in order to initialize the RGMII interface.

1. Configure the CN50XX core.
   The RGMII packet interface buffers packet data in main memory. Therefore, the FPA, IPD, PIP, PKO, and DDR controller (LMC) units must all be correctly configured to be able to store and send packet data.
2. Configure ASX registers.
   ASX registers will not normally to set. The power-on values were chosen to be compatible with most systems.
3. Configure GMX registers.
   GMX personality should be setup here, including the total number of ports used on each interface.
4. Determine the Link status.
   Link status can be determined by a number of methods, described in Section 13.4.
5. Configure the RGMII link based on link status.
   Section 13.4.3 provides guidelines for configuring the CSRs based on the link status.
6. Enable ASX.
   Set ASX0_RX_PRT_EN[PRT_EN] / ASX0_TX_PRT_EN[PRT_EN] for each port to be used.
7. Enable GMX.
   Set GMX0_PRT*n*_CFG[EN] for each port to be used.

Initialization is now complete. At this point, packets can be received on the receive channel and packets can be sent on the transmit channel.

13.8 GMX Registers

The GMX registers are listed in Table 13–8.

Table 13–8 GMX Registers

Register	Address	CSR Type[1]	Detailed Description
GMX0_RX0_INT_REG	0x0001180008000000	RSL	See page 530.
GMX0_RX0_INT_EN	0x0001180008000008	RSL	See page 531.
GMX0_PRT0_CFG	0x0001180008000010	RSL	See page 531.
GMX0_RX0_FRM_CTL	0x0001180008000018	RSL	See page 532.
GMX0_RX0_FRM_CHK	0x0001180008000020	RSL	See page 534.
GMX0_RX0_JABBER	0x0001180008000038	RSL	See page 534
GMX0_RX0_DECISION	0x0001180008000040	RSL	See page 535
GMX0_RX0_UDD_SKP	0x0001180008000048	RSL	See page 536
GMX0_RX0_STATS_CTL	0x0001180008000050	RSL	See page 536
GMX0_RX0_IFG	0x0001180008000058	RSL	See page 537
GMX0_RX0_RX_INBND	0x0001180008000060	RSL	See page 537
GMX0_RX0_PAUSE_DROP_TIME	0x0001180008000068	RSL	See page 537
GMX0_RX0_STATS_PKTS	0x0001180008000080	RSL	See page 538
GMX0_RX0_STATS_OCTS	0x0001180008000088	RSL	See page 538
GMX0_RX0_STATS_PKTS_CTL	0x0001180008000090	RSL	See page 538
GMX0_RX0_STATS_OCTS_CTL	0x0001180008000098	RSL	See page 539
GMX0_RX0_STATS_PKTS_DMAC	0x00011800080000A0	RSL	See page 539
GMX0_RX0_STATS_OCTS_DMAC	0x00011800080000A8	RSL	See page 539
GMX0_RX0_STATS_PKTS_DRP	0x00011800080000B0	RSL	See page 540
GMX0_RX0_STATS_OCTS_DRP	0x00011800080000B8	RSL	See page 540
GMX0_RX0_STATS_PKTS_BAD	0x00011800080000C0	RSL	See page 540
GMX0_RX0_ADR_CTL	0x0001180008000100	RSL	See page 541
GMX0_RX0_ADR_CAM_EN	0x0001180008000108	RSL	See page 541
GMX0_RX0_ADR_CAM0 ... GMX0_RX0_ADR_CAM5	0x0001180008000180 ... 0x00011800080001A8	RSL	See page 542
GMX0_TX0_CLK	0x0001180008000208	RSL	See page 542
GMX0_TX0_THRESH	0x0001180008000210	RSL	See page 542
GMX0_TX0_APPEND	0x0001180008000218	RSL	See page 543
GMX0_TX0_SLOT	0x0001180008000220	RSL	See page 543
GMX0_TX0_BURST	0x0001180008000228	RSL	See page 543
GMX0_SMAC0	0x0001180008000230	RSL	See page 543
GMX0_TX0_PAUSE_PKT_TIME	0x0001180008000238	RSL	See page 544
GMX0_TX0_MIN_PKT	0x0001180008000240	RSL	See page 544
GMX0_TX0_PAUSE_PKT_INTERVAL	0x0001180008000248	RSL	See page 545
GMX0_TX0_SOFT_PAUSE	0x0001180008000250	RSL	See page 545
GMX0_TX0_PAUSE_TOGO	0x0001180008000258	RSL	See page 546
GMX0_TX0_PAUSE_ZERO	0x0001180008000260	RSL	See page 546
GMX0_TX0_STATS_CTL	0x0001180008000268	RSL	See page 546

Table 13–8 GMX Registers (Continued)

Register	Address	CSR Type[1]	Detailed Description
GMX0_TX0_CTL	0x0001180008000270	RSL	See page 546
GMX0_TX0_STAT0 ... GMX0_TX0_STAT9	0x0001180008000280 ... 0x00011800080002C8	RSL	See page 547
GMX0_BIST	0x0001180008000400	RSL	See page 550
GMX0_RX_PRTS	0x0001180008000410	RSL	See page 550
GMX0_RX_BP_DROP0 GMX0_RX_BP_DROP1 GMX0_RX_BP_DROP2	0x0001180008000420 0x0001180008000428 0x0001180008000430	RSL	See page 550
GMX0_RX_BP_ON0 GMX0_RX_BP_ON1 GMX0_RX_BP_ON2	0x0001180008000440 0x0001180008000448 0x0001180008000450	RSL	See page 551
GMX0_RX_BP_OFF0 GMX0_RX_BP_OFF1 GMX0_RX_BP_OFF2	0x0001180008000460 0x0001180008000468 0x0001180008000470	RSL	See page 552
GMX0_TX_PRTS	0x0001180008000480	RSL	See page 552
GMX0_TX_IFG	0x0001180008000488	RSL	See page 552
GMX0_TX_JAM	0x0001180008000490	RSL	See page 552
GMX0_TX_COL_ATTEMPT	0x0001180008000498	RSL	See page 553
GMX0_TX_PAUSE_PKT_DMAC	0x00011800080004A0	RSL	See page 553
GMX0_TX_PAUSE_PKT_TYPE	0x00011800080004A8	RSL	See page 553
GMX0_TX_OVR_BP	0x00011800080004C8	RSL	See page 553
GMX0_TX_BP	0x00011800080004D0	RSL	See page 554
GMX0_TX_CORRUPT	0x00011800080004D8	RSL	See page 554
GMX0_RX_PRT_INFO	0x00011800080004E8	RSL	See page 555
GMX0_TX_LFSR	0x00011800080004F8	RSL	See page 555
GMX0_TX_INT_REG	0x0001180008000500	RSL	See page 555
GMX0_TX_INT_EN	0x0001180008000508	RSL	See page 555
GMX0_NXA_ADR	0x0001180008000510	RSL	See page 556
GMX0_BAD_REG	0x0001180008000518	RSL	See page 556
GMX0_STAT_BP	0x0001180008000520	RSL	See page 556
GMX0_TX_CLK_MSK0 GMX0_TX_CLK_MSK1	0x0001180008000780 0x0001180008000788	RSL	See page 557
GMX0_RX_TX_STATUS	0x00011800080007E8	RSL	See page 557
GMX0_INF_MODE	0x00011800080007F8	RSL	See page 557
GMX0_RX1_INT_REG	0x0001180008000800	RSL	See page 530.
GMX0_RX1_INT_EN	0x0001180008000808	RSL	See page 531.
GMX0_PRT1_CFG	0x0001180008000810	RSL	See page 531.
GMX0_RX1_FRM_CTL	0x0001180008000818	RSL	See page 532.
GMX0_RX1_FRM_CHK	0x0001180008000820	RSL	See page 534.
GMX0_RX1_JABBER	0x0001180008000838	RSL	See page 534
GMX0_RX1_DECISION	0x0001180008000840	RSL	See page 535
GMX0_RX1_UDD_SKP	0x0001180008000848	RSL	See page 536
GMX0_RX1_STATS_CTL	0x0001180008000850	RSL	See page 536

Table 13–8 GMX Registers (Continued)

Register	Address	CSR Type[1]	Detailed Description
GMX0_RX1_IFG	0x0001180008000858	RSL	See page 537
GMX0_RX1_RX_INBND	0x0001180008000860	RSL	See page 537
GMX0_RX1_PAUSE_DROP_TIME	0x0001180008000868	RSL	See page 537
GMX0_RX1_STATS_PKTS	0x0001180008000880	RSL	See page 538
GMX0_RX1_STATS_OCTS	0x0001180008000888	RSL	See page 538
GMX0_RX1_STATS_PKTS_CTL	0x0001180008000890	RSL	See page 538
GMX0_RX1_STATS_OCTS_CTL	0x0001180008000898	RSL	See page 539
GMX0_RX1_STATS_PKTS_DMAC	0x00011800080008A0	RSL	See page 539
GMX0_RX1_STATS_OCTS_DMAC	0x00011800080008A8	RSL	See page 539
GMX0_RX1_STATS_PKTS_DRP	0x00011800080008B0	RSL	See page 540
GMX0_RX1_STATS_OCTS_DRP	0x00011800080008B8	RSL	See page 540
GMX0_RX1_STATS_PKTS_BAD	0x00011800080008C0	RSL	See page 540
GMX0_RX1_ADR_CTL	0x0001180008000900	RSL	See page 541
GMX0_RX1_ADR_CAM_EN	0x0001180008000908	RSL	See page 541
GMX0_RX1_ADR_CAM0 ... GMX0_RX1_ADR_CAM5	0x0001180008000980 ... 0x00011800080009A8	RSL	See page 542
GMX0_TX1_CLK	0x0001180008000A08	RSL	See page 542
GMX0_TX1_THRESH	0x0001180008000A10	RSL	See page 542
GMX0_TX1_APPEND	0x0001180008000A18	RSL	See page 543
GMX0_TX1_SLOT	0x0001180008000A20	RSL	See page 543
GMX0_TX1_BURST	0x0001180008000A28	RSL	See page 543
GMX0_SMAC1	0x0001180008000A30	RSL	See page 543
GMX0_TX1_PAUSE_PKT_TIME	0x0001180008000A38	RSL	See page 544
GMX0_TX1_MIN_PKT	0x0001180008000A40	RSL	See page 544
GMX0_TX1_PAUSE_PKT_INTERVAL	0x0001180008000A48	RSL	See page 545
GMX0_TX1_SOFT_PAUSE	0x0001180008000A50	RSL	See page 545
GMX0_TX1_PAUSE_TOGO	0x0001180008000A58	RSL	See page 546
GMX0_TX1_PAUSE_ZERO	0x0001180008000A60	RSL	See page 546
GMX0_TX1_STATS_CTL	0x0001180008000A68	RSL	See page 546
GMX0_TX1_CTL	0x0001180008000A70	RSL	See page 546
GMX0_TX1_STAT0 ... GMX0_TX1_STAT9	0x0001180008000A80 ... 0x0001180008000AC8	RSL	See page 547
GMX0_RX2_INT_REG	0x0001180008001000	RSL	See page 530.
GMX0_RX2_INT_EN	0x0001180008001008	RSL	See page 531.
GMX0_PRT2_CFG	0x0001180008001010	RSL	See page 531.
GMX0_RX2_FRM_CTL	0x0001180008001018	RSL	See page 532.
GMX0_RX2_FRM_CHK	0x0001180008001020	RSL	See page 534.
GMX0_RX2_JABBER	0x0001180008001038	RSL	See page 534
GMX0_RX2_DECISION	0x0001180008001040	RSL	See page 535
GMX0_RX2_UDD_SKP	0x0001180008001048	RSL	See page 536
GMX0_RX2_STATS_CTL	0x0001180008001050	RSL	See page 536

Table 13–8 GMX Registers (Continued)

Register	Address	CSR Type[1]	Detailed Description
GMX0_RX2_IFG	0x0001180008001058	RSL	See page 537
GMX0_RX2_RX_INBND	0x0001180008001060	RSL	See page 537
GMX0_RX2_PAUSE_DROP_TIME	0x0001180008001068	RSL	See page 537
GMX0_RX2_STATS_PKTS	0x0001180008001080	RSL	See page 538
GMX0_RX2_STATS_OCTS	0x0001180008001088	RSL	See page 538
GMX0_RX2_STATS_PKTS_CTL	0x0001180008001090	RSL	See page 538
GMX0_RX2_STATS_OCTS_CTL	0x0001180008001098	RSL	See page 539
GMX0_RX2_STATS_PKTS_DMAC	0x00011800080010A0	RSL	See page 539
GMX0_RX2_STATS_OCTS_DMAC	0x00011800080010A8	RSL	See page 539
GMX0_RX2_STATS_PKTS_DRP	0x00011800080010B0	RSL	See page 540
GMX0_RX2_STATS_OCTS_DRP	0x00011800080010B8	RSL	See page 540
GMX0_RX2_STATS_PKTS_BAD	0x00011800080010C0	RSL	See page 540
GMX0_RX2_ADR_CTL	0x0001180008001100	RSL	See page 541
GMX0_RX2_ADR_CAM_EN	0x0001180008001108	RSL	See page 541
GMX0_RX2_ADR_CAM0 ... GMX0_RX2_ADR_CAM5	0x0001180008001180 ... 0x00011800080011A8	RSL	See page 542
GMX0_TX2_CLK	0x0001180008001208	RSL	See page 542
GMX0_TX2_THRESH	0x0001180008001210	RSL	See page 542
GMX0_TX2_APPEND	0x0001180008001218	RSL	See page 543
GMX0_TX2_SLOT	0x0001180008001220	RSL	See page 543
GMX0_TX2_BURST	0x0001180008001228	RSL	See page 543
GMX0_SMAC2	0x0001180008001230	RSL	See page 543
GMX0_TX2_PAUSE_PKT_TIME	0x0001180008001238	RSL	See page 544
GMX0_TX2_MIN_PKT	0x0001180008001240	RSL	See page 544
GMX0_TX2_PAUSE_PKT_INTERVAL	0x0001180008001248	RSL	See page 545
GMX0_TX2_SOFT_PAUSE	0x0001180008001250	RSL	See page 545
GMX0_TX2_PAUSE_TOGO	0x0001180008001258	RSL	See page 546
GMX0_TX2_PAUSE_ZERO	0x0001180008001260	RSL	See page 546
GMX0_TX2_STATS_CTL	0x0001180008001268	RSL	See page 546
GMX0_TX2_CTL	0x0001180008001270	RSL	See page 546
GMX0_TX2_STAT0 ... GMX0_TX2_STAT9	0x0001180008001280 ... 0x00011800080012C8	RSL	See page 547

1. RSL-type registers are accessed indirectly across the I/O bus.

GMX Interrupt Registers
GMX0_RX(0..2)_INT_REG

This register allows interrupts to be sent to the control processor. If an error is noted in this register, and the corresponding bit in GMX0_RX(0..2)_INT_EN is set, an exception is raised. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:20>	—	RAZ	—	—	Reserved
<19>	PAUSE_DRP	R/W1C	0	0	Pause packet was dropped due to full GMX RX FIFO.
<18>	PHY_DUPX	R/W1C	0	0	Change in the RMGII inbound LinkDuplex
<17>	PHY_SPD	R/W1C	0	0	Change in the RMGII inbound LinkSpeed
<16>	PHY_LINK	R/W1C	0	0	Change in the RMGII inbound LinkStatus
<15>	IFGERR	R/W1C	0	0	Interframe gap violation
<14>	COLDET	R/W1C	0	0	Collision detection. Collisions can only occur in half-duplex mode. The receiver determines if a collision occurred when the current frame being received does not satisfy the slottime.
<13>	FALERR	R/W1C	0	0	False-carrier error, or carrier-extend error after slottime is satisfied.
<12>	RSVERR	R/W1C	0	0	RGMII reserved opcode error.
<11>	PCTERR	R/W1C	0	0	Bad preamble / protocol error. Checks that the frame transitions from PREAMBLE ⇒ SFD ⇒ DATA. Does not check the number of PREAMBLE cycles.
<10>	OVRERR[1]	R/W1C	0	0	Internal data aggregation overflow error. This interrupt should never assert.
<9>	NIBERR[1]	R/W1C	0	0	Nibble error (hi_nibble ≠ lo_nibble). This error is illegal at 1000Mbs speeds (GMX0_PRT(0..2)_CFG[SPEED] = 0) and never asserts.
<8>	SKPERR[1]	R/W1C	0	0	Skipper error.
<7>	RCVERR[1]	R/W1C	0	0	Data-reception error. Frame was received with RMGII data-reception error
<6>	LENERR[1]	R/W1C	0	0	Length error: frame was received with a length error. Length errors occur when the received packet does not match the length field. LENERR is only checked for packets between 64 and 1500 bytes. For untagged frames, the length must be an exact match. For tagged frames the length or length+4 must match.
<5>	ALNERR[1]	R/W1C	0	0	Alignment error: frame was received with an alignment error. Indicates that the packet received was not an integer number of bytes. If FCS checking is enabled, ALNERR only asserts if the FCS is bad. If FCS checking is disabled, ALNERR asserts in all non-integer frame cases
<4>	FCSERR[1]	R/W1C	0	0	FCS/CRC error. Frame was received with FCS/CRC error
<3>	JABBER[1]	R/W1C	0	0	System-length error: frame was received with length > sys_length. A JABBER error indicates that a packet was received that is longer than the maximum allowed packet as defined by the system. GMX truncates the packet at the JABBER count. Failure to do so could lead to system instability.
<2>	MAXERR[1]	R/W1C	0	0	Maximum-length error: frame was received with length > max_length. For untagged frames, the total frame DA+SA+TL+DATA+PAD+FCS > GMX0_RX(0..2)_FRM_MAX. For tagged frames, DA+SA+VLAN+TL+DATA+PAD+FCS > GMX0_RX(0..2)_FRM_MAX + 4.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<1>	CAREXT[1]	R/W1C	0	0	RGMII carrier-extend error.
<0>	MINERR[1]	R/W1C	0	0	Minimum-length error. Frame was received with length < min_length. Total frame, DA+SA+TL+DATA+PAD+FCS < GMX0_RX(0..2)_FRM_MIN.

1. Exception conditions <10:0> can also set the **rcv/opcode** in the received packet's work-queue entry. GMX0_RX(0..2)_FRM_CHK provides a bit mask for configuring which conditions set the error.

GMX Interrupt-Enable Registers
GMX0_RX(0..2)_INT_EN

This register provides the interrupt-enable bits for the corresponding error in GMX0_RX(0..2)_INT_REG. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:20>	—	RAZ	—	—	Reserved.
<19>	PAUSE_DRP	R/W	0	0	Pause packet was dropped due to full GMX RX FIFO.
<18>	PHY_DUPX	R/W	0	0	Change in the RMGII inbound LinkDuplex.
<17>	PHY_SPD	R/W	0	0	Change in the RMGII inbound LinkSpeed.
<16>	PHY_LINK	R/W	0	0	Change in the RMGII inbound LinkStatus.
<15>	IFGERR	R/W	0	0	Interframe-gap violation.
<14>	COLDET	R/W	0	0	Collision detection.
<13>	FALERR	R/W	0	0	False-carrier error, or carrier-extend error after slottime is satisfied.
<12>	RSVERR	R/W	0	0	RGMII reserved opcodes.
<11>	PCTERR	R/W	0	0	Bad preamble / protocol.
<10>	OVRERR	R/W	0	0	Internal data aggregation overflow error.
<9>	NIBERR	R/W	0	0	Nibble error (hi_nibble ≠ lo_nibble).
<8>	SKPERR	R/W	0	0	Skipper error.
<7>	RCVERR	R/W	0	0	Frame was received with RMGII data reception error.
<6>	LENERR	R/W	0	0	Frame was received with length error.
<5>	ALNERR	R/W	0	0	Frame was received with an alignment error.
<4>	FCSERR	R/W	0	0	Frame was received with FCS/CRC error.
<3>	JABBER	R/W	0	0	Frame was received with length > sys_length.
<2>	MAXERR	R/W	0	0	Frame was received with length > max_length.
<1>	CAREXT	R/W	0	0	RGMII carrier extend error.
<0>	MINERR	R/W	0	0	Frame was received with length < min_length.

GMX Port Configuration Registers
GMX0_PRT(0..2)_CFG

This register controls the configuration of the port. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	—	—	Reserved
<3>	SLOTTIME	R/W	1	—	Slot time for half-duplex operation 0 = 512 bitimes (10/100 Mbs operation) 1 = 4096 bitimes (1000 Mbs operation)
<2>	DUPLEX	R/W	1	—	Duplex mode: 0 = half-duplex (collisions/extensions/bursts), 1 = full-duplex

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<1>	SPEED	R/W	1	—	Link speed 0 = 10/100 Mbs operation in RGMII mode: GMX0_TX(0..2)_CLK[CLK_CNT] > 1 in MII mode: GMX0_TX(0..2)_CLK[CLK_CNT] = 1 1 = 1000 Mbs operation
<0>	EN	R/W	0	—	Link enable. When EN is clear, packets are not received or transmitted (including PAUSE and JAM packets). If EN is cleared while a packet is currently being received or transmitted, the packet is allowed to complete before the bus is idled. On the RX side, subsequent packets in a burst are ignored.

Frame Control Registers
GMX0_RX(0..2)_FRM_CTL

This register controls the handling of the frames. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:11>	—	RAZ	—	—	Reserved
<10>	NULL_DIS	R/W	0	0	Null tick disable. When set to 1, do not modify the MOD bits on NULL ticks that are due to partial packets.
<9>	PRE_ALIGN	R/W	1	1	Preamble parser align. When set to 1, the preamble parser aligns the the SFD byte regardless of the number of previous preamble nibbles. In this mode, PREAMBLE can be consumed by the hardware so when PRE_ALIGN is set, PRE_FREE, PRE_STRP must be set for correct operation. PRE_CHK must be set to enable this and all PREAMBLE features.
<8:7>	—	RAZ	—	—	Reserved
<6>	PRE_FREE	R/W	1	1	Allows for less strict preamble checking. 0–254 cycles of PREAMBLE followed by SFD.
<5>	CTL_SMAC	R/W	0	0	Control pause frames can match station SMAC.
<4>	CTL_MCST	R/W	1	1	Control pause frames can match globally assign multicast address.
<3>	CTL_BCK	R/W	1	1	Forward pause information to TX block.
<2>	CTL_DRP	R/W	1	1	Drop control-pause frames.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<1>	PRE_STRP	R/W	1	1	Strip off the preamble (when present). 0 = PREAMBLE + SFD is sent to core as part of frame 1 = PREAMBLE + SFD is dropped When PRE_STRP = 1 (indicating that the PREAMBLE will be sent), PRE_STRP determines if the PREAMBLE + SFD bytes are thrown away or sent to the core as part of the packet. In either mode, the PREAMBLE + SFD bytes are not counted toward the packet size when checking against the MIN and MAX bounds. Furthermore, the bytes are skipped when locating the start of the L2 header for DMAC and Control frame recognition
<0>	PRE_CHK	R/W	1	1	Check the preamble for correctness. This port is configured to send PREAMBLE + SFD to begin every frame. GMX checks that the PREAMBLE is sent correctly. When PRE_CHK = 1, the RGMII state expects a typical frame consisting of INTER_FRAME ⇒ PREAMBLE(×7) ⇒ SFD(×1) ⇒ DAT. The state machine watches for this exact sequence in order to recognize a valid frame and push frame data into the CN50XX. There must be exactly seven PREAMBLE cycles followed by the single SFD cycle for the frame to be accepted. When a problem does occur within the PREAMBLE sequence, the frame is marked as bad and not sent into the core. The GMX0_RX(0..2)_INT_REG[PCTERR] interrupt is also raised.

Notes:

- CTL_BCK/CTL_DRP

  These bits control how the hardware handles incoming PAUSE packets. The most common modes of operation:

  - CTL_BCK = 1, CTL_DRP = 1: hardware handles everything
  - CTL_BCK = 0, CTL_DRP = 0: software sees all pause frames
  - CTL_BCK = 0, CTL_DRP = 1: all pause frames are completely ignored

  These control bits should be set to CTL_BCK = 0,CTL_DRP = 0 in half-duplex mode. Since PAUSE packets only apply to fulldup operation, any PAUSE packet would constitute an exception which should be handled by the processing cores. PAUSE packets should not be forwarded.

Frame Check Registers
GMX0_RX(0..2)_FRM_CHK

This register indicates which frame errors will set the ERR bit of the frame. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:10>	—	RAZ	—	—	Reserved.
<9>	NIBERR	R/W	1	1	Nibble error (i.e. hi_nibble ≠ lo_nibble).
<8>	SKPERR	R/W	1	1	Skipper error.
<7>	RCVERR	R/W	1	1	Frame was received with RMGII data-reception error.
<6>	—	RAZ	—	—	Reserved.
<5>	ALNERR	R/W	1	1	Frame was received with an alignment error.
<4>	FCSERR	R/W	1	1	Frame was received with FCS/CRC error.
<3>	JABBER	R/W	1	1	Frame was received with length > sys_length.
<2>	—	RAZ	—	—	Reserved.
<1>	CAREXT	R/W	1	1	RGMII carrier extend error.
<0>	—	RAZ	—	—	Reserved.

GMX Maximum Packet-Size Registers
GMX0_RX(0..2)_JABBER

This register specifies the maximum size for packets, beyond which the GMX truncates. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	CNT	R/W	0x2800	0x2800	Byte count for jabber check. Failing packets set the JABBER interrupt and are optionally sent with opcode = JABBER. GMX truncates the packet to CNT bytes. ● CNT must be 8-byte aligned such that CNT[2:0] = 000. ● CNT ≥ GMX0_RX(0..2)_FRM_MAX[LEN].

Notes:

The packet that is sent to the packet-input logic will have an additional eight bytes if GMX0_RX(0..2)_FRM_CTL[PRE_CHK] = 1 and GMX0_RX(0..2)_FRM_CTL[PRE_STRP] = 0. The maximum packet that will be sent is defined as:

```
max_sized_packet =
GMX0_RXn_JABBER[CNT] +
((GMX0_RXn_FRM_CTL[PRE_CHK] & GMX0_RXn_FRM_CTL[PRE_STRP]) × 8)
```

CNT must be ≥ GMX0_RX(0..2)_FRM_MAX[LEN]. Smaller values cause packets that are within the LEN length to be rejected because they exceed the CNT limit.

GMX Packet Decision Registers
GMX0_RX(0..2)_DECISION

This register specifies the byte count used to determine when to accept or to filter a packet. As each byte in a packet is received by GMX, the L2 byte count (i.e. the number of bytes from the beginning of the L2 header (DMAC)) is compared against CNT. In normal operation, the L2 header begins after the PREAMBLE + SFD (GMX0_RX(0..2)_FRM_CTL[PRE_CHK] = 1) and any optional UDD skip data (GMX0_RX(0..2)_UDD_SKP[LEN]). See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved
<4:0>	CNT	R/W	0x18	0x18	The byte count used to decide when to accept or filter a packet. See to Table 13-9.

Notes:

When GMX0_RX(0..2)_FRM_CTL[PRE_CHK] = 0, PREAMBLE + SFD are prepended to the packet and require UDD skip length to account for them.

Table 13-9GMX Decisions

Port Mode	L2 Size[1]	
	≤GMX_RX_DECISION bytes (default = 0x18)	>GMX_RX_DECISION bytes (default = 0x18)
RGMII/Full Duplex	Accept packet No filtering is applied	Apply filters Accept packet based on DMAC and PAUSE packet filters
RGMII/Half Duplex	Drop packet Packet is unconditionally dropped	Apply filters Accept packet based on DMAC

1. where L2_size = MAX(0, total_packet_size – GMX0_RX(0..2)_UDD_SKP[LEN] – ((GMX0_RX(0..2)_FRM_CTL[PRE_CHK] = 1) × 8)

GMX User-Defined Data Skip Registers
GMX0_RX(0..2)_UDD_SKP

This register specifies the amount of user-defined data (UDD) added before the start of the L2C data. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:9>	—	RAZ	—	—	Reserved
<8>	FCSSEL	R/W	0x0	0	Include the skip bytes in the FCS calculation 0 = all skip bytes are included in FCS 1 = the skip bytes are not included in FCS The skip bytes are part of the packet and are sent through the I/O bus packet interface and are handled by PKI. The system can determine if the UDD bytes are included in the FCS check by using the FCSSEL field, if the FCS check is enabled.
<7>	—	RAZ	—	—	Reserved
<6:0>	LEN	R/W	0x0	0	Amount of user-defined data before the start of the L2C data, in bytes. 0 means L2C comes first; maximum value is 64. LEN must be 0x0 in half-duplex operation unless GMX0_RX(0..2)_FRM_CTL[PRE_CHK] = 0.

Notes:

Assume that the PREAMBLE/SFD is always at the start of the frame, even before UDD bytes. In most cases, there will be no preamble in these cases, since it will be RGMII-to-RGMII communication without a PHY involved.

In all cases, the UDD bytes are sent through the packet interface as part of the packet. The UDD bytes are never stripped from the actual packet.

If LEN ≠ 0 (i.e. there is some UDD), then length checking should be disabled (i.e. GMX0_RX(0..2)_FRM_CHK[LENERR] should be cleared and GMX0_RX(0..2)_INT_REG[LENERR] set to 0.

Address filtering and control-packet filtering can be performed as needed.

GMX RX Statistics Control Registers
GMX0_RX(0..2)_STATS_CTL

This register controls the receive statistics registers. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved
<0>	RD_CLR	R/W	0	0	Clear on read. When this bit is set, any receive statistics registers are cleared when read.

GMX Minimum Interframe-Gap Cycles Registers
GMX0_RX(0..2)_IFG

This register specifies the minimum number of interframe-gap (IFG) cycles between packets. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	—	—	Reserved
<3:0>	IFG	R/W	0xC	0xC	Minimum number IFG cycles between packets. Used to determine IFGERR level. 1000Mbs: IFG = 0.096µs or 12 clock cycles 100Mbs: IFG = 0.96µs or 24 clock cycles 10Mbs: IFG = 9.6µs or 24 clock cycles To simplify the programming model, IFG is doubled internally when GMX_PRT_CFG[SPEED] = 0.

InBand Link Status Registers
GMX0_RX(0..2)_RX_INBND

The fields in this register are only valid if the attached PHY is operating in RGMII mode and supports the optional in-band status (see Section 3.4.1 of the RGMII specification, version 1.3 for more information). See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	—	—	Reserved
<3>	DUPLEX	RO	0	—	RGMII inbound LinkDuplex: 0 = half-duplex, 1 = full-duplex
<2:1>	SPEED	RO	0x0	—	RGMII inbound LinkSpeed 00 = 2.5MHz 01 = 25MHz 10 = 125MHz 11 = Reserved
<0>	STATUS	RO	0	—	RGMII inbound LinkStatus: 0 = down, 1 = up

GMX Pause Drop Time Registers
GMX0_RX(0..2)_PAUSE_DROP_TIME

This register specifies the TIME field in a PAUSE packet that was dropped due to the GMX RX FIFO full condition. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	STATUS	R/W1C	0x0	—	The time extracted from the dropped PAUSE packet.

GMX RX Good Packets Registers
GMX0_RX(0..2)_STATS_PKTS

This register provides a count of good received packets. The following are examples of "not good" packets:

- packets recognized as PAUSE packets
- packets dropped due to the DMAC filter
- packets dropped due to FIFO full status
- packets that have opcodes ≠ 0 (FCS, Length, etc.).

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	CNT	RC/W	0x0	—	Count of received good packets.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_RX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

GMX RX Good Packets Octet Registers
GMX0_RX(0..2)_STATS_OCTS

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved.
<47:0>	CNT	RC/W	0x0	—	Octet count of received good packets.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_RX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

GMX RX Pause Packets Registers
GMX0_RX(0..2)_STATS_PKTS_CTL

This register provides a count of all packets received that were recognized as flow-control or PAUSE packets. Note that PAUSE packets with any kind of error are also counted in GMX0_RX(0..2)_STATS_PKTS_BAD.

- Pause packets can be optionally dropped or forwarded based on GMX0_RX(0..2)_FRM_CTL[CTL_DRP]. This count increments regardless of whether the packet is dropped.
- Pause packets are never counted in GMX0_RX(0..2)_STATS_PKTS.
- Packets dropped due to the DMAC filter are counted in GMX0_RX(0..2)_STATS_PKTS_DMAC and not here.

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	CNT	RC/W	0x0	—	Count of received flow-control or PAUSE packets.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_RX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

GMX RX Pause Packets Octet Registers GMX0_RX(0..2)_STATS_OCTS_CTL

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved.
<47:0>	CNT	RC/W	0x0	—	Octet count of received flow-control or PAUSE packets.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_RX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

GMX RX DMAC Packets Registers GMX0_RX(0..2)_STATS_PKTS_DMAC

This register provides a count of all packets received that were dropped by the DMAC filter. Packets that match the DMAC are dropped and counted here regardless of whether they were bad packets. These packets are never counted in GMX0_RX(0..2)_STATS_PKTS. Note that some packets that were not able to satisfy the DECISION_CNT may not actually be dropped by CN50XX, but they are counted here as if they were dropped. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	CNT	RC/W	0x0	—	Count of filtered DMAC packets.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_RX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

GMX RX DMAC Packets Octet Registers GMX0_RX(0..2)_STATS_OCTS_DMAC

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved.
<47:0>	CNT	RC/W	0x0	—	Octet count of filtered DMAC packets.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_RX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

GMX RX Overflow Packets Registers GMX0_RX(0..2)_STATS_PKTS_DRP

This register provides a count of all packets received that were dropped due to a full receive FIFO. It counts all packets dropped by the FIFO, both good and bad packets received. It does not count packets dropped by the DMAC or PAUSE-packet filters.

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved.
<31:0>	CNT	RC/W	0x0	—	Count of dropped packets.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_RX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

GMX RX Overflow Packets Octet Registers GMX0_RX(0..2)_STATS_OCTS_DRP

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved.
<47:0>	CNT	RC/W	0x0	—	Octet count of dropped packets.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_RX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

GMX RX Bad Packets Registers GMX0_RX(0..2)_STATS_PKTS_BAD

This register provides a count of all packets received with some error that were not dropped either due to the DMAC filter or lack of room in the receive FIFO. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31:0>	CNT	RC/W	0x0	—	Count of received packets with opcode ≠ 0

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_RX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

Address-Filtering Control Registers

GMX0_RX(0..2)_ADR_CTL

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	—	—	Reserved
<3>	CAM_MODE	R/W	0x0	—	Allow or deny DMAC address filter 0 = reject the packet on DMAC address match 1 = accept the packet on DMAC address match
<2:1>	MCST	R/W	0x0	—	Multicast Mode 0 = Use the address-filter CAM 1 = Force reject all multicast packets 2 = Force accept all multicast packets 3 = Reserved
<0>	BCST	R/W	0x1	—	Accept all broadcast packets

Algorithm:

Here is some pseudo-code that represents the address-filter behavior.

```
bool dmac_addr_filter(uint8 prt, uint48 dmac) {
  ASSERT(prt >= 0 && prt <= 3);
  if (is_bcst(dmac))                                  // broadcast accept
    return (GMX_RX{prt}_ADR_CTL[BCST] ? ACCEPT : REJECT);
  if (is_mcst(dmac) & GMX_RX{prt}_ADR_CTL[MCST] == 1)   // multicast reject
    return REJECT;
  if (is_mcst(dmac) & GMX_RX{prt}_ADR_CTL[MCST] == 2)   // multicast accept
    return ACCEPT;

  cam_hit = 0;

  for (i=0; i<8; i++) {
    if (GMX_RX{prt}_ADR_CAM_EN[EN<i>] == 0)
      continue;
    uint48 unswizzled_mac_adr = 0x0;
    for (j=5; j>=0; j--) {
       unswizzled_mac_adr = (unswizzled_mac_adr << 8) |
                                    GMX_RX{prt}_ADR_CAM{j}[ADR<i*8+7:i*8>];
    }
    if (unswizzled_mac_adr == dmac) {
      cam_hit = 1;
      break;
    }
  }

  if (cam_hit)
    return (GMX_RX{prt}_ADR_CTL[CAM_MODE] ? ACCEPT : REJECT);
  else
    return (GMX_RX{prt}_ADR_CTL[CAM_MODE] ? REJECT : ACCEPT);
}
```

Address-Filtering Control Enable Registers

GMX0_RX(0..2)_ADR_CAM_EN

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved.
<7:0>	EN	R/W	0x0	—	CAM-entry enable bits.

Address-Filtering CAM Control Registers
GMX0_RX(0..2)_ADR_CAM(0..5)

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	ADR	R/W	0x0	—	The DMAC address to match on. Each entry contributes eight bits to one of eight matchers. Write transactions to this register do not change the CSR when GMX0_PRT(0..2)_CFG[EN] is enabled. The CAM matches against unicast or multicast DMAC addresses.

GMX TX Clock Generation Registers
GMX0_TX(0..2)_CLK

This register provides the ability to regulate the TXC frequency. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	—	—	Reserved.
<5:0>	CLK_CNT	R/W	0x1	0x1	Clock control. Controls the RGMII TXC frequency. TXC (period) = GMI_REF_CLK (period) × CLK_CNT **NOTE:** Do not set CLK_CNT = 0, as it does not generate a clock. If GMX_PRT*n*_CFG[SPEED] = 0, CLK_CNT must be > 1. **NOTE:** In MII mode, CLK_CNT = 1.

Example: given a 125MHz PLL and 250MHz internal RGMII clock:

```
When CLK_CNT =  0x1 =  1 → 8ns ×  1 =   8ns = 125.0 MHz TXC clock
When CLK_CNT =  0x5 =  5 → 8ns ×  5 =  40ns =  25.0 MHz TXC clock
When CLK_CNT = 0x32 = 50 → 8ns × 50 = 400ns =   2.5 MHz TXC clock
```

TX Threshold Registers
GMX0_TX(0..2)_THRESH

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:7>	—	RAZ	—	—	Reserved
<6:0>	CNT	R/W	0x20	0x20	Number of 16-byte ticks to accumulate in the TX FIFO before sending on the RGMII interface. This field should be large enough to prevent underflow on the RGMII interface and must never be set to less than 0x4. This register cannot exceed the TX FIFO depth of 0x40 words.

TX Append Control Registers
GMX0_TX(0..2)_APPEND

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:4>	—	RAZ	—	—	Reserved
<3>	FORCE_FCS	R/W	1	1	Append the ethernet FCS on each pause packet when FCS = 0. PAD must also be set to 0.
<2>	FCS	R/W	1	1	Append the ethernet FCS on each packet.
<1>	PAD	R/W	1	1	Append PAD bytes such that a minimum-sized packet is transmitted.
<0>	PREAMBLE	R/W	1	1	Append the ethernet preamble on each transfer.

TX Slottime Counter Registers
GMX0_TX(0..2)_SLOT

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:10>	—	RAZ	—	—	Reserved
<9:0>	SLOT	R/W	0x200	0x200	Slottime (refer to 802.3 to set correctly) 10/100Mbs: set SLOT to 0x40 1000Mbs: set SLOT to 0x200

TX Burst-Counter Registers
GMX0_TX(0..2)_BURST

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	BURST	R/W	0x2000	0x2000	Burst (refer to 802.3 to set correctly) 10/100Mbs: set BURST to 0x0 1000Mbs: set BURST to 0x2000

RGMII SMAC Registers
GMX0_SMAC(0..2)

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved.
<47:0>	SMAC	R/W	0x0	—	The SMAC field is used for generating and accepting Control Pause packets.

TX Pause Packet Pause-Time Registers GMX0_TX(0..2)_PAUSE_PKT_TIME

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	TIME	R/W	0x60	—	Provides the pause_time field placed is outbound PAUSE packets, in 512 bit-times. Normally, TIME > GMX0_TX(0..2)_PAUSE_PKT_INTERVAL[INTERVAL].

Notes:

Choosing proper values of GMX0_TX(0..2)_PAUSE_PKT_TIME[TIME] and GMX0_TX(0..2)_PAUSE_PKT_INTERVAL[INTERVAL] can be challenging. Cavium Networks suggests that TIME be much greater than INTERVAL and GMX0_TX(0..2)_PAUSE_ZERO[SEND] be set to 1. This allows a periodic refresh of the PAUSE count and then when the backpressure condition is lifted, a PAUSE packet with TIME = 0 is sent indicating that CN50XX is ready for additional data.

If the system chooses to not set GMX0_TX(0..2)_PAUSE_ZERO[SEND], then Cavium Networks suggests that TIME and INTERVAL are programmed such that they satisfy the following rule:

INTERVAL ≤ TIME – (largest_pkt_size + IFG + pause_pkt_size)

where:

largest_pkt_size = the largest packet that the system can send (normally 1518 bytes)

IFG = the interframe gap

pause_pkt_size = the size of the PAUSE packet (normally 64 bytes).

RGMII TX Minimum-Size-Packet Registers GMX0_TX(0..2)_MIN_PKT

This register specifies the minimum size a packet should be. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved
<7:0>	MIN_SIZE	R/W	0x3B	0x3B	Minimum frame size in bytes before the FCS is applied. Padding is only appended when GMX0_TX(0..2)_APPEND[PAD] for the corresponding RGMII port is set.

TX Pause-Packet Transmission-Interval Registers GMX0_TX(0..2)_PAUSE_PKT_INTERVAL

This register specifies how often PAUSE packets are sent. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	INTERVAL	R/W	0xFF	—	Arbitrate for a PAUSE packet every (INTERVAL × 512) bit-times. Normally, 0 < INTERVAL < GMX0_TX(0..2)_PAUSE_PKT_TIME[TIME]. INTERVAL = 0 only send a single PAUSE packet for each backpressure event.

Notes:

Choosing proper values of GMX0_TX(0..2)_PAUSE_PKT_TIME[TIME] and GMX0_TX(0..2)_PAUSE_PKT_INTERVAL[INTERVAL] can be challenging. Cavium Networks suggests that TIME be much greater than INTERVAL and GMX0_TX(0..2)_PAUSE_ZERO[SEND] be set to 1. This allows a periodic refresh of the PAUSE count and then when the backpressure condition is lifted, a PAUSE packet with TIME = 0 is sent indicating that CN50XX is ready for additional data.

If the system chooses to not set GMX0_TX(0..2)_PAUSE_ZERO[SEND], then Cavium Networks suggests that TIME and INTERVAL are programmed such that they satisfy the following rule:

$$\text{INTERVAL} \leq \text{TIME} - (\text{largest\_pkt\_size} + \text{IFG} + \text{pause\_pkt\_size})$$

where:

largest_pkt_size = the largest packet that the system can send (normally 1518 bytes)

IFG = the interframe gap

pause_pkt_size = the size of the PAUSE packet (normally 64 bytes).

TX Software-Pause Registers GMX0_TX(0..2)_SOFT_PAUSE

This register specifies the pause time. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	TIME	R/W	0x0	—	Back off the TX bus for (TIME × 512) bit-times

TX Time-to-Backpressure Registers
GMX0_TX(0..2)_PAUSE_TOGO

This register is the pause-deferral counter. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved.
<15:0>	TIME	RO	—	—	Amount of time remaining to backpressure.

TX Pause-Zero-Enable Registers
GMX0_TX(0..2)_PAUSE_ZERO

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved
<0>	SEND	R/W	1	1	Send pause-zero enable.When this bit is set, and the backpressure condition is clear, it allows sending a pause packet with pause_time of 0 to enable the channel.

GMX TX Statistics Control Registers
GMX0_TX(0..2)_STATS_CTL

This register controls the transmit statistics registers. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved
<0>	RD_CLR	R/W	0	0	Clear on read. When this bit is set, any transmit statistics registers are cleared when read.

GMX Transmit Control Registers
GMX0_TX(0..2)_CTL

This register enables error checking in the STX. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved
<1>	XSDEF_EN	R/W	1	1	Enables the excessive-deferral check for statistics and interrupts.
<0>	XSCOL_EN	R/W	1	1	Enables the excessive-collision check for statistics and interrupts.

Transmit Statistics Registers 0
GMX0_TX(0..2)_STAT0

This register maintains statistics on excessive deferrals and excessive collisions. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	XSDEF	RC/W	0x0	—	Number of packets dropped (never successfully sent) due to excessive deferrals.
<31:0>	XSCOL	RC/W	0x0	—	Number of packets dropped (never successfully sent) due to excessive collision. Defined by GMX0_TX_COL_ATTEMPT[LIMIT].

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_TX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

Transmit Statistics Registers 1
GMX0_TX(0..2)_STAT1

This register maintains statistics on single collisions and multiple collisions. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	SCOL	RC/W	0x0	—	Number of packets sent with a single collision.
<31:0>	MSCOL	RC/W	0x0	—	Number of packets sent with multiple collisions but less than GMX0_TX_COL_ATTEMPT[LIMIT].

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_TX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

Transmit Statistics Registers 2
GMX0_TX(0..2)_STAT2

This register maintains statistics on the number of octets sent on the interface. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved
<47:0>	OCTS	RC/W	0x0	—	Number of total octets sent on the interface. Does not count octets from frames that were truncated due to collisions in half-duplex mode.

NOTE:
- Octet counts are the sum of all data transmitted on the wire, including packet data, pad bytes, FCS bytes, PAUSE bytes, and JAM bytes. The octet counts do not include the preamble byte or extend cycles.
- Cleared by either a write (of any value) or a read when GMX0_TX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

Transmit Statistics Registers 3
GMX0_TX(0..2)_STAT3

Used for GMX_TX_STATS_PKTS.

This register maintains statistics on the number of frames sent on the interface. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	—	RAZ	—	—	Reserved
<31:0>	PKTS	RC/W	0x0	—	Number of total frames sent on the interface. Does not count frames that were truncated due to collisions in half-duplex mode.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_TX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

Transmit Statistics Registers 4
GMX0_TX(0..2)_STAT4

This register maintains statistics on the number of packets sent on the interface with octet counts of 64 and less than 64. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	HIST1	RC/W	0x0	—	Number of packets sent with an octet count of 64.
<31:0>	HIST0	RC/W	0x0	—	Number of packets sent with an octet count < 64.

NOTE:
- Packet length is the sum of all data transmitted on the wire for the given packet, including packet data, pad bytes, FCS bytes, PAUSE bytes, and JAM bytes. The octet counts do not include the preamble byte or extend cycles.
- Cleared by either a write (of any value) or a read when GMX0_TX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

Transmit Statistics Registers 5
GMX0_TX(0..2)_STAT5

This register maintains statistics on the number of packets sent on the interface with octet counts between 65 and 127 and between 128 and 255. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	HIST3	RC/W	0x0	—	Number of packets sent with an octet count of 128 – 255.
<31:0>	HIST2	RC/W	0x0	—	Number of packets sent with an octet count of 65 – 127.

NOTE:
- Packet length is the sum of all data transmitted on the wire for the given packet, including packet data, pad bytes, FCS bytes, PAUSE bytes, and JAM bytes. The octet counts do not include the preamble byte or extend cycles.
- Cleared by either a write (of any value) or a read when GMX0_TX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

Transmit Statistics Registers 6
GMX0_TX(0..2)_STAT6

This register maintains statistics on the number of packets sent on the interface with octet counts between 256 and 511 and between 512 and 1023. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	HIST5	RC/W	0x0	—	Number of packets sent with an octet count of 512 – 1023.
<31:0>	HIST4	RC/W	0x0	—	Number of packets sent with an octet count of 256 – 511.

NOTE:
- Packet length is the sum of all data transmitted on the wire for the given packet, including packet data, pad bytes, FCS bytes, PAUSE bytes, and JAM bytes. The octet counts do not include the preamble byte or extend cycles.
- Cleared by either a write (of any value) or a read when GMX0_TX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

Transmit Statistics Registers 7
GMX0_TX(0..2)_STAT7

This register maintains statistics on the number of packets sent on the interface with octet counts between 1024 and 1518 and greater than 1518. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	HIST7	RC/W	0x0	—	Number of packets sent with an octet count of > 1518.
<31:0>	HIST6	RC/W	0x0	—	Number of packets sent with an octet count of 1024 – 1518.

NOTE:
- Packet length is the sum of all data transmitted on the wire for the given packet, including packet data, pad bytes, FCS bytes, PAUSE bytes, and JAM bytes. The octet counts do not include the preamble byte or extend cycles.
- Cleared by either a write (of any value) or a read when GMX0_TX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

Transmit Statistics Registers 8
GMX0_TX(0..2)_STAT8

This register maintains statistics on the number of packets sent to broadcast DMAC and to multicast DMAC. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	MCST	RC/W	0x0	—	Number of packets sent to multicast DMAC. Does not include BCST packets.
<31:0>	BCST	RC/W	0x0	—	Number of packets sent to broadcast DMAC. Does not include MCST packets.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_TX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.
- Note that the GMX determines if the packet is MCST or BCST from the DMAC of the packet. GMX assumes that the DMAC lies in the first six bytes of the packet as per the 802.3 frame definition. If the system requires additional data before the L2 header, then the MCST and BCST counters may not reflect reality and should be ignored by software.

Transmit Statistics Registers 9
GMX0_TX(0..2)_STAT9

This register maintains statistics on the number of underflow packets and control packets. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:32>	UNDFLW	RC/W	0x0	—	Number of underflow packets.
<31:0>	CTL	RC/W	0x0	—	Number of Control packets (PAUSE flow control) generated by GMX. It does not include control packets forwarded or generated by the cores.

NOTE:
- Cleared by either a write (of any value) or a read when GMX0_TX(0..2)_STATS_CTL[RD_CLR] is set.
- Counters will wrap.

GMX BIST Results Register
GMX0_BIST

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved
<11:0>	STATUS	RO	0x0	0	BIST results. 0 = pass, 1 = fail

RX Ports Register
GMX_RX_PRTS

This register specifies the number of ports, which indicates the number of FIFOs that make up the RX buffer. It always powers on to three ports. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved
<2:0>	PRTS	R/W	0x3	0x3	In RGMII mode, the RX buffer can be carved into several logical buffers depending on the number or implemented ports. 0 or 1 port = 512ticks / 4096bytes 2 ports = 256ticks / 2048bytes 3 ports = 128ticks / 1024bytes 4–8 ports = N/A

RX FIFO Packet-Drop Registers
GMX0_RX_BP_DROP(0..2)

This register provides the mark for an RX FIFO beyond which packets are dropped and not buffered. The actual watermark is dynamic in RGMII mode with respect to the GMX_RX_PRTS register. GMX_RX_PRTS controls the depth

of the port's FIFO, so as ports are added or removed, the drop point may change. The relationship of the values in GMX0_RX_BP_DROP(0..2), GMX0_RX_BP_ON(0..2), and GMX0_RX_BP_OFF(0..2) is shown Figure 13–6.

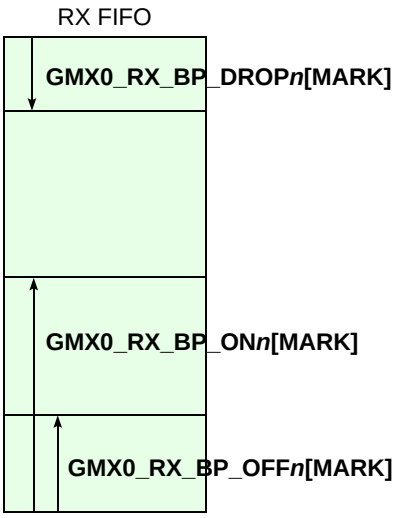


Figure 13–6 RX FIFO Values

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	—	—	Reserved
<5:0>	MARK	R/W	—	—	Number of eight-byte ticks to reserve in the RX FIFO. When the FIFO exceeds this count, packets are dropped and not buffered. MARK should typically be programmed to the number of ports + 1. Failure to program correctly can lead to system instability. ● Reset for RGMII mode = 0x2

RX Backpressure On Registers GMX0_RX_BP_ON(0..2)

This register provides the high-water mark for port/interface backpressure, the FIFO-full level at which backpressure is asserted. In RGMII mode, backpressure is given per port. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:9>	—	RAZ	—	—	Reserved.
<8:0>	MARK	R/W	—	—	High-water mark (in eight-byte ticks) for asserting backpressure. MARK must satisfy: BP_OFF[MARK] ≤ BP_ON[MARK] < (FIFO_SIZE – BP_DROP[MARK]) The default value is half the FIFO, which for RGMII mode = 0x40 (512 bytes). A value of 0x0 immediately asserts backpressure.

RX Backpressure Off Registers
GMX0_RX_BP_OFF(0..2)

This register provides the low-water mark for port/interface backpressure, the FIFO-full level at which backpressure can be deasserted. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	—	—	Reserved.
<5:0>	MARK	R/W	0x10	0x10	Low-water mark (in eight-byte ticks) to deassert backpressure.

TX Ports Register
GMX0_TX_PRTS

This register specifies the number of ports allowed on the interface, which is one higher than highest-numbered port (e.g. if port 3 is the highest port number, this register specifies 4). See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved.
<4:0>	PRTS	R/W	0x3	—	Number of ports allowed on the interface.

TX Interframe Gap Register
GMX0_TX_IFG

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved.
<7:4>	IFG2	R/W	0x4	—	1/3 of the interframe gap timing.
<3:0>	IFG1	R/W	0x8	—	2/3 of the interframe gap timing.

TX JAM Pattern Register
GMX0_TX_JAM

This register provides the pattern used in JAM bytes. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved.
<7:0>	JAM	R/W	0xEE	—	JAM pattern.

TX Collision Attempts Before Dropping Frame Register GMX0_TX_COL_ATTEMPT

This register provides the number of collision attempts allowed before dropping the frame. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved
<4:0>	LIMIT	R/W	0x10	0x10	Number of collision attempts allowed.

TX Pause-Packet DMAC-Field Register GMX0_TX_PAUSE_PKT_DMAC

This register provides the DMAC value that is placed in outbound PAUSE packets. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	—	RAZ	—	—	Reserved
<47:0>	DMAC	R/W	0x0180C2000001	0x0180C2000001	The DMAC field, which is placed is outbound PAUSE packets.

TX Pause Packet Type Field Register GMX0_TX_PAUSE_PKT_TYPE

This register provides the Type field that is placed in outbound PAUSE packets. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	TYPE	R/W	0x8808	0x8808	The Type field placed is outbound pause packets.

TX Override Backpressure Register GMX0_TX_OVR_BP

This register provides the ability to assert per-port backpressure, as well as the ability to override the assertion of backpressure for RGMII ports. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved.
<11>	—	R/W	0	0	Spare.
<10:8>	EN	R/W	0x0	0x0	Per-port backpressure enable/override (<8> for port 0, ..., <10> for port 2). Bits in this field must be set to allow backpressure to be asserted for the corresponding port. Ports that don't have the corresponding bit set are not able to assert backpressure.
<7>	—	R/W	0	0	Spare.
<6:4>	BP	R/W	0x0	0x0	Per-port backpressure status to use: (<4> for port 0, ..., <6> for port 2) 0 = port is available 1 = port should be backpressured

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<3>	—	R/W	0	0	Spare.
<2:0>	IGN_FULL	R/W	0x0	0x0	Ignore the RX FIFO full when computing backpressure (<0> for port 0, ..., <2> for port 2).

TX Backpressure Status Register GMX0_TX_BP

This register shows the status of backpressure for RGMII ports. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved
<2:0>	BP	RO	0x0	0x0	Per-port backpressure status: (<0> for port 0, ..., <2> for port 2) 0 = port is available, 1 = port should be backpressured

TX Corrupt Packets Register GMX0_TX_CORRUPT

This register enables the STX to corrupt a packet under certain conditions. Packets sent from PKO with the ERR wire asserted are corrupted by the transmitter if CORRUPT[port#] is set.

In RGMII mode, corruption means that GMX sends a bad FCS value. If GMX0_TX(0..2)_APPEND[FCS] is clear then no FCS is sent and the GMX cannot corrupt it. The corrupt FCS value is 0xEEEEEEEE. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved
<2:0>	CORRUPT	R/W	0x7	0x7	Per-port error propagation. 0 = Never corrupt packets 1 = Corrupt packets with ERR

RX Port State Information Register GMX0_RX_PRT_INFO

This register shows the contents of the linear feedback shift register (LFSR), which is used to implement truncated binary exponential backoff. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:19>	—	RAZ	—	—	Reserved
<18:16>	DROP	R/W	0x0	0x0	Per-port indication that data was dropped.
<15:3>	—	RAZ	—	—	Reserved
<2:0>	COMMIT	R/W	0x0	0x0	Per-port indication that SOP was accepted.

TX LFSR Register
GMX0_TX_LFSR

This register shows the contents of the linear feedback shift register (LFSR), which is used to implement truncated binary exponential backoff. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	LFSR	R/W	0xFFFF	—	Contains the current state of the LFSR, which is used to feed random numbers to compute truncated binary exponential backoff.

TX Interrupt Register
GMX0_TX_INT_REG

This register allows per-port interrupts to be sent to the control processor. If an error is noted in this register, and the corresponding bit in GMX0_TX_INT_EN is set, an exception is raised. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:20>	—	RAZ	—	—	Reserved
<19>	—	R/W1C	0	0	Spare bit.
<18:16>	LATE_COL	R/W1C	0x0	0x0	TX late collision
<15>	—	R/W1C	0	0	Spare bit.
<14:12>	XSDEF	R/W1C	0x0	0x0	TX excessive deferrals (RGMII/half-duplex mode only)
<11>	—	R/W1C	0	0	Spare bit.
<10:8>	XSCOL	R/W1C	0x0	0x0	TX excessive collisions (RGMII/half-duplex mode only)
<7:5>	—	R/W1C	0x0	0x0	Spare bits.
<4:2>	UNDFLW	R/W1C	0x0	0x0	TX underflow (RGMII mode only)
<1>	—	R/W1C	0	0	Spare bit.
<0>	PKO_NXA	R/W1C	0	0	Port address out-of-range from PKO interface.

TX Interrupt-Enable Register
GMX0_TX_INT_EN

This register provides the interrupt-enable bits for the corresponding error in GMX0_TX_INT_REG. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:20>	—	RAZ	—	—	Reserved
<19>	—	R/W	0	0	Spare bit.
<18:16>	LATE_COL	R/W	0x0	0x0	TX late collision
<15>	—	R/W	0	0	Spare bit.
<14:12>	XSDEF	R/W	0x0	0x0	TX excessive deferrals (RGMII/half-duplex mode only)
<11>	—	R/W	0	0	Spare bit.
<10:8>	XSCOL	R/W	0x0	0x0	TX excessive collisions (RGMII/half-duplex mode only)
<7:5>	—	R/W	0x0	0x0	Spare bits.
<4:2>	UNDFLW	R/W	0x0	0	TX Underflow (RGMII mode only)
<1>	—	R/W	0x0	0x0	Spare bits.
<0>	PKO_NXA	R/W	0x0	0	Port address out-of-range from PKO interface

Address-out-of-Range Error Register
GMX0_NXA_ADR

This register shows the logged address that caused an address-out-of-range error. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	—	—	Reserved
<5:0>	PRT	RO	0x0	—	Logged address for address-out-of-range exceptions. The logged address is from the first exception that caused the problem. The I/O bus has a higher priority than PKO.

GMX Miscellaneous Error Register
GMX_BAD_REG

This register contains a variety of error conditions that were not covered in other error registers. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:31>	—	RAZ	—	—	Reserved.
<30:27>	INB_NXA	R/W1C	0x0	0x0	Inbound port > GMX0_RX_PRTS
<26>	STATOVR	R/W1C	0	0	TX statistics overflow.
<25>	—	R/W1C	0	0	Spare bit.
<24:22>	LOSTSTAT	R/W1C	0x0	0x0	TX statistics data was over-written (per RGM port). TX statistics registers are corrupted.
<21:18>	—	RAZ	0x0	0x0	Reserved. Safe to write, read as 0x0.
<17:5>	—	R/W1C	0x0	0x0	Spare bits.
<4:2>	OUT_OVR	R/W1C	0x0	0x0	Outbound data FIFO overflow (per port).
<1:0>	—	R/W1C	0x0	0x0	Spare bits.

GMX Backpressure Statistics Register
GMX_STAT_BP

This register indicates the number of cycles that the TX/Stats block has held up operation. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:17>	—	RAZ	—	—	Reserved.
<16>	BP	RO	0	0	Outbound I/O bus FIFO overflow.
<15:0>	CNT	R/W1C	0x0	0x0	Outbound collision occurred between PKO and the I/O bus.

Mode Change Mask Registers
GMX0_TX_CLK_MSK0/1

Register 0 is used for port 0, and register 1 is used for ports 1 and 2 . See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	MSK	R/W	0	—	Set this bit to 1 when switching clocks.

GMX RX/TX Status Register
GMX0_RX_TX_STATUS

This register provides the status since the last read operation. See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:7>	—	RAZ	—	—	Reserved.
<6:4>	TX	RC	0x0	0x0	Transmit data since last read operation.
<3>	—	RAZ	—	—	Reserved.
<2:0>	RX	RC	0x0	0x0	Receive data since last read operation.

Interface Mode Register
GMX0_INF_MODE

See Table 13–8 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved.
<2>	P0MII	R/W	—	—	Port 0 configuration: 0 = RGMII port, 1 = MII port.
<1>	EN	R/W	—	—	Interface enable. This field must be set to enable the packet interface. It should be enabled before any other requests to GMX, including enabling port backpressure with IPD_CTL_STATUS[PBP_EN].
<0>	TYPE	R/W	—	—	Ports 1 and 2 configuration: 0 = both ports are RGMII ports 1 = port 1 is either GMII or MII port, port 2 is disabled (GMII or MII port is selected by GMX_PRT1_CFG[SPEED]: 0 = MII, 1 = GMII.).

13.9 ASX Registers

The ASX registers are listed in Table 13-10.

Table 13-10 ASX Registers

Register	Address	CSR Type[1]	Detailed Description
ASX0_RX_PRT_EN	0x00011800B0000000	RSL	See page 558
ASX0_TX_PRT_EN	0x00011800B0000008	RSL	See page 559
ASX0_INT_REG	0x00011800B0000010	RSL	See page 559
ASX0_INT_EN	0x00011800B0000018	RSL	See page 559
ASX0_RX_CLK_SET0 ... ASX0_RX_CLK_SET2	0x00011800B0000020 ... 0x00011800B0000030	RSL	See page 560
ASX0_PRT_LOOP	0x00011800B0000040	RSL	See page 561
ASX0_TX_CLK_SET0 ... ASX0_TX_CLK_SET2	0x00011800B0000048 ... 0x00011800B0000058	RSL	See page 562
ASX0_TX_COMP_BYP	0x00011800B0000068	RSL	See page 562
ASX0_TX_HI_WATER000 ... ASX0_TX_HI_WATER002	0x00011800B0000080 ... 0x00011800B0000090	RSL	See page 562
ASX0_GMII_RX_CLK_SET	0x00011800B0000180	RSL	See page 563
ASX0_GMII_RX_DAT_SET	0x00011800B0000188	RSL	See page 563
ASX0_MII_RX_DAT_SET	0x00011800B0000190	RSL	See page 563

1. RSL-type registers are accessed indirectly across the I/O bus.

RGMII RX Port Enable Register ASX0_RX_PRT_EN

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved
<2:0>	PRT_EN	R/W	0x0	0x1	Port enable. Must be set for CN50XX to receive RMGII traffic. When this bit clear for a given port, all the RGMII cycles appear as interframe cycles.

RGMII TX Port Enable Register
ASX0_TX_PRT_EN

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved
<2:0>	PRT_EN	R/W	0x0	0x1	Port enable. Must be set for CN50XX to receive RMGII traffic. When this bit clear for a given port, all the RGMII cycles appear as interframe cycles.

RGMII Interrupt Register
ASX0_INT_REG

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:11>	—	RAZ	—	—	Reserved.
<10:8>	TXPSH	R/W1C	0x0	0x0	TX FIFO overflow on RMGII port.
<7>	—	RAZ	—	—	Reserved.
<6:4>	TXPOP	R/W1C	0x0	0x0	TX FIFO underflow on RMGII port.
<3>	—	RAZ	—	—	Reserved.
<2:0>	OVRFLW	R/W1C	0x0	0x0	RX FIFO overflow on RMGII port.

RGMII Interrupt-Enable Register
ASX0_INT_EN

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:11>	—	RAZ	—	—	Reserved
<10:8>	TXPSH	R/W	0x0	0x1	TX FIFO overflow on RMGII port
<7>	—	RAZ	—	—	Reserved
<6:4>	TXPOP	R/W	0x0	0x1	TX FIFO underflow on RMGII port
<3>	—	RAZ	—	—	Reserved
<2:0>	OVRFLW	R/W	0x0	0x1	RX FIFO overflow on RMGII port

RGMII RX Clock-Delay Registers
ASX0_RX_CLK_SET(0..2)

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved
<4:0>	SETTING	R/W	0x18	0x0	Delay setting to place on the open-loop RXC (RGMII receive clock) delay line, which can delay the received clock. This field can be used if the board and/or transmitting device has not otherwise delayed the clock. A value of 0x0 disables the delay line. The delay line should be disabled unless the transmitter or board does not delay the clock. **NOTE:** Note that this delay line provides only a coarse control over the delay. Generally, it can only reliably provide a delay in the range 1.25 – 2.5 ns, which may not be adequate for some system applications. The open loop delay line selects from among a series of tap positions. Each incremental tap position adds a delay of 50 ps to 135 ps per tap, depending on the chip, its temperature, and the voltage. To achieve from 1.25 – 2.5 ns of delay on the received clock, a fixed value of 0x18 may work. For more precision, Cavium recommends the following settings based on the chip voltage: **V_{DD} SETTING V_{DD} SETTING** 1.0 0x12 1.2 0x17 1.05 0x13 1.25 0x18 1.1 0x15 1.3 0x19 1.15 0x16

RGMII Internal Loopback-Enable Register ASX0_PRT_LOOP

TX FIFO output goes into RX FIFO.

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:7>	—	RAZ	—	—	Reserved
<6:4>	EXT_LOOP	R/W	0x0	0x0	External-loopback enable (per-port) 0 = No loopback (TX FIFO is filled by RMGII) 1 = RX FIFO drives the TX FIFO ● GMX_PRT_CFG[DUPLEX] must be 1 (full-duplex mode) ● GMX_PRT_CFG[SPEED] must be 1 (GigE speed) ● core clock > 250MHZ ● RXC must not deviate from the ±50ppm ● if TXC>RXC, idle cycle may drop over time
<3>	—	RAZ	—	—	Reserved
<2:0>	PRT_LOOP	R/W	0x0	0x0	Internal-loopback enable (per-port) 0 = No loopback (RX FIFO is filled by RMGII) 1 = TX FIFO drives the RX FIFO ● GMX_PRT_CFG[DUPLEX] must be 1 (full-duplex mode) ● GMX_PRT_CFG[SPEED] must be 1 (GigE speed) ● GMX_TX_CLK[CLK_CNT] must be 1 **NOTE:** In internal loop-back mode, the RGMII link status is not used (since there is no real PHY). Software cannot use the inband status.

RGMII TX Clock-Delay Registers ASX0_TX_CLK_SET(0..2)

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved
<4:0>	SETTING	R/W	0x18	0x0	Delay setting to place on the open-loop TXC (RGMII transmit clock) delay line, which can delay the transmitted clock. This field can be used if the board and/or transmitting device has not otherwise delayed the clock. A value of 0x0 disables the delay line. The delay line should be disabled unless the transmitter or board does not delay the clock. **NOTE:** Note that this delay line provides only a coarse control over the delay. Generally, it can only reliably provide a delay in the range 1.25 – 2.5 ns, which may not be adequate for some system applications. The open loop delay line selects from among a series of tap positions. Each incremental tap position adds a delay of 50 ps to 135 ps per tap, depending on the chip, its temperature, and the voltage. To achieve from 1.25 – 2.5 ns of delay on the received clock, a fixed value of 0x18 may work. For more precision, Cavium recommends the following settings based on the chip voltage: V_{DD} SETTING V_{DD} SETTING 1.0 0x12 1.2 0x17 1.05 0x13 1.25 0x18 1.1 0x15 1.3 0x19 1.15 0x16

RGMII Compensation Clock-Delay Register ASX0_TX_COMP_BYP

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:17>	—	RAZ	—	—	Reserved.
<16>	BYPASS	R/W	0	0	Compensation bypass.
<15:13>	—	RAZ	—	—	Reserved.
<12:8>	PCTL	R/W	0x10	—	PCTL compensation value.
<7:5>	—	RAZ	—	—	Reserved.
<4:0>	NCTL	R/W	0x10	—	NCTL compensation value.

RGMII TX FIFO High Watermark Registers ASX0_TX_HI_WATER(0..2)

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved.
<2:0>	MARK	R/W	0x0	0x0	TX FIFO high watermark to stall GMX. Value of 0x0 maps to 0x8.

RGMII Clock Delay Register
ASX0_GMII_RX_CLK_SET

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved
<4:0>	SETTING	R/W	0x0	0x0	Setting to place on the RXCLK (GMII receive clock) delay line. The intrinsic delay can range from 50ps to 80ps per tap.

RGMII Receive Data Delay Register
ASX0_GMII_RX_DAT_SET

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved
<4:0>	SETTING	R/W	0x0	0x0	Setting to place on the RXD (GMII receive data) delay line. The intrinsic delay can range from 50ps to 80ps per tap.

RGMII Receive Data Delay Register
ASX0_MII_RX_DAT_SET

See Table 13-10 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved
<4:0>	SETTING	R/W	0x0	0x0	Setting to place on the RXD (MII receive data) delay line. The intrinsic delay can range from 50ps to 80ps per tap.

Chapter 14

PCM/TDM Interface

This chapter contains the following subjects:

Overview

The PCM/TDM unit is a highly flexible implementation capable of connecting to many variations on the TDM serial interface.

The key features are:

- Two-bit clock (BCLK)/FSYNC pairs.
  - Each is capable of being generated or received
  - Supports all frequencies from 256KHz to 32.768 MHz
  - 32 to 512 (256KHz to 32.768MHz) channels are supported (assuming standard 8-KHz FSYNC signal)
  - FSYNC polarity, location, length, and sample points are highly configurable
  - Support for 1, 2, or 4 clocks per bit
  - Support for T1 frame bit
  - Extra and missed FSYNC detection
- Four data highways
  - Each can be associated with either of the two BCLK/FSYNC pairs
  - Transmit/receive on any combination of channels
  - Support for LSB-first or MSB-first data transmission/reception
  - Highly configurable sample points and drive times
  - Transmit and receive DMA engines per highway (for a total of 8)
    - Transmit engine
      - 16-byte FIFO to buffer output data
      - Programmable fetch threshold between 0 and 15 bytes
      - Programmable fetch size between 1 and 16 bytes
      - Configurable transmit-memory region size
      - Interrupts based on number of frames read and memory-region wrap
    - Receive engine
      - Received data written to main memory with no processor intervention
      - Configurable receive-memory region size
      - Interrupts based on number of frames written and memory-region wrap

A high-level block diagram is shown in Figure 14–1.

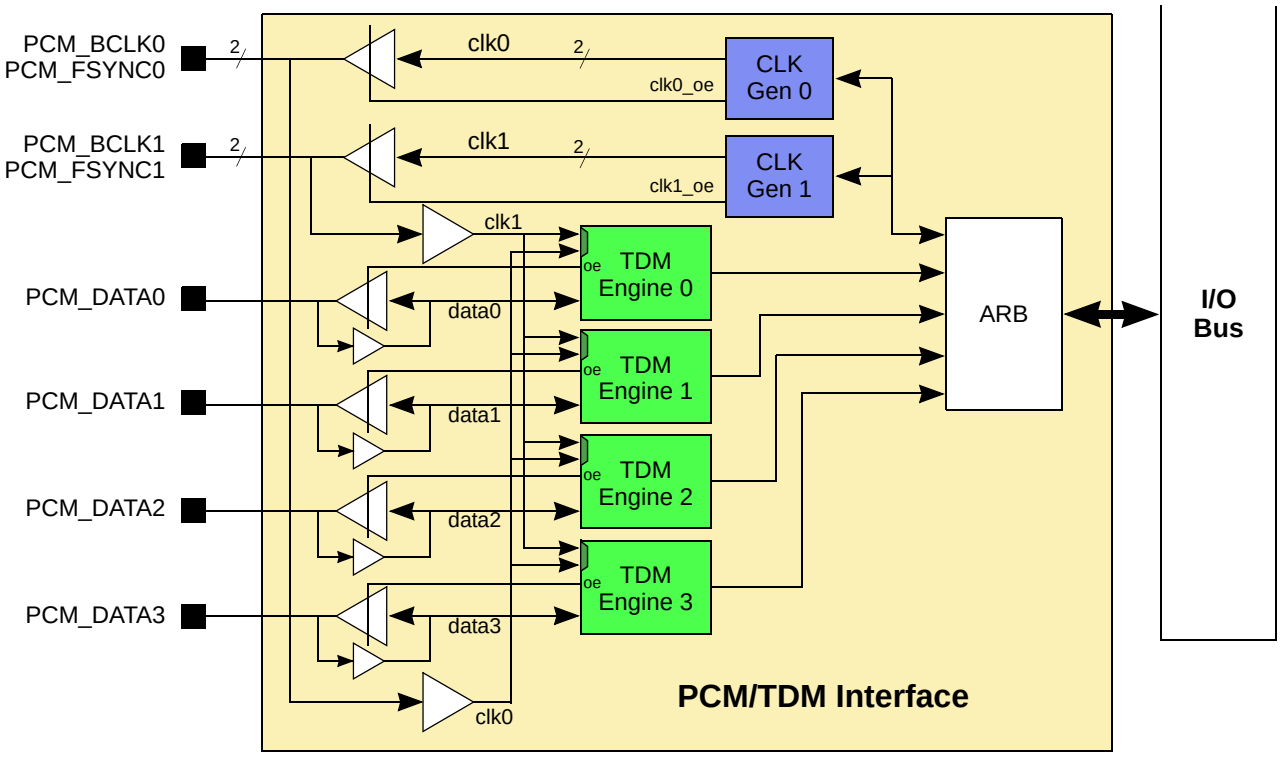


Figure 14–1 PCM/TDM Block Diagram

14.1 Signal Usage

All PCM/TDM signals are dual-function signals. When they are not being used for PCM/TDM purposes, they can function as GPIO pins (see Chapter 15 for more details). Table 14–1 describes the PCM/TDM signals and the CSR bits that switch them from standard GPIO pins to their PCM/TDM functionality. In addition, GPIO_XBIT_CFG(16..19)[TX_OE] or GPIO_BIT_CFG(12..15)[TX_OE] must be 0 for any of these pins when using the PCM/TDM functionality associated with the given pin.

Table 14–1 Signal Functionality

Signal Name	Enable	Description
GPIO_19/PCM_BCLK0	PCM_CLK0_GEN[N] ≠ 0x0	Bit clock for clock pair 0
GPIO_18/PCM_FSYNC0	PCM_CLK0_GEN[N] ≠ 0x0	FSYNC for clock pair 0
GPIO_17/PCM_DATA1	PCM1_TDM_CFG[ENABLE] = 1	Data highway for TDM engine 1
GPIO_16/PCM_DATA0	PCM0_TDM_CFG[ENABLE] = 1	Data highway for TDM engine 0
GPIO_15/PCM_BCLK1	PCM_CLK1_GEN[N] ≠ 0x0	Bit clock for clock pair 1
GPIO_14/PCM_FSYNC1	PCM_CLK1_GEN[N] ≠ 0x0	FSYNC for clock pair 1
GPIO_13/PCM_DATA3	PCM3_TDM_CFG[ENABLE] = 1	Data highway for TDM engine 3
GPIO_12/PCM_DATA2	PCM2_TDM_CFG[ENABLE] = 1	Data highway for TDM engine 2

As Table 14–1 shows, most of the pins can be independently used in PCM/TDM mode and GPIO mode. For the remainder of this chapter, the pins will be referred to by their PCM/TDM names.

14.2 Clocking

The PCM/TDM unit has completely independent BCLK/FSYNC pairs. The PCM_CLK0/1_CFG and PCM_CLK0/1_GEN registers control the characteristics of each clock pair, as shown in Table 14–2.

Table 14–2 BCLK/FSYNC Generation

CSR Field	Description
PCM_CLK*n*_GEN[N]	Controls the frequency of the bit clock generated. Must be 0 for reception.
PCM_CLK*n*_GEN[NUMSAMP]	Specifies the number of BCLK samples required to detect a change in BCLK. A value of 0 effectively removes filtering.
PCM_CLK*n*_CFG[BCLKPOL]	If set to 1, this field inverts the PCM_BCLKn pin on both transmissions and receptions.
PCM_CLK*n*_CFG[BITLEN]	Specifies the number of BCLK cycles per bit time.
PCM_CLK*n*_CFG[NUMSLOTS]	Specifies the number of 8-bit timeslots per frame (maximum: 512 – EXTRABIT).
PCM_CLK*n*_CFG[EXTRABIT]	Indicates whether an extra frame bit is present.
PCM_CLK*n*_CFG[FSYNCLOC]	Specifies the number of BCLK edges before the first bit of the frame at which FSYNC asserts.
PCM_CLK*n*_CFG[FSYNCLEN]	Specifies the number of BCLK edges for which FSYNC remains asserted.
PCM_CLK*n*_CFG[FSYNCPOL]	Indicates whether FSYNC should assert low or high.
PCM_CLK*n*_CFG[FSYNCSAMP]	Specifies the number of core clock cycles after BCLK edge to sample PCM_FSYNC*n*.

14.2.1 BCLK Generation

The BCLK clock-generator frequency is controlled by the following relation:

$$f(BCLK) = \frac{f(ECLK) \times PCM\_CLKn\_GEN[N]}{2^{32}}$$

where:
f(BCLK) is the frequency of BCLK
f(ECLK) is the frequency of ECLK (core clock)

- To stop the BCLK clock generator, PCM_CLK*n*_GEN[N] must be written to 0. This also disables the driving of the PCM_BCLK*n* and PCM_FSYNC*n* pins.
- Any value other than 0 turns the PCM_BCLK*n* and PCM_FSYNC*n* pins into outputs and prevents their use as a GPIO pin. Supported BCLK frequencies range from 256 KHz to 32.768 MHz.

An example of a BCLK frequency is shown in Table 14–3.

Table 14–3 Sample BCLK Frequency

Desired BCLK Frequency	ECLK Frequency	PCM_CLK*n*_GEN[N]
8.192 MHz	350 MHz	0x05FDEAB9
	500 MHz	0x225C17D0

14.2.2 FSYNC Generation

The FSYNC generator is enabled whenever the corresponding BCLK generator is enabled. The exact shape of the FSYNC waveform is determined by the fields PCM_CLK*n*_CFG[BITLEN, NUMSLOTS, EXTRABIT, FSYNCLOC, FSYNCLEN, FSYNCPOL].

The FSYNC frequency will be whatever is required to satisfy settings for these fields. If a standard 8-KHz FSYNC frequency is desired, [BITLEN, NUMSLOTS, EXTRABIT]must have the correct values for the chosen BCLK frequency.

14.2.3 BCLK Reception

BCLK is received by the TDM/PCM module via sampling the PCM_BCLKn pin into the ECLK (core clock) domain. This is true whether CN50XX is generating BCLK or not. The PCM_CLK*n*_GEN[NUMSAMP] setting is used to filter noise on a BCLK input with a slow transition time. If the CN50XX is generating BCLK, it can be set to 0.

Based on these settings, an internal BCLK is generated with one-rise edge per bit time. This internal BCLK is used for FSYNC reception and all data sampling and driving in the TDM engines.

14.2.4 FSYNC Reception

The PCM_FSYNC0/1 pin is sampled to detect an assertion. The sample point is defined as a number of ECLK (core clock) cycles (specified by PCM_CLK*n*_CFG[FSYNCSAMP]) after a BCLK falling or rising edge. If PCM_CLK*n*_CFG[FSYNCLOC] is odd a falling edge is used; if it is even a rising edge is used.

Once an assertion has been detected, it is moved by the number of BCLK edges specified by PCM_CLK*n*_CFG[FSYNCLOC] and the start of the frame (bit 0 of bit time 0) is aligned to this point. If this does not match the expected position of FSYNC assertion, an FSYNC_EXTRA error is logged in the PCM_INT_SUM register associated with any TDM engines using the BCLK/FSYNC pair. In addition, the TDM engine stops transmitting and receiving. An optional interrupt may be sent as well.

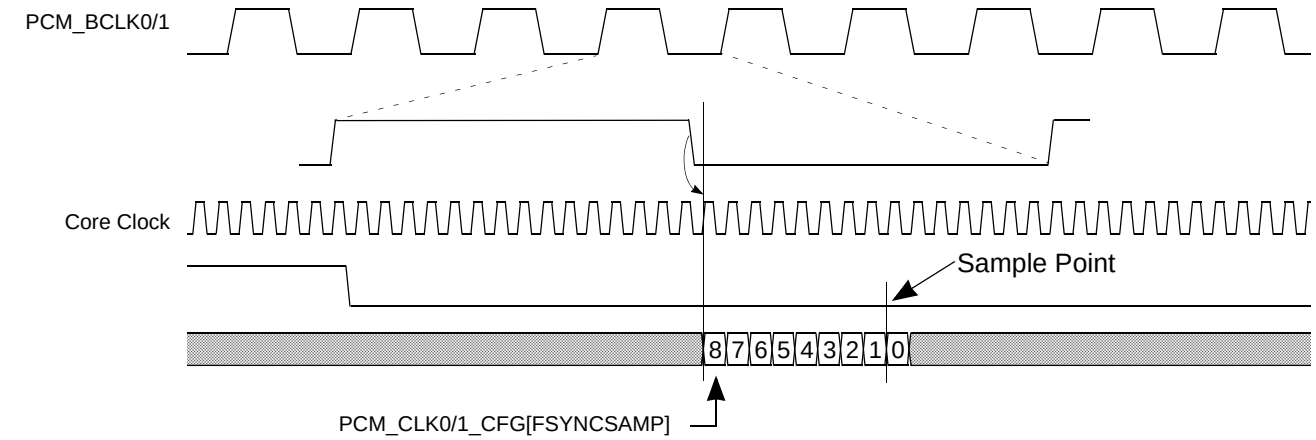
A second check on FSYNC reception looks at when the internal counters think an FSYNC assertion should have happened. If no FSYNC assertion is seen, an FSYNC_MISSING error is logged in the PCM_INT_SUM register associated with any TDM engines using the BCLK/FSYNC pair. In addition, the TDM engine stops transmitting and receiving. An optional interrupt may be sent as well.

14.2.5 Examples BCLK/FSYNC Waveforms

14.2.5.1 FSYNC Sampling

Examples of FSYNC sampling are shown in Figure 14–2.

FSYNC Sampling with [FSYNCLOC] = odd and [FSYNCSAMP] = 0x8



FSYNC Sampling with [FSYNCLOC] = even and [FSYNCSAMP] = 0x8

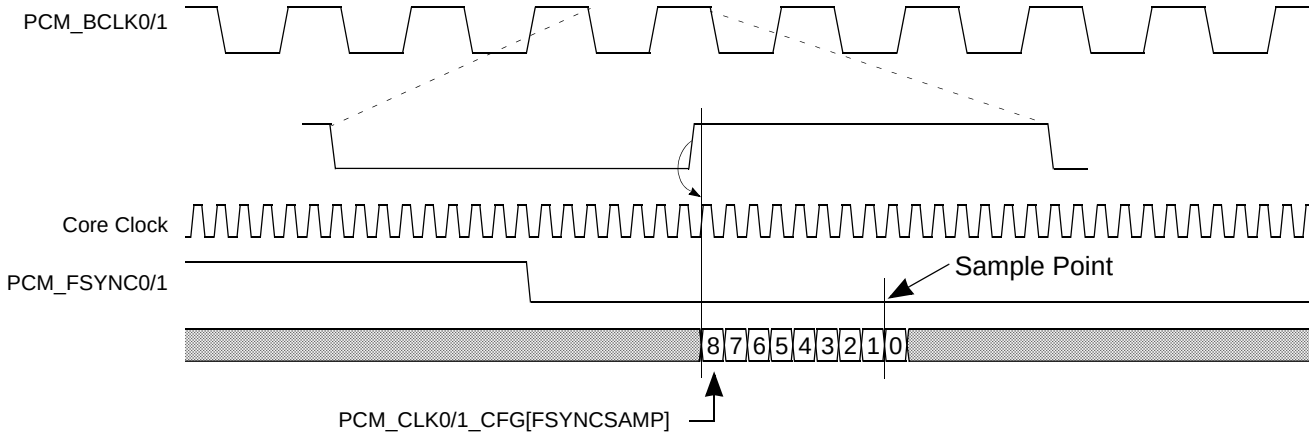


Figure 14–2 FSYNC Sampling

14.2.5.2 Internal BCLK

An example of BCLK waveforms are shown in Figure 14–3.

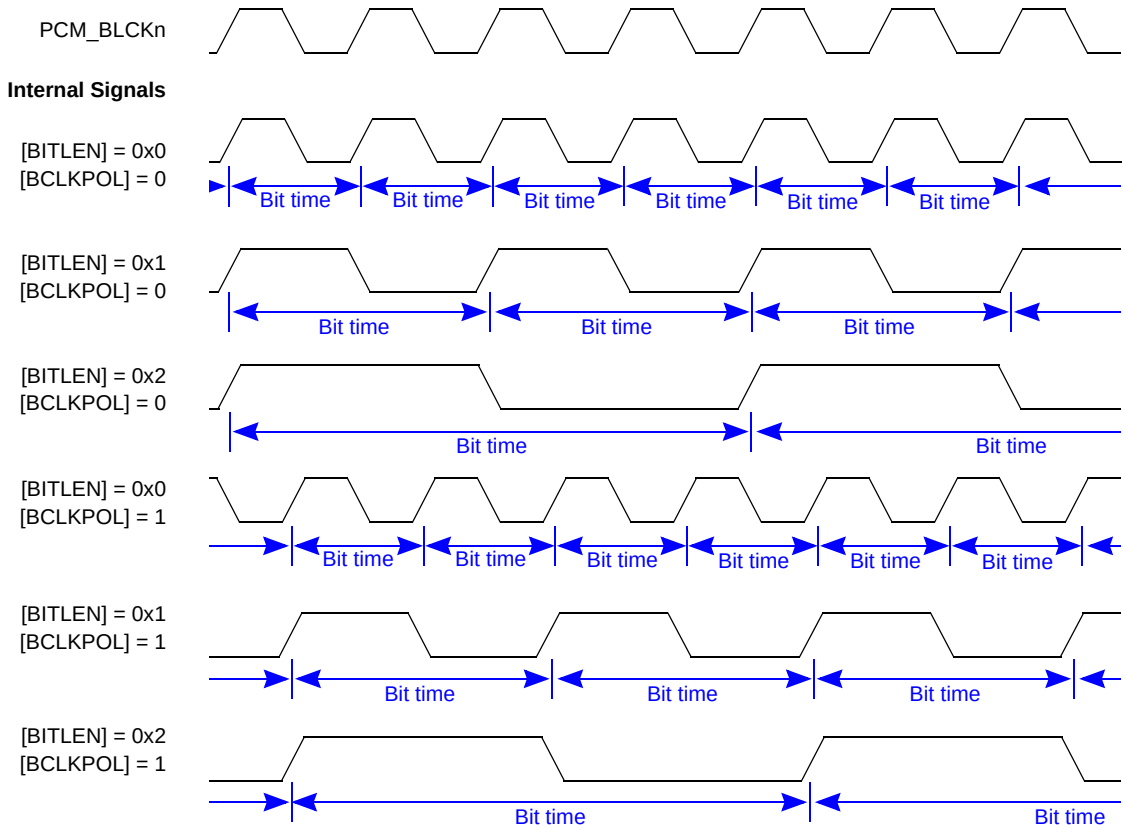


Figure 14–3 BCLK Waveforms

14.3 TDM Engines

The CN50XX has four TDM engines, each attached to its own PCM_DATA*n* pin. Configuration for the engine is controlled via the PCM*n*_TDM_CFG and PCM*n*_DMA_CFG registers. A TDM engine cannot operate without its corresponding DMA engines. The TDM-engine block diagram is shown in Figure 14–4.

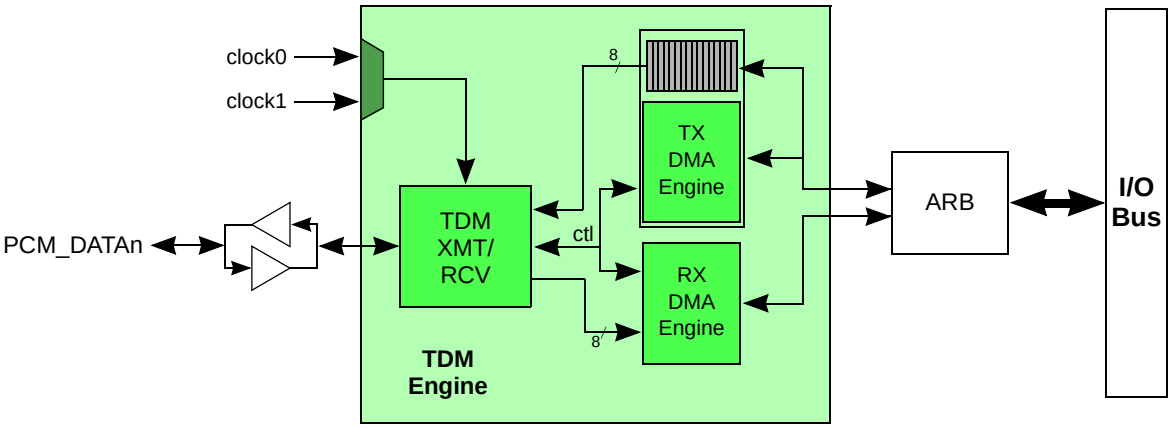


Figure 14–4 TDM-Engine Block Diagram

14.3.1 TDM Engine Configuration

Each TDM engine can use either of the two BCLK/FSYNC pairs. The PCM*n*_TDM_CFG[USECLK1] field selects which one is used. Note that the TDM engine must be disabled before changing the [USECLK1] setting.

The total number of slots in a frame is inherited from the chosen BCLK/FSYNC pair (PCM_CLK0/1_CFG[NUMSLOTS]). The number of transmit and receive slots are specified in each TDM engine via the PCM*n*_TDM_CFG[RXSLOTS, TXSLOTS] settings.

Note that each TDM engine can transmit or receive on any of the timeslots independently. The exact timeslots that are transmitted/received is specified by the PCM*n*_TXMSK(0..7) and PCM*n*_RXMSK(0..7) registers. The sum of all 1s in the PCM*n*_TXMSK(0..7) registers must equal PCM*n*_TDM_CFG[TXSLOTS]. Likewise, the sum of all 1s in the PCM*n*_RXMSK(0..7) registers must equal PCM*n*_TDM_CFG[RXSLOTS].

The PCM*n*_TDM_CFG[LSBFIRST] bit, when set to 1, tells the TDM engine to send the least-significant bit of the data byte first instead of the most-significant bit, otherwise the most-significant bit is sent first.

PCM*n*_TDM_CFG[SAMPPT] controls the sample point for receiving data from the PCM_DATA*n* pin. It specifies the number of ECLK (core clock) cycles from the start of a bit time to sample the pin.

When the TDM engine reaches the end of a transmission timeslot and is not going to be transmitting in the next timeslot, PCM*n*_TDM_CFG[DRVTIM] specifies the number of ECLK (core clock) cycles from the start of that final bit time that the CN50XX's driver should turn off. This allows the CN50XX to release the PCM_DATA*n* wire so another device can start driving the next timeslot without contention.

The PCM*n*_TDM_CFG[ENABLE] bit switches the PCM_DATA*n* pin from a GPIO to the TDM functionality.

Examples of data sampling and data driving are shown in Figure 14–5.

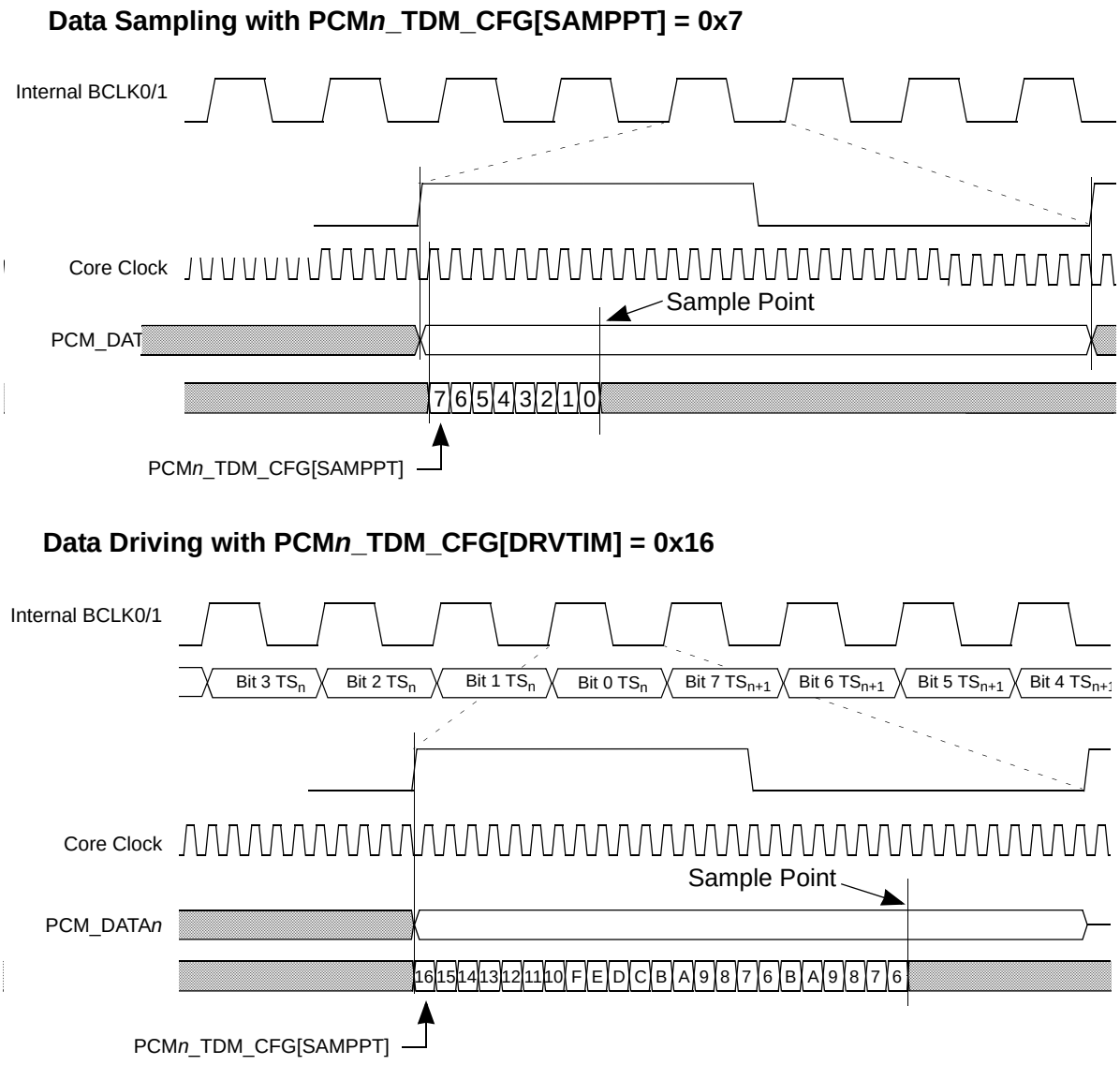


Figure 14–5 Data Sampling/Data Driving

14.3.2 DMA Engines

There are two DMA engines in each TDM engine: one to receive data and write it to main memory and one to read data from main memory for transmission. Both DMA engines use a common memory layout for the memory regions written to and read from.

14.3.2.1 Transmit/Receive Memory Regions

The format of the memory regions used for transmitting and receiving data is optimized to best use the CN50XX L2 cache. To prevent large portions of the L2 cache from being swept with each frame, eight frames worth of data for a single channel are stored per 64-bit word in memory. A group of 64-bit words comprising all timeslots in a frame contains all data to be transmitted or received for eight complete frames, and is termed a superframe.

The PCM*n*_TXSTART register points to the start of the memory-transmit region, and PCM*n*_TXCNT specifies the number of superframes in the memory-transmit region.

For example, a four-superframe memory region to support transmission of three timeslots (timeslots 5, 9, and 12) would be laid out in memory as shown in Table 14–4, with the following parameters set:

PCMn_TXCNT = 0x4
PCMn_DMA_CFG[TXSLOTS] = 3
PCMn_TXMSK0<5, 9, 12> = 1

Table 14–4 Superframe Memory-Region Example

Address	Slot	Data							
		[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
PCM*n*_TXSTART + 0	TS5	TS5/Fr0	TS5/Fr1	TS5/Fr2	TS5/Fr3	TS5/Fr4	TS5/Fr5	TS5/Fr6	TS5/Fr7
PCM*n*_TXSTART + 8	TS9	TS9/Fr0	TS9/Fr1	TS9/Fr2	TS9/Fr3	TS9/Fr4	TS9/Fr5	TS9/Fr6	TS9/Fr7
PCM*n*_TXSTART + 16	TS12	TS12/Fr0	TS12/Fr1	TS12/Fr2	TS12/Fr3	TS12/Fr4	TS12/Fr5	TS12/Fr6	TS12/Fr7
PCM*n*_TXSTART + 24	TS5	TS5/Fr8	TS5/Fr9	TS5/Fr10	TS5/Fr11	TS5/Fr12	TS5/Fr13	TS5/Fr14	TS5/Fr15
PCM*n*_TXSTART + 32	TS9	TS9/Fr8	TS9/Fr9	TS9/Fr10	TS9/Fr11	TS9/Fr12	TS9/Fr13	TS9/Fr14	TS9/Fr15
PCM*n*_TXSTART + 40	TS12	TS12/Fr8	TS12/Fr9	TS12/Fr10	TS12/Fr11	TS12/Fr12	TS12/Fr13	TS12/Fr14	TS12/Fr15
PCM*n*_TXSTART + 48	TS5	TS5/Fr16	TS5/Fr17	TS5/Fr18	TS5/Fr19	TS5/Fr20	TS5/Fr21	TS5/Fr22	TS5/Fr23
PCM*n*_TXSTART + 56	TS9	TS9/Fr16	TS9/Fr17	TS9/Fr18	TS9/Fr19	TS9/Fr20	TS9/Fr21	TS9/Fr22	TS9/Fr23
PCM*n*_TXSTART + 64	TS12	TS12/Fr16	TS12/Fr17	TS12/Fr18	TS12/Fr19	TS12/Fr20	TS12/Fr21	TS12/Fr22	TS12/Fr23
PCM*n*_TXSTART + 72	TS5	TS5/Fr24	TS5/Fr25	TS5/Fr26	TS5/Fr27	TS5/Fr28	TS5/Fr29	TS5/Fr30	TS5/Fr31
PCM*n*_TXSTART + 80	TS9	TS9/Fr24	TS9/Fr25	TS9/Fr26	TS9/Fr27	TS9/Fr28	TS9/Fr29	TS9/Fr30	TS9/Fr31
PCM*n*_TXSTART + 88	TS12	TS12/Fr24	TS12/Fr25	TS12/Fr26	TS12/Fr27	TS12/Fr28	TS12/Fr29	TS12/Fr30	TS12/Fr31

PCM*n*_TXSTART must point to an eight-byte-aligned physical address. The memory-region size can be very large if required: PCM*n*_TXCNT can range up to 65535, with a maximum of 512 slots per frame; the maximum memory region could be nearly 32MB, holding data for 524,280 frames.

When the BCLK/FSYNC pair specifies that an extra frame bit is in the frame (i.e. PCM_CLK*n*_CFG[EXTRABIT] is set to 1), the value for that bit is stored as the first channel of a frame. The bit resides in either the MSB or the LSB of the data byte, depending on the setting of PCM*n*_TDM_CFG[LSBFIRST] and whether it is a transmission or reception. The bit placement is shown in Table 14–5.

Table 14–5 LSB/MSB Bit Placement

	PCM*n*_TDM_CFG[LSBFIRST]	
	= 0	= 1
Transmission	MSB	LSB
Reception	LSB	MSB

The memory receive region is handled in the same manner using the PCM*n*_RXSTART and PCM*n*_RXCNT CSRs.

14.3.2.2 Transmit DMA Engine

The transmit DMA engine is configured via the PCM*n*_DMA_CFG, PCM*n*_TXSTART, and PCM*n*_TXCNT registers.

- The PCM*n*_TXSTART, PCM*n*_TXCNT registers and PCM*n*_DMA_CFG[TXSLOTS] field describe the transmit-memory region (see Section 14.3.2.1 for complete details).

  The PCM*n*_DMA_CFG[TXSLOTS] field tells the DMA engine the number of timeslots that are actually transmitted during each frame. This can range from all to none of the total number of slots in a frame. This number also is the number of 64-bit words in a superframe.

- The DMA engine's data requests are controlled via the PCM*n*_DMA_CFG[THRESH] and PCM*n*_DMA_CFG[FETCHSIZ] fields.

  The transmit DMA engine has a 16-byte FIFO to hold data waiting to be sent. When the number of bytes remaining in the FIFO is less than or equal to PCM*n*_DMA_CFG[THRESH], the DMA engine requests up to PCM*n*_DMA_CFG[FETCHSIZ] + 1 (the real fetch size) 64-bit words and extracts the bytes associated with the needed frame. Because of this behavior it is required that:

  $$(\text{threshold setting}) + (\text{the real fetch size}) \leq 16$$

  Otherwise the 16-byte FIFO will overflow.

The transmit DMA engine does not fetch across a frame boundary. In the case where there are fewer bytes remaining in the frame than the real fetch size, the DMA engine instead fetches all the remaining bytes in the frame and reevaluates the FIFO status upon receiving those bytes. Each transmit DMA engine can have only one request to main memory at a time.

There are two interrupts available to let the core know that the transmit memory region needs attention.

- The first is the TXRD interrupt, which signals an interrupt after PCM*n*_DMA_CFG[TXRD] full frames (see Table 14–4) have been read by the transmit DMA engine.
- The second option is the TXWRAP interrupt. This interrupt signals to the core when the transmit DMA engine has finished reading the last superframe of the transmit-memory region (i.e. it is about to start reading the first superframe again – it has wrapped to the beginning of the transmit-memory region).

One or both of these interrupts can be disabled if desired. If both are disabled the core will need to poll the PCM*n*_INT_SUM register to determine when the transmit memory region needs attention.

In addition, a TXEMPTY interrupt occurs if the transmit FIFO is empty when the TDM engine needs another byte to transmit. Note that this is an error. Operation is undefined after a TXEMPTY condition is encountered.

14.3.2.3 Receive DMA Engine

The receive DMA engine is configured via the PCM*n*_DMA_CFG, PCM*n*_RXSTART, and PCM*n*_RXCNT registers.

- The PCM*n*_RXSTART, PCM*n*_RXCNT registers and PCM*n*_DMA_CFG[RXSLOTS] field describe the transmit-memory region (see Section 14.3.2.1 for complete details).

  The PCM*n*_DMA_CFG[RXSLOTS] field tells the DMA engine the number of timeslots that are actually received during each frame. This can range from all to none of the total number of slots in a frame. This number also is the number of 64-bit words in a superframe.

When a data byte is received by the TDM engine, it is immediately sent to the receive DMA engine, which writes each byte to main memory as soon as is practical. No write merging is done.

There are two interrupts available to let the CPU know that the receive memory region needs attention.

- The first option is the RXST interrupt, which signals an interrupt after PCM*n*_DMA_CFG[RXST] full frames have been written by the receive DMA engine.
- The second option is the RXWRAP interrupt. This interrupt signals to the core when the receive DMA engine has finished writing the last superframe of the receive-memory region (i.e. it is about to start writing the first superframe again – it has wrapped to the beginning of the receive-memory region).

These interrupts are not signaled until confirmation of the write to main memory has been received. No other interlock is required to assure the core that it will be able to see the data.

One or both of these interrupts can be disabled if desired. If both are disabled the CPU needs to poll the PCM*n*_INT_SUM register to determine when the receive-memory region needs attention.

In addition, an RXOVF interrupt occurs if the previous data byte has not been sent to main memory when the TDM engine provides another byte to send. This is an error condition indicating receive data has been lost.

14.4 Initialization Sequence

Perform the following steps in order to initialize the PCM/TDM interface. To determine the best values, refer to Sections 14.2 and 14.3.

1. Initialize the clock generation/reception by writing the appropriate values to PCM_CLK*n*_CFG and PCM_CLK*n*_GEN registers.
2. Setup transmission and reception parameters for each TDM engine with the following steps:
   - write the appropriate value to PCM*n*_DMA_CFG.
   - disable the FSYNCEXTRA and FSYNCMISSED interrupts by writing PCM*n*_INT_ENA[FSYNCEXTRA, FSYNCMISSED] = 0.
   - configure memory regions by writing the appropriate values to:
     PCM*n*_TXSTART[ADDR]
     PCM*n*_TXCNT[CNT]
     PCM*n*_TXMSK(0..7)[MASK]
     PCM*n*_RXSTART[ADDR]
     PCM*n*_RXCNT[CNT]
     PCM*n*_RXMSK(0..7)[MASK]
   - write PCM*n*_TDM_CFG[ENABLE] = 0.
3. Wait for clock reception to stabilize (i.e. read PCM_CLK*n*_CFG until [FSYNCGOOD] = 1).
4. Clear the FSYNCEXTRA and FSYNCMISSED interrupts by writing PCM*n*_INT_SUM[FSYNCEXTRA, FSYNCMISSED] = 1.
5. Enable the desired interrupts by writing 1 to the appropriate fields in PCM*n*_INT_ENA.
6. Write PCM*n*_TDM_CFG[ENABLE] = 1 to begin transmission and reception of PCM data.

14.5 PCM/TDM Registers

The PCM/TDM registers are listed in Table 14–6.

Table 14–6 PCM/TDM Registers

Register	Address	CSR Type[1]	Detailed Description
PCM_CLK0_CFG	0x0001070000010000	NCB	See page 584.
PCM_CLK0_GEN	0x0001070000010008	NCB	See page 585.
PCM0_TDM_CFG	0x0001070000010010	NCB	See page 585.
PCM0_DMA_CFG	0x0001070000010018	NCB	See page 586.
PCM0_INT_ENA	0x0001070000010020	NCB	See page 586.
PCM0_INT_SUM	0x0001070000010028	NCB	See page 587.
PCM0_TDM_DBG	0x0001070000010030	NCB	See page 587.
PCM_CLK0_DBG	0x0001070000010038	NCB	See page 587.
PCM0_TXSTART	0x0001070000010040	NCB	See page 587
PCM0_TXCNT	0x0001070000010048	NCB	See page 587
PCM0_TXADDR	0x0001070000010050	NCB	See page 588
PCM0_RXSTART	0x0001070000010058	NCB	See page 588
PCM0_RXCNT	0x0001070000010060	NCB	See page 588
PCM0_RXADDR	0x0001070000010068	NCB	See page 588
PCM0_TXMSK0 ... PCM0_TXMSK7	0x0001070000010080 ... 0x00010700000100B8	NCB	See page 588
PCM0_RXMSK0 ... PCM0_RXMSK7	0x00010700000100C0 ... 0x00010700000100F8	NCB	See page 590
PCM_CLK1_CFG	0x0001070000014000	NCB	See page 584.
PCM_CLK1_GEN	0x0001070000014008	NCB	See page 585.
PCM1_TDM_CFG	0x0001070000014010	NCB	See page 585.
PCM1_DMA_CFG	0x0001070000014018	NCB	See page 586.
PCM1_INT_ENA	0x0001070000014020	NCB	See page 586.
PCM1_INT_SUM	0x0001070000014028	NCB	See page 587.
PCM1_TDM_DBG	0x0001070000014030	NCB	See page 587.
PCM_CLK1_DBG	0x0001070000014038	NCB	See page 587.
PCM1_TXSTART	0x0001070000014040	NCB	See page 587
PCM1_TXCNT	0x0001070000014048	NCB	See page 587
PCM1_TXADDR	0x0001070000014050	NCB	See page 588
PCM1_RXSTART	0x0001070000014058	NCB	See page 588
PCM1_RXCNT	0x0001070000014060	NCB	See page 588
PCM1_RXADDR	0x0001070000014068	NCB	See page 588
PCM1_TXMSK0 ... PCM1_TXMSK7	0x0001070000014080 ... 0x00010700000140B8	NCB	See page 588
PCM1_RXMSK0 ... PCM1_RXMSK7	0x00010700000140C0 ... 0x00010700000140F8	NCB	See page 590
PCM2_TDM_CFG	0x0001070000018010	NCB	See page 585.
PCM2_DMA_CFG	0x0001070000018018	NCB	See page 586.
PCM2_INT_ENA	0x0001070000018020	NCB	See page 586.
PCM2_INT_SUM	0x0001070000018028	NCB	See page 587.

Table 14–6 PCM/TDM Registers (Continued)

Register	Address	CSR Type[1]	Detailed Description
PCM2_TDM_DBG	0x0001070000018030	NCB	See page 587.
PCM2_TXSTART	0x0001070000018040	NCB	See page 587
PCM2_TXCNT	0x0001070000018048	NCB	See page 587
PCM2_TXADDR	0x0001070000018050	NCB	See page 588
PCM2_RXSTART	0x0001070000018058	NCB	See page 588
PCM2_RXCNT	0x0001070000018060	NCB	See page 588
PCM2_RXADDR	0x0001070000018068	NCB	See page 588
PCM2_TXMSK0 ... PCM2_TXMSK7	0x0001070000018080 ... 0x00010700000180B8	NCB	See page 588
PCM2_RXMSK0 ... PCM2_RXMSK7	0x00010700000180C0 ... 0x00010700000180F8	NCB	See page 590
PCM3_TDM_CFG	0x000107000001C010	NCB	See page 585.
PCM3_DMA_CFG	0x000107000001C018	NCB	See page 586.
PCM3_INT_ENA	0x000107000001C020	NCB	See page 586.
PCM3_INT_SUM	0x000107000001C028	NCB	See page 587.
PCM3_TDM_DBG	0x000107000001C030	NCB	See page 587.
PCM3_TXSTART	0x000107000001C040	NCB	See page 587
PCM3_TXCNT	0x000107000001C048	NCB	See page 587
PCM3_TXADDR	0x000107000001C050	NCB	See page 588
PCM3_RXSTART	0x000107000001C058	NCB	See page 588
PCM3_RXCNT	0x000107000001C060	NCB	See page 588
PCM3_RXADDR	0x000107000001C068	NCB	See page 588
PCM3_TXMSK0 ... PCM3_TXMSK7	0x000107000001C080 ... 0x000107000001C0B8	NCB	See page 588
PCM3_RXMSK0 ... PCM3_RXMSK7	0x000107000001C0C0 ... 0x000107000001C0F8	NCB	See page 590

1. NCB-type registers are accessed directly across the I/O Bus.

PCM Clock Configuration Registers
PCM_CLK0/1_CFG

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	FSYNCGOOD	RO	0	1	FSYNC status. When set, the last frame had a correctly positioned FSYNC pulse. When clear, either no FSYNC pulse or an extra FSYNC pulse was seen on most recent frame. **NOTE:** This is intended for startup. The FSYNCEXTRA and FSYNCMISSING interrupts are intended for detecting loss of SYNC during normal operation.
<62:48>	—	RAZ	—	—	Reserved.
<47:32>	FSYNCSAMP	R/W	0x0	—	Number of ECLK (core clock) cycles from the appropriate internal BCLK edge to sample FSYNC. **NOTE:** Used initially to synchronize to the start of a frame, later used to check for FSYNC errors.
<31:26>	—	RAZ	—	—	Reserved.
<25:21>	FSYNCLEN	R/W	0x0	0x2	Number of ½ BCLK cycles FSYNC is asserted for. **NOTE:** Only used when PCM_CLK0/1_GEN[N] ≠ 0
<20:16>	FSYNCLOC	R/W	0x0	0x0	FSYNC location, in ½ BCLK cycles before timeslot 0, bit 0. **NOTE:** Also used to detect framing errors and therefore must have a correct value even if PCM_CLK0/1_GEN[N] = 0
<15:6>	NUMSLOTS	R/W	0x0	—	Number of eight-bit slots in a frame. **NOTE:** This, along with EXTRABIT and f(BCLK) determines FSYNC frequency when PCM_CLK0/1_GEN[N] ≠ 0 **NOTE:** Also used to detect framing errors and therefore must have a correct value even if PCM_CLK0/1_GEN[N]=0
<5>	EXTRABIT	R/W	0	0	Extra frame bit. When set, add one extra bit time for frame bit. When clear, no frame bit. **NOTE:** If PCM_CLK0/1_GEN[N] ≠ 0, FSYNC is delayed one extra bit time. **NOTE:** Also used to detect framing errors and therefore must have a correct value even if PCM_CLK0/1_GEN[N] = 0 **NOTE:** The extra bit comes from the LSB/MSB of the first byte of the frame in the transmit memory region.
<4:3>	BITLEN	R/W	0x0	0x0	Number of BCLK cycles in a bit time. 0 : 1 BCLK cycle 1 : 2 BCLK cycles 2 : 4 BCLK cycles 3 : undefined
<2>	BCLKPOL	R/W	0	0	BCLK polarity. When set, BCLK falling edge is start of bit time. When clear, BCLK rising edge is start of bit time. **NOTE:** Also used to detect framing errors and therefore must have a correct value even if PCM_CLK0/1_GEN[N]=0
<1>	FSYNCPOL	R/W	0	0	FSYNC polarity. When set, FSYNC idles high, asserts low. When clear, FSYNC idles low, asserts high. **NOTE:** Also used to detect framing errors and therefore must have a correct value even if PCM_CLK0/1_GEN[N] = 0.
<0>	ENA	R/W	0	0	Enable. When set, clock-receiving logic is looking for SYNC. When clear, clock-receiving logic is doing nothing,

PCM Clock Registers

PCM_CLK0/1_GEN

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	DELTASAMP	R/W	0x0	0x0	Signed number of ECLK (core clock) cycles to move sampled BCLK edge. **NOTE:** The complete number of ECLK cycles to move is: NUMSAMP + 2 + 1 + DELTASAMP NUMSAMP: to compensate for sampling delay + 2: to compensate for dual-rank synchronizer + 1: for uncertainty + DELTASAMP: for debugging
<47:32>	NUMSAMP	R/W	0x0	—	Number of ECLK (core clock) samples to detect BCLK change when receiving clock.
<31:0>	N	R/W	0x0	—	Determines BCLK frequency [f(BCLK)] when generating clock. $f(BCLK) = f(ECLK) \times \frac{N}{2^{32}} \rightarrow N = \frac{f(BCLK)}{f(ECLK)} \times 2^{32}$ **NOTE:** Writing N = 0 stops the clock generator, and causes BCLK and FSYNC to be received.

PCM TDM Configuration Registers

PCM(0..3)_TDM_CFG

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:48>	DRVTIM	R/W	0x0	—	Drive time. The number of ECLK (core clock) cycles from start of bit time to stop driving last bit of timeslot (if not driving next timeslot).
<47:32>	SAMPPT	R/W	0x0	—	Sample time. The number of ECLK (core clock) cycles from start of bit time to sample data bit.
<31:3>	—	RAZ	—	—	Reserved.
<2>	LSBFIRST	R/W	0	0	LSB first. When this bit is set to 1, shift/receive LSB first. When it is clear, shift/receive MSB first.
<1>	USECLK1	R/W	0	0	Use CLK1. When this bit is set to 1, this TDM engine is based on BCLK/FSYNC1. When it is clear, this TDM engine is based on BCLK/FSYNC0.
<0>	ENABLE	R/W	0	0	TDM engine enable. When this bit is set to 1, TDM engine is enabled, otherwise the pins are GPIO pins. **NOTE:** When TDM is disabled by detection of an FSYNC error, all transmission and reception is halted. In addition, PCMn_TX/RXADDR are updated to point to the position at which the error was detected.

PCM DMA Configuration Registers
PCM(0..3)_DMA_CFG

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	RDPEND	RO	0	0	Read pending. When set, L2C read responses are outstanding. When clear, no L2C read responses pending. **NOTE:** When restarting after stopping a running TDM engine, software must wait for RDPEND to read 0 before writing PCM*n*_TDM_CFG[ENABLE] to a 1.
<62:54>	—	RAZ	—	—	Reserved.
<53:44>	RXSLOTS	R/W	0x0	—	Number of eight-bit slots to receive per frame (i.e. number of slots in a receive superframe).
<43:42>	—	RAZ	—	—	Reserved.
<41:32>	TXSLOTS	R/W	0x0	—	Number of eight-bit slots to transmit per frame (i.e. number of slots in a transmit superframe).
<31:30>	—	RAZ	—	—	Reserved.
<29:20>	RXST	R/W	0x0	0x1	Number of frame writes for interrupt.
<19>	—	RAZ	—	—	Reserved.
<18>	USELDT	R/W	0	0	Use LDT command. When set, use LDT command to read from L2C. When clear, use LDI command to read from L2C.
<17:8>	TXRD	R/W	0x0	0x1	Number of frame reads for interrupt.
<7:4>	FETCHSIZ	R/W	0x0	0x7	FETCHSIZ+1 timeslots are read when threshold is reached.
<3:0>	THRESH	R/W	0x0	0x8	Threshold. If number of bytes remaining in the DMA FIFO is ≤ THRESH, initiate a fetch of timeslot data from the transmit memory region. **NOTE:** There are only 16 bytes of buffer for each engine so the settings for FETCHSIZ and THRESH must be such that the buffer is not overrun: THRESH + min(FETCHSIZ + 1,TXSLOTS) must be ≤ 16.

PCM Interrupt-Enable Registers
PCM(0..3)_INT_ENA

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved.
<7>	RXOVF	R/W	0	1	RX overflow. Enable interrupt if RX byte overflows.
<6>	TXEMPTY	R/W	0	1	TX empty. Enable interrupt on TX byte empty.
<5>	TXRD	R/W	0	1	TX read. Enable DMA engine frame-read interrupts.
<4>	TXWRAP	R/W	0	1	TX wrap. Enable TX region-wrap interrupts.
<3>	RXST	R/W	0	1	RX store. Enable DMA engine frame-store interrupts.
<2>	RXWRAP	R/W	0	1	RX wrap. Enable RX region-wrap interrupts.
<1>	FSYNCEXTRA	R/W	0	1	Enable FSYNC extra interrupts. **NOTE:** FSYNCEXTRA errors are defined as an FSYNC found in the wrong spot of a frame given the programming of PCM_CLK0/1_CFG[NUMSLOTS] and PCM_CLK0/1_CFG[EXTRABIT].
<0>	FSYNCMISSED	R/W	0	1	Enable FSYNC missed interrupts. **NOTE:** FSYNCMISSED errors are defined as an FSYNC missing from the correct spot in a frame given the programming of PCM_CLK0/1_CFG[NUMSLOTS] and PCM_CLK0/1_CFG[EXTRABIT].

PCM Interrupt Registers
PCM(0..3)_INT_SUM

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved.
<7>	RXOVF	R/W1C	0	0	RX byte overflowed.
<6>	TXEMPTY	R/W1C	0	0	TX byte was empty when sampled.
<5>	TXRD	R/W1C	0	0	DMA engine frame-read interrupt occurred.
<4>	TXWRAP	R/W1C	0	0	TX region wrap-interrupt occurred.
<3>	RXST	R/W1C	0	0	DMA engine frame-store interrupt occurred.
<2>	RXWRAP	R/W1C	0	0	RX region-wrap interrupt occurred.
<1>	FSYNCEXTRA	R/W1C	0	0	FSYNC extra interrupt occurred.
<0>	FSYNCMISSED	R/W1C	0	0	FSYNC missed interrupt occurred.

PCM TDM Debug Registers
PCM(0..3)_TDM_DBG

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	DEBUGINFO	RO	—	—	Miscellaneous debug information.

PCM Clock Debug Registers
PCM0/1_CLK_DBG

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	DEBUGINFO	RO	—	—	Miscellaneous debug information.

PCM Transmit-Memory Start Registers
PCM(0..3)_TXSTART

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	—	—	Reserved
<35:3>	ADDR	R/W	—	—	Starting physical address for the transmit memory region.
<2:0>	—	RAZ	—	—	Reserved

PCM Transmit Count Registers
PCM(0..3)_TXCNT

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved
<15:0>	CNT	R/W	—	—	Number of superframes in transmit memory region.

PCM Transmit Address Registers
PCM(0..3)_TXADDR

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	—	—	Reserved
<35:3>	ADDR	R/W	—	—	Physical address of the next read from the transmit memory region.
<2:0>	FRAM	R/W	—	—	Frame offset. **NOTE:** This is used to extract the correct byte from each 64-bit word read from the transmit memory region.

PCM Receive-Memory Start Registers
PCM(0..3)_RXSTART

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	—	—	Reserved.
<35:3>	ADDR	R/W	—	—	Starting physical address for the receive memory region.
<2:0>	—	RAZ	—	—	Reserved.

PCM Receive Count Registers
PCM(0..3)_RXCNT

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved.
<15:0>	CNT	R/W	—	—	Number of superframes in receive memory region.

PCM Receive Address Registers
PCM(0..3)_RXADDR

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	—	—	Reserved.
<35:0>	ADDR	R/W	—	—	Physical address of the next write operation to the receive memory region.

PCM Transmit Mask 0 Registers
PCM(0..3)_TXMSK0

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Transmit mask bits for slots 63 to 0 (1 = transmit, 0 = don't transmit)

PCM Transmit Mask 1 Registers
PCM(0..3)_TXMSK1

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Transmit mask bits for slots 127 to 64 (1 = transmit, 0 = don't transmit)

PCM Transmit Mask 2 Registers
PCM(0..3)_TXMSK2

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Transmit mask bits for slots 191 to 128 (1 = transmit, 0 = don't transmit)

PCM Transmit Mask 3 Registers
PCM(0..3)_TXMSK3

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Transmit mask bits for slots 255 to 192 (1 = transmit, 0 = don't transmit).

PCM Transmit Mask 4 Registers
PCM(0..3)_TXMSK4

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Transmit mask bits for slots 319 to 256 (1 = transmit, 0 = don't transmit)

PCM Transmit Mask 5 Registers
PCM(0..3)_TXMSK5

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Transmit mask bits for slots 383 to 320 (1 = transmit, 0 = don't transmit)

PCM Transmit Mask 6 Registers
PCM(0..3)_TXMSK6

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Transmit mask bits for slots 447 to 384 (1 = transmit, 0 = don't transmit)

PCM Transmit Mask 7 Registers
PCM(0..3)_TXMSK7

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Transmit mask bits for slots 511 to 448 (1 = transmit, 0 = don't transmit)

PCM Receive Mask 0 Registers
PCM(0..3)_RXMSK0

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Receive mask bits for slots 63 to 0 (1 = receive, 0 = don't receive).

PCM Receive Mask 1 Registers
PCM(0..3)_RXMSK1

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Receive mask bits for slots 127 to 64 (1 = receive, 0 = don't receive).

PCM Receive Mask 2 Registers
PCM(0..3)_RXMSK2

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Receive mask bits for slots 191 to 128 (1 = receive, 0 = don't receive).

PCM Receive Mask 3 Registers
PCM(0..3)_RXMSK3

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Receive mask bits for slots 255 to 192 (1 = receive, 0 = don't receive).

PCM Receive Mask 4 Registers
PCM(0..3)_RXMSK4

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Receive mask bits for slots 319 to 256 (1 = receive, 0 = don't receive).

PCM Receive Mask 5 Registers
PCM(0..3)_RXMSK5

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Receive mask bits for slots 383 to 320 (1 = receive, 0 = don't receive).

PCM Receive Mask 6 Registers
PCM(0..3)_RXMSK6

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Receive mask bits for slots 447 to 384 (1 = transmit, 0 = don't transmit).

PCM Receive Mask 7 Registers
PCM(0..3)_RXMSK7

See Table 14–6 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:0>	MASK	R/W	—	—	Receive mask bits for slots 511 to 448 (1 = transmit, 0 = don't transmit).

Chapter 15

GPIO Unit

This section contains the following subjects:

Overview

The GPIO interface is a collection of general-purpose pins that can be configured for various tasks. There are a total of 24 GPIO signals, and most can be appropriated for use by other functions, making them unavailable as GPIO pins.

The GPIO interface can operate in either of two modes: GPIO mode (normal operation, in which GPIO_DBG_ENA[DBG_ENA] = 0x0) or debug mode (in which GPIO_DBG_ENA[DBG_ENA] = 0x1FFFFF).

- Debug mode: When required, the GPIO interface can be placed into debug mode, in which the debug port can be driven onto GPIO bits <20:0>.
- GPIO mode: When debug mode is not asserted, only the low-order eight bits (bits <7:0>) function strictly as GPIO signals. The other 16 bits can be overridden to be used by other functions. The dual-function bits are shown in Table 15–1.

Table 15–1 GPIO Dual-Function Signals

GPIO Signals	Function	Override Enable
<23:20>	MPI/SPI	Write 1s to appropriate bits in MPI_CFG (refer to MPI/SPI Registers).
<19:12>	PCM/TDM	Write 1s to appropriate bits in PCM*n*_TDM_CFG and PCM_CLK*n*_CFG (refer to PCM/TDM Registers).
<11:8>	Boot Bus[1]	Write 1s to bits <11:8> in GPIO_BOOT_ENA.

1. If you use GPIO bits <11:8> as (low-active) boot-bus chip-enable signals, external pullup resistors must be placed on the signals.

When in normal operation (or GPIO mode), all GPIO signals that have not been overridden by other functions can be configured as needed as:

- a data-input signal
- a data-output signal

In addition, signals <15:0> only can be configured as one of the following:

- a level-sensitive interrupt signal
- an edge-triggered interrupt signal

GPIO_BIT_CFG*n* and GPIO_XBIT_CFG*n* control the pin mode of each GPIO signal, including a glitch-filter.

15.1 GPIO Operations

Figure 15–1 shows the GPIO cell.

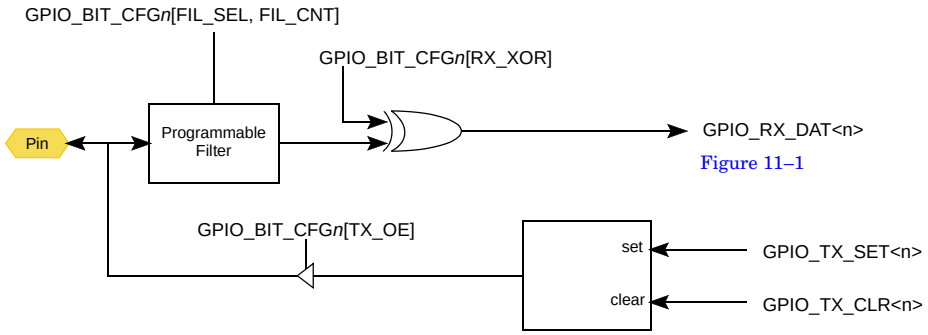


Figure 15–1 GPIO Cell

15.1.1 Reading the GPIO Bus

To read the GPIO signals, software simply reads the GPIO_RX_DAT register, which contains the state of the GPIO signals. Signals that have been overridden by other functions are undefined in this register.

15.1.2 Writing the GPIO Bus

To write the GPIO signals, software first picks which bits will be outputs by setting the appropriate GPIO_BIT_CFG*n*[TX_OE] bits to turn on the output drivers for the desired pins. Then software can change the data by using the GPIO_TX_SET and GPIO_TX_CLR registers. Both these registers use a bit mask to change data.

15.1.3 GPIO Interrupts

Finally, GPIO sends interrupt information to the CIU based on the GPIO_BIT_CFG*n*[INT_EN] and GPIO_BIT_CFG*n*[INT_TYPE]. If the pin is in edge-triggered mode, then software must clear the edge detector by writing to the GPIO_INT_CLR register.

15.2 Glitch Filters

For each GPIO bit, data comes in through the appropriate pin and is synchronized into the core-clock domain. It then enters the glitch filter.

The global counter counts every core-clock cycle. The glitch filter taps into the global counter to control the sample rate. The bit select chooses one of the 16 bits to watch for a positive edge and to sample the pin.

For example:

A value of 0 would select counter<0> and would sample every 4 ns.

A value of 4 would select counter<4> which would make a sample every 64ns.

Once a sample is made, if the sampled value ≠ current state, the local counter increments until the local counter = CSR count value.

If sampled value = current state, then the counter resets.

If the CSR count value = 0, then the filter is essentially disabled and just adds a pipe stage.

To get the two data points of 60ns and 1ms, the programming would be as follows (assuming a core-clock rate of 500 MHz):

60ns: bit_select = 0, count = 15

$$15 \times 4\text{ns per sample} = 60\text{ ns}$$

1ms: bit_select = 15, count = 8

$$8 \times [2^{(15+1)} \times 2]\text{ns per sample} = 1.048576\text{ ms}$$

or bit_select = 14, count = 15

$$15 \times [2^{(14+1)} \times 2]\text{ns per sample} = 0.983040\text{ ms}$$

15.3 GPIO Registers

The GPIO registers are listed in Table 15–2

Table 15–2 GPIO Registers

Register	Address	CSR Type[1]	Detailed Description
GPIO_BIT_CFG0 ... GPIO_BIT_CFG15	0x0001070000000800 ... 0x0001070000000878	NCB	See page 598
GPIO_RX_DAT	0x0001070000000880	NCB	See page 598
GPIO_TX_SET	0x0001070000000888	NCB	See page 598
GPIO_TX_CLR	0x0001070000000890	NCB	See page 598
GPIO_INT_CLR	0x0001070000000898	NCB	See page 599
GPIO_DBG_ENA	0x00010700000008A0	NCB	See page 599
GPIO_BOOT_ENA	0x00010700000008A8	NCB	See page 599
GPIO_XBIT_CFG16 ... GPIO_XBIT_CFG23	0x0001070000000900 ... 0x0001070000000938	NCB	See page 599

1. NCB-type registers are accessed directly across the I/O Bus.

GPIO Bit Configuration Registers
GPIO_BIT_CFG(0..15)

See Table 15–2 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved.
<11:8>	FIL_SEL	R/W	0x0	0x0	Filter select. Global counter bit-select (controls sample rate).
<7:4>	FIL_CNT	R/W	0x0	0x0	Filter count. Specifies the number of consecutive samples to change state.
<3>	INT_TYPE	R/W	0	0	Interrupt type: 0 = level-sensitive (default), 1 = rising-edge triggered.
<2>	INT_EN	R/W	0	0	Interrupt enable. Bit mask to indicate which bits to raise interrupt
<1>	RX_XOR	R/W	0	0	Receive inversion. When set to 1, inverts the received GPIO signal.
<0>	TX_OE	R/W	0	0	Transmit output enable. When set to 1, the GPIO pin is driven as an output pin.

GPIO Receive Data Register
GPIO_RX_DAT

See Table 15–2 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:24>	—	RAZ	—	—	Reserved.
<23:0>	DAT	RO	0x0	0x0	GPIO Read Data

GPIO Transmit Set Register
GPIO_TX_SET

See Table 15–2 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:24>	—	RAZ	—	—	Reserved.
<23:0>	SET	WO	0x0	0x0	Bit mask to indicate which bits to drive to 1.

GPIO Transmit Clear Register
GPIO_TX_CLR

See Table 15–2 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:24>	—	RAZ	—	—	Reserved.
<23:0>	CLR	WO	0x0	0x0	Bit mask to indicate which bits to drive to 0.

GPIO Interrupt Clear Register
GPIO_INT_CLR

See Table 15–2 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:16>	—	RAZ	—	—	Reserved.
<15:0>	TYPE	WO	0x0	0x0	Clear the interrupt rising-edge detector

GPIO Debug Enable Register
GPIO_DBG_ENA

See Table 15–2 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:21>	—	RAZ	—	—	Reserved.
<20:0>	DBG_ENA	RO	0x0	0x0	Enable the debug port to be driven onto the GPIO bus.

GPIO Boot Bus Enable Register
GPIO_BOOT_ENA

See Table 15–2 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved.
<11:8>	BOOT_ENA	RO	0x0	—	Drive boot-bus chip enable signals [7:4] onto GPIO bus bits <11:8>.
<7:0>	—	RAZ	—	—	Reserved.

GPIO Extended Bit Configuration Registers
GPIO_XBIT_CFG(16..23)

See Table 15–2 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved.
<11:8>	FIL_SEL	R/W	0x0	0x0	Global counter bit-select (controls sample rate)
<7:4>	FIL_CNT	R/W	0x0	0x0	Number of consecutive samples to change state
<3:2>	—	RAZ	—	—	Reserved.
<1>	RX_XOR	R/W	0	0	Invert the GPIO pin
<0>	TX_OE	R/W	0	0	Drive the GPIO pin as an output pin

Chapter 16

UART Interface

This chapter contains the following subjects:

Overview

CN50XX's two UART interfaces are four-pin serial interfaces. The UARTs are typically used for serial communication with a peripheral, modem (data carrier equipment, DCE), or data set. Either a cnMIPS core or a remote PCI host can use the UARTs. However, for the remainder of this chapter, we assume it is only the cores, though.

The CN50XX cores transfer bytes to and receives characters from the UART core via 64-bit CSR accesses. The UART core transfers and receives the characters serially. Either polling or interrupts can be used to transfer the bytes.

The UARTs have the following characteristics:

- modeled after the industry-standard 16550
  - programmable data bits per character, optional odd/even parity generation/ checking, and number of stop bits
  - line break generation and detection
- programmable baud rate = core-clock speed / (divisor × 16)
  - up to 10 Mbaud
- 64-byte transmit and receive FIFOs
- 16750-compatible auto flow control
- interrupts
  - several prioritized interrupt types
  - separate enables for each type
  - programmable transmit holding register empty interrupt
- shadow registers to read/write important fields individually
- software programmable reset
- testing/diagnostic capabilities
  - loopback for RTS/CTS testing
  - FIFO status and testing
  - transmit halt mode

16.1 UART (RS232) Serial Protocol

Because the serial communication is asynchronous, additional bits (start and stop) are added to the serial data to indicate the beginning and end. Using these bits allows two devices to be synchronized. This structure of serial data accompanied by start and stop bits is referred to as a character, as shown in Figure 16–1.

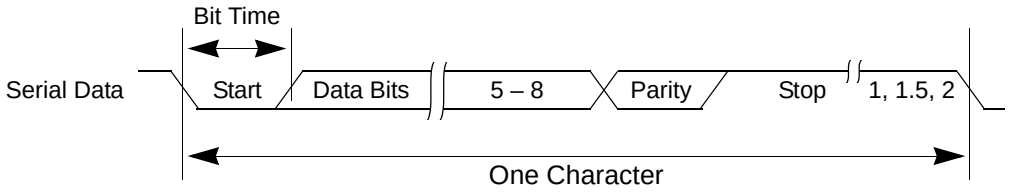


Figure 16–1 Serial Data Format

An additional parity bit may be added to the serial character. This bit appears after the last data bit and before the stop bit or bits in the character structure for simple error checking on the received data.

The line control register (MIO_UART_*n*_LCR) is used to control the serial-character characteristics. The individual bits of the data word are sent after the start bit, starting with the least-significant bit (LSB). These are followed by the optional parity bit, followed by the stop bits, which can be 1, 1.5 or 2.

All the bits in the transmission (with exception to the half stop bit when 1.5 stop bits are used) are transmitted for exactly the same time duration, which is referred to as a bit period or bit time. One bit time equals 16 baud clocks. To ensure stability on the line, the receiver samples the serial input data at approximately the mid point of the bit time once the start bit has been detected. As the exact number of baud clocks that each bit was transmitted for is known, calculating the mid point for sampling is not difficult, i.e. every 16 baud clocks after the mid point sample of the start bit. Figure 16–2 shows the sampling points of the first couple of bits in a serial character.

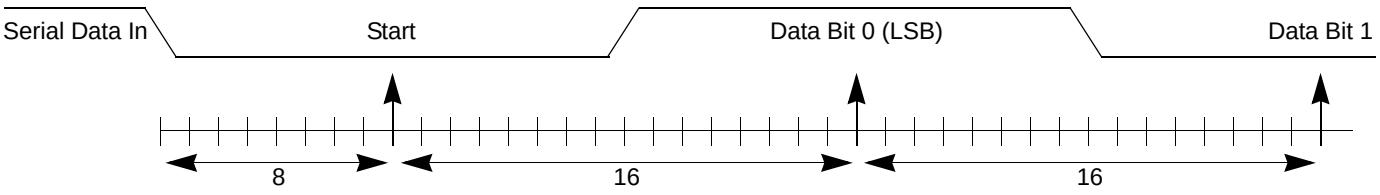


Figure 16–2 Receiver Serial Data Sample Points

The baud clock is a programmable number of core clock cycles selected by the divisor latch register (MIO_UART*n*_DLH and MIO_UART*n*_DLL). With a divisor of one, the baud clock equals the CN50XX core clock, and the baud rate is 1/16 the core clock.

16.2 UART Interrupts

For the following section, refer to Figure 11-2 in the CIU chapter. The UART interrupt asserts whenever one of the several prioritized interrupt types are enabled and active. That is, it asserts whenever MIO_UART*n*_IIR[IID] ≠ 1.

The following interrupt types can be enabled with the MIO_UART*n*_IER register:

- receiver error
- receiver data available
- character timeout (in FIFO mode only)
- transmitter holding register empty or at or below threshold (in programmable THRE interrupt mode)
- modem status changed
- busy detect indication

These interrupt types are covered in more detail in the MIO_UART0/1 _IIR description.

16.3 UART AutoFlow Control

CN50XX implements a 16750-compatible autoRTS and autoCTS serial-data flow-control mode. It is important to note that autoRTS and autoCTS modes can only be enabled when the FIFOs are enabled (i.e. when MIO_UART*n*_FCR[EN] is set).

16.3.1 UART AutoRTS

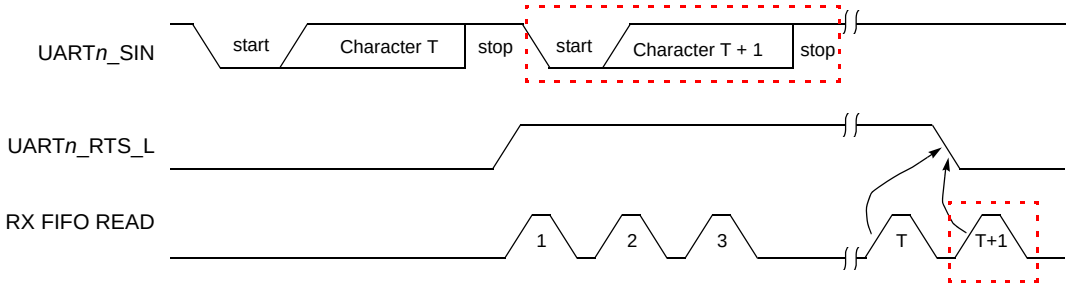
AutoRTS is active when:

- MIO_UART*n*_MCR[RTS] and MIO_UART*n*_MCR[AFCE] are both set
- The FIFOs are enabled (i.e. MIO_UART*n*_FCR[EN] is set)

When autoRTS is enabled (active), UART*n*_RTS_L is forced inactive (high) when the receiver FIFO level reaches the threshold set by MIO_UART*n*_FCR[RXTRIG]. When UART*n*_RTS_L is connected to the **cts_l** input of another UART device, the other UART stops sending serial data until the receiver FIFO has available space (until it is completely empty).

The selectable receiver FIFO threshold values are: 1, ¼, ½, and “2 less than full”. Since one additional character may be transmitted to CN50XX after **rts_n** has become inactive (due to data already having entered the transmitter block in the other UART), setting the threshold to “2 less than full” allows maximum use of the FIFO with a safety zone of one character.

Once the cores drain the receiver FIFO completely empty (by reading MIO_UART*n*_RBR), UART*n*_RTS_L again becomes active (low), signalling the other UART to continue sending data. Figure 16–3 shows a timing diagram of autoRTS operation.



T = Receiver FIFO Threshold Value

Character **T+1** is received because UART*n*_RTS_L was not detected before the next character entered the sending UART's transmitter.

Figure 16–3 AutoRTS Timing

When autoRTS is disabled, UART*n*_RTS_L is controlled solely by MIO_UART*n*_MCR[RTS].

16.3.2 UART AutoCTS

AutoCTS is active when:

- MIO_UART*n*_MCR[AFCE] is set to 1;
- The FIFOs are enabled (i.e. MIO_UART*n*_FCR[EN] is set).

When autoCTS is enabled (active), CN50XX stops transmitting whenever the UART*n*_CTS_L is inactive (high). This prevents overflowing the FIFO of the receiving UART. If UART*n*_CTS_L is not made inactive before the middle of the last stop bit, CN50XX transmits another character before stopping. While CN50XX is not transmitting, the transmitter FIFO can still be written, and even overflowed.

When autoCTS is enabled, the cores can poll any of the following before each write to avoid transmitter FIFO overflow:

- The cores can poll MIO_UART*n*_UART[TFNF] to check if the transmit FIFO is full.
- The cores can poll MIO_UART*n*_TFL[RFL] for the current transmit FIFO level.
- The cores can poll MIO_UART*n*_LSR[THRE] when programmable THRE Interrupt mode is enabled. MIO_UART*n*_LSR[THRE] indicates the transmit FIFO full status in that case. See Section 16.3.3 for more details on programmable THRE interrupt mode.

When UART*n*_CTS_L becomes active (low) again after being inactive (high), transmission resumes. Figure BBB is a timing diagram showing autoCTS operation.

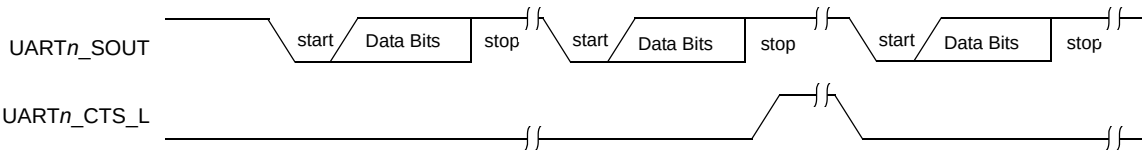


Figure 16–4 AutoCTS Timing

When autoCTS is disabled, the transmitter is unaffected by UART*n*_CTS_L.

16.3.3 UART Programmable THRE Interrupt

System performance can be improved with the UART's programmable THRE Interrupt mode. Programmable THRE Interrupt mode is enabled when:

- MIO_UART_IER[PTIME] is set.
- The FIFOs are enabled (i.e. MIO_UART*n*_FCR[EN] is set).

When programmable THRE interrupt mode is enabled, a THRE interrupt is active when the FIFO size is below the threshold selected by MIO_UART*n*_FCR[TXTRIG]. The available empty thresholds are:

- empty
- 2
- ¼
- ½.

When programmable THRE interrupt mode is enabled, the MIO_UART*n*_LSR[THRE] bit also switches its function. Instead of asserting when the transmitter FIFO is empty, it asserts when the FIFO is full, which allows software to fill the FIFO each transmit sequence by polling MIO_UART*n*_LSR[THRE] before writing another character. Proper threshold value selection may both maintain constant character transmission and minimize interruptions.

When programmable THRE interrupt mode is not enabled, THRE interrupts occur when the transmit FIFO is empty, and MIO_UART*n*_LSR[THRE] asserts on an empty transmit FIFO.

16.4 UART Registers

The UART registers are listed in Table 16–1.

Table 16–1 UART Registers

Register	Address	CSR Type[1]	Detailed Description
MIO_UART0_RBR	0x0001180000000800	RSL	See page 608
MIO_UART0_IER	0x0001180000000808	RSL	See page 609
MIO_UART0_IIR	0x0001180000000810	RSL	See page 610
MIO_UART0_LCR	0x0001180000000818	RSL	See page 611
MIO_UART0_MCR	0x0001180000000820	RSL	See page 612
MIO_UART0_LSR	0x0001180000000828	RSL	See page 614
MIO_UART0_MSR	0x0001180000000830	RSL	See page 615
MIO_UART0_SCR	0x0001180000000838	RSL	See page 615
MIO_UART0_THR	0x0001180000000840	RSL	See page 616
MIO_UART0_FCR	0x0001180000000850	RSL	See page 617
MIO_UART0_DLL	0x0001180000000880	RSL	See page 618
MIO_UART0_DLH	0x0001180000000888	RSL	See page 618
MIO_UART0_FAR	0x0001180000000920	RSL	See page 619
MIO_UART0_TFR	0x0001180000000928	RSL	See page 619
MIO_UART0_RFW	0x0001180000000930	RSL	See page 619
MIO_UART0_USR	0x0001180000000938	RSL	See page 620
MIO_UART0_TFL	0x0001180000000A00	RSL	See page 620
MIO_UART0_RFL	0x0001180000000A08	RSL	See page 620
MIO_UART0_SRR	0x0001180000000A10	RSL	See page 621
MIO_UART0_SRTS	0x0001180000000A18	RSL	See page 621
MIO_UART0_SBCR	0x0001180000000A20	RSL	See page 621
MIO_UART0_SFE	0x0001180000000A30	RSL	See page 621
MIO_UART0_SRT	0x0001180000000A38	RSL	See page 622
MIO_UART0_STT	0x0001180000000B00	RSL	See page 622
MIO_UART0_HTX	0x0001180000000B08	RSL	See page 622
MIO_UART1_RBR	0x0001180000000C00	RSL	See page 608
MIO_UART1_IER	0x0001180000000C08	RSL	See page 609
MIO_UART1_IIR	0x0001180000000C10	RSL	See page 610
MIO_UART1_LCR	0x0001180000000C18	RSL	See page 611
MIO_UART1_MCR	0x0001180000000C20	RSL	See page 612
MIO_UART1_LSR	0x0001180000000C28	RSL	See page 614
MIO_UART1_MSR	0x0001180000000C30	RSL	See page 615
MIO_UART1_SCR	0x0001180000000C38	RSL	See page 615
MIO_UART1_THR	0x0001180000000C40	RSL	See page 616
MIO_UART1_FCR	0x0001180000000C50	RSL	See page 617
MIO_UART1_DLL	0x0001180000000C80	RSL	See page 618
MIO_UART1_DLH	0x0001180000000C88	RSL	See page 618
MIO_UART1_FAR	0x0001180000000D20	RSL	See page 619
MIO_UART1_TFR	0x0001180000000D28	RSL	See page 619

Table 16–1 UART Registers (Continued)

Register	Address	CSR Type[1]	Detailed Description
MIO_UART1_RFW	0x0001180000000D30	RSL	See page 619
MIO_UART1_USR	0x0001180000000D38	RSL	See page 620
MIO_UART1_TFL	0x0001180000000E00	RSL	See page 620
MIO_UART1_RFL	0x0001180000000E08	RSL	See page 620
MIO_UART1_SRR	0x0001180000000E10	RSL	See page 621
MIO_UART1_SCTS	0x0001180000000E18	RSL	See page 621
MIO_UART1_SBCR	0x0001180000000E20	RSL	See page 621
MIO_UART1_SFE	0x0001180000000E30	RSL	See page 621
MIO_UART1_SRT	0x0001180000000E38	RSL	See page 622
MIO_UART1_STT	0x0001180000000F00	RSL	See page 622
MIO_UART1_HTX	0x0001180000000F08	RSL	See page 622

1. RSL-type registers are accessed indirectly across the I/O Bus.

MIO UART0/1 Receive Buffer Register
MIO_UART0/1 _RBR

The receive buffer register is a read-only register that contains the data byte received on the serial input port (SIN). The data in this register is valid only if the MIO_UART*n*_LSR[DR] bit is set (See the "MIO UART0/1 Line Status Register" on page 614). See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved
<7:0>	RBR	RO	0x0	—	Receive buffer register. Contains the data byte received on the serial input port

- When the FIFOs are programmed ON, this register accesses the head of the receive FIFO.
- When the FIFOs are programmed OFF, the data in the RBR must be read before the next data arrives, otherwise it is overwritten, resulting in an overrun error.
- If the receive FIFO is full (64 characters) and this register is not read before the next data character arrives, then the data already in the FIFO is preserved, but any incoming data is lost. An overrun error also occurs.

NOTE: MIO_UART*n*_LCR[DLAB] must be clear to access this register.

The address in Table 16–1 is an alias to simplify these CSR descriptions. The MIO_UART*n*_RBR, MIO_UART*n*_THR, and MIO_UART*n*_DLL registers are one register with different uses at different times.

MIO UART0/1 Interrupt-Enable Register MIO_UART0/1 _IER

The interrupt-enable register is a read/write register that contains four bits that enable the generation of interrupts:

- enable received data available interrupt (ERBFI)
- enable transmitter holding register empty interrupt (ETBEI)
- enable receiver line status interrupt (ELSI)
- enable modem status interrupt (EDSSI).

The IER also contains the enable bit for the programmable transmit holding register empty (THRE) interrupt mode (PTIME). See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved
<7>	PTIME	R/W	0	—	Programmable THRE interrupt mode enable
<6:4>	—	R/W	0x0	—	Reserved
<3>	EDSSI	R/W	0	—	Enable modem status interrupt
<2>	ELSI	R/W	0	—	Enable receiver line status interrupt
<1>	ETBEI	R/W	0	—	Enable transmitter holding register empty interrupt
<0>	ERBFI	R/W	0	—	Enable received data available interrupt

NOTE: MIO_UART*n*_LCR[DLAB] must be clear to access this register.

The address in Table 16–1 is an alias to simplify these CSR descriptions. The MIO_UART*n*_IER and MIO_UART*n*_DLH registers are one register with different uses at different times.

MIO UART0/1 Interrupt Identity Register MIO_UART0/1 _IIR

The interrupt identity register is a read-only register that identifies the source of an interrupt. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved
<7:6>	FEN	RO	0x0	—	FIFO-enabled. 00 = FIFOs disabled, 01 = reserved, 10 = reserved, 11 = FIFOs enabled
<5:4>	—	RAZ	0x0	—	Reserved
<3:0>	IID	RO	0x1	—	Interrupt ID. Identifies the highest priority pending interrupt. The interrupt-source decoding, interrupt priority, and interrupt-reset control are shown in the following table:

IID	Priority Level	Interrupt Type	Interrupt Source	Interrupt reset by:
0001	—	None	None	—
0110	highest	Receiver line status	Overrun, parity, or framing errors, or break interrupt	Reading the line-status register.
0100	second	Received data available	Receiver data available (FIFOs disabled) or RX FIFO trigger level reached (FIFOs enabled)	Reading the receiver buffer register (FIFOs disabled) or the FIFO drops below the trigger level (FIFOs enabled)
1100	second	Character Timeout Indication	No characters in or out of the RX FIFO during the last four character times and there is at least one character in it during this time.	Reading the receiver buffer register.
0010	third	Transmitter holding register empty	Transmitter holding register empty (programmable THRE mode disabled) or TX FIFO at or below threshold (programmable THRE mode enabled)	Reading the interrupt identity register (if it is the source of the interrupt) or writing into MIO_UART*n*_THR (FIFOs or THRE mode disabled) or TX FIFO above threshold (FIFOs or THRE mode enabled)
0000	fourth	Modem status changed	Clear to send (CTS),[1] data set ready (DSR), ring indicator (RI), or data carrier detect (DCD) changed.	Reading the modem status register.
0111	fifth	Busy detect indication	Software has tried to write the line control register while the busy bit of the UART status register was set.	Reading the UART status register.

1. A change in CTS causes an interrupt only when autoflow-control mode is disabled.

NOTE: The address in Table 16–1 is an alias to simplify these CSR descriptions. The MIO_UART*n*_FCR and MIO_UART*n*_IIR registers are one register with different uses at different times.

MIO UART0/1 Line Control Register
MIO_UART0/1 _LCR

The line control register controls the format of the data that is transmitted and received by the UART. It is always readable, but it is writeable only when the UART is not busy (i.e. when MIO_UART*n*_USR[BUSY] = 0. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description[1]
<63:8>	—	RAZ	—	—	Reserved
<7>	DLAB	R/W	0	—	Divisor latch address bit. Setting this bit enables reading and writing of the divisor latch register (MIO_UART*n*_DLL and MIO_UART*n*_DLH) to set the baud rate of the UART. This bit must be cleared after initial baud-rate setup in order to access other registers.
<6>	BREAK	R/W	0	—	Break control bit. Setting this bit when not in loopback mode (i.e. MIO_UART*n*_MCR[LOOP] = 0) sends a break signal by holding the UART*n*_SOUT line low. When in loopback mode, the break condition is internally looped back to the receiver.
<5>	—	R/W	0	—	Reserved.
<4>	EPS	R/W	0	—	Even parity select bit. Selects between even and odd parity. 1 = an even number of ones is transmitted or checked 0 = an odd number of ones is transmitted or checked
<3>	PEN	R/W	0	—	Parity enable bit. Enables and disables parity generation and detection in transmitted and received serial character respectively.
<2>	STOP	R/W	0	—	Stop control bit. Controls the number of stop bits transmitted. 0 = one stop bit is transmitted in the serial data. 1 = two stop bits are generated and transmitted in the serial data out, UNLESS CLS = 00, then one and a half stop bits are generated. Note that, regardless of the number of stop bits selected, the receiver only checks the first stop bit.
<1:0>	CLS	R/W	0x0	—	Character length select field. Selects the number of data bits per character that are transmitted and received. 00 = 5 bits (bits 0-4 sent) 01 = 6 bits (bits 0-5 sent) 10 = 7 bits (bits 0-6 sent) 11 = 8 bits (all bits sent)

1. The LCR is writable only when the UART is not busy (i.e. when MIO_UART*n*_USR[BUSY] = 0). The LCR is always readable.

MIO UART0/1 Modem-Control Register MIO_UART0/1 _MCR

The lower four bits of the modem-control register directly manipulate the outputs of the UART. The DTR, RTS, OUT1, and OUT2 bits are inverted and then drive the corresponding UART outputs: dtr_n, rts_n (UART*n*_RTS_L), out1_n, and out2_n.

NOTE: The dtr_n, out1_n, and out2_n outputs are not present on the pins of CN50XX, but the DTR, OUT1, and OUT2 bits still function in loopback mode.

See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved
<7:6>	—	R/W	0x0	—	Reserved
<5>	AFCE	R/W	0	—	Autoflow-control enable (AFCE) bit. When FIFOs are enabled and this bit is set, 16750-compatible autoRTS and autoCTS serial data flow control features are enabled.
<4>	LOOP	R/W	0	—	Loopback bit. When set, data on the UART*n*_SOUT line is held high, while serial data output is looped back to the UART*n*_SIN line, internally. In this mode all the interrupts are fully functional. This feature is used for diagnostic purposes. Also, in loopback mode, the modem-control inputs (dsr_n, cts_n, ri_n, dcd_n) are disconnected and the four modem-control outputs (dtr_n, rts_n, out1_n, out1_n) are looped back to the inputs, internally
<3>	OUT2	R/W	0	—	OUT2 output bit. This bit is inverted and drives the out2_n signal
<2>	OUT1	R/W	0	—	OUT1 output bit. This bit is inverted and drives the out1_n signal
<1>	RTS	R/W	0	—	RTS output bit. This bit is inverted and drives the rts_n signal.[1]
<0>	DTR	R/W	0	—	DTR output bit. This bit is inverted and drives the dtr_n signal.

1. When autoRTS is enabled, the rts_n output is controlled in the same way, but is also gated with the receiver FIFO threshold trigger (rts_n is inactive high when above the threshold). The rts_n output is deasserted whenever RTS = 0.

AutoRTS becomes active when the following occurs:

1. MIO_UART*n*_MCR[RTS] is set.
2. FIFOs are enabled by setting MIO_UART*n*_FCR[EN].
3. MIO_UART*n*_MCR[AFCE] is set (must be set after MIO_UART*n*_FCR[EN])

When autoRTS is active, the rts_n output is forced inactive high when the receiver FIFO level reaches the threshold set by MIO_UART*n*_FCR[RXTRIG]. When rts_n is connected to the cts_n input of another UART device, the other UART stops sending serial data until the receiver FIFO has available space.

The selectable receiver FIFO threshold values are:

- 1
- ¼
- ½
- 2-less-than-full.

Since one additional character may be transmitted to the UART after rts_n has become inactive (due to data already having entered the transmitter block in the other UART), setting the threshold to 2-less-than-full allows maximum use of the FIFO with a safety zone of one character.

Once the receiver FIFO becomes completely empty by reading the MIO_UART*n*_RBR, rts_n again becomes active low, signalling the other UART to continue sending data.

It is important to note that, even if everything else is set to enabled and the correct MIO_UART*n*_MCR bits are set: if the FIFOs are disabled through MIO_UART*n*_FCR[EN], autoflow control is also disabled. When autoRTS is disabled or inactive, rts_n is controlled solely by MIO_UART*n*_MCR[RTS].

AutoCTS becomes active when the following occurs:

1. FIFOs are enabled by setting MIO_UART*n*_FCR[EN].
2. MIO_UART*n*_MCR[AFCE] is set (must be set after MIO_UART*n*_FCR[EN])

When active, the UART transmitter is disabled whenever the cts_n input becomes inactive high. This prevents overflowing the FIFO of the receiving UART.

Note that if the cts_n input is not inactivated before the middle of the last stop bit, another character is transmitted before the transmitter is disabled. While the transmitter is disabled, the transmitter FIFO can still be written to, and even overflowed. Therefore, when using this mode, either the true FIFO depth (64 characters) must be known to software, or the programmable THRE interrupt mode must be enabled to access the FIFO-full status through the MIO_UART*n*_LSR. When using the FIFO-full status, software can poll this before each write to the transmitter FIFO.

NOTE: FIFO-full status is also available in the MIO_UART*n*_USR register, or the actual level of the FIFO may be read through the MIO_UART*n*_TFL register.

When the cts_n input becomes active low again, transmission resumes. It is important to note that, even if everything else is set to enabled, autoflow control is also disabled if the FIFOs are disabled through MIO_UART*n*_FCR[EN]. When autoCTS is disabled or inactive, the transmitter is unaffected by cts_n.

MIO UART0/1 Line Status Register MIO_UART0/1 _LSR

The line status register contains status of the receiver and transmitter data transfers. This status can be read by the user at anytime.

See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved.
<7>	FERR	RC	0	—	Error in receiver FIFO bit. This bit is active only when FIFOs are enabled. It is set when there is at least one parity error, framing error, or break indication in the FIFO. This bit is cleared when MIO_UART*n*_LSR is read and the character with the error is at the top of the receiver FIFO and there are no subsequent errors in the FIFO.
<6>	TEMT	RO	1	—	Transmitter empty bit. In FIFO mode, this bit is set whenever MIO_UART*n*_TSR and the FIFO are both empty. In nonFIFO mode, this bit is set whenever MIO_UART*n*_THR and MIO_UART*n*_TSR are both empty. This bit is typically used to make sure it is safe to change control registers. Changing control registers while the transmitter is busy can result in corrupt data being transmitted.
<5>	THRE	RO	1	—	Transmitter holding register empty bit. When programmable THRE interrupt mode is disabled, this bit indicates that the UART can accept a new character for transmission. This bit is set whenever data is transferred from MIO_UART*n*_THR to the transmitter shift register and no new data has been written to MIO_UART*n*_THR. This also causes a THRE interrupt to occur, if the THRE interrupt is enabled. When FIFOs and programmable THRE Interrupt mode are enabled, this bit's functionality is switched to indicate the transmitter FIFO is full, and no longer controls THRE Interrupts, which are then controlled by the MIO_UART*n*_FCR[TXTRIG] threshold setting.
<4>	BI	RC	0	—	Break interrupt bit.
<3>	FE[1]	RC	0	—	Framing error bit. This bit is set whenever there is a framing error in the receiver. A framing error occurs when the receiver does not detect a valid STOP bit in the received data. In FIFO mode, since the framing error is associated with a character received, it is revealed when the character with the framing error is at the top of the FIFO
<2>	PE[1]	RC	0	—	Parity error bit. This bit is set whenever there is a parity error in the receiver if the parity-enable bit (MIO_UART*n*_LCR[PEN]) is set. In FIFO mode, since the parity error is associated with a character received, it is revealed when the character with the parity error arrives at the top of the FIFO.
<1>	OE[1]	RC	0	—	Overrun error bit. When set, this bit indicates an overrun error has occurred because a new data character was received before the previous data was read. In nonFIFO mode, this bit is set when a new character arrives in the receiver before the previous character was read from MIO_UART*n*_RBR. When this happens, the data in MIO_UART*n*_RBR is overwritten. In FIFO mode, an overrun error occurs when the FIFO is full and a new character arrives at the receiver. The data in the FIFO is retained and the data in the receive shift register is lost.
<0>	DR	RO	0	—	Data ready bit. When set, this bit indicates the receiver contains at least one character in the MIO_UART*n*_RBR or the receiver FIFO. This bit is cleared when the MIO_UART*n*_RBR is read in nonFIFO mode, or when the receiver FIFO is empty in FIFO mode.

1. The OE, PE, and FE bits are reset when a read of the MIO_UART*n*_LSR is performed.

MIO UART0/1 Modem Status Register MIO_UART0/1 _MSR

The modem status register contains the current status of the modem-control input lines and if they changed. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
63:8>	—	RAZ	—	—	Reserved.
<7>	DCD	RO	0	—	DCD input bit. This bit is the complement of the modem-control line dcd_n. In loopback mode, this bit is the same as MIO_UART*n*_MCR[OUT2].
<6>	RI	RO	0	—	RI input bit. This bit is the complement of the modem-control line ri_n. In loopback mode, this bit is the same as MIO_UART*n*_MCR[OUT1].
<5>	DSR	RO	0	—	DSR input bit. This bit is the complement of the modem-control line dsr_n. In loopback mode, this bit is the same as MIO_UART*n*_MCR[DTR].
<4>	CTS	RO	—	—	CTS input bit. This bit is the complement of the modem-control line cts_n. In loopback mode, this bit is the same as MIO_UART*n*_MCR[RTS].
<3>	DDCD	RC	0	—	DCD change bit. Records whether the dcd_n line has changed since the last time the user read the MIO_UART*n*_MSR. In loopback mode, reflects changes in MIO_UART*n*_MCR[OUT2].
<2>	TERI	RC	0	—	RI change bit. Indicates ri_n has changed from an active-low, to an inactive-high state since the last time the user read the MIO_UART*n*_MSR. In loopback mode, reflects state changes from high to low in MIO_UART*n*_MCR[OUT1].
<1>	DDSR	RC	0	—	DSR change bit. Records whether the dsr_n line has changed since the last time the user read the MIO_UART*n*_MSR. In loopback mode, reflects changes in MIO_UART*n*_MCR[DTR].
<0>	DCTS	RC	0	—	CTS change bit. Records whether the cts_n line has changed since the last time the user read the MIO_UART*n*_MSR. In loopback mode, reflects changes in MIO_UART*n*_MCR[RTS].

NOTE: The dsr_n, ri_n, and dcd_n inputs are internally tied to power and not present on the pins of CN50XX. Thus the DSR, RI, and DCD bits are 0 when not in loopback mode.

MIO UART0/1 Scratchpad Register MIO_UART0/1 _SCR

The scratchpad register is an 8-bit read/write register for programmers to use as a temporary storage space. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	NS	NS	Reserved
<7:0>	SCR	R/W	NS	NS	Scratchpad Register

MIO UART0/1 Transmit Holding Register
MIO_UART0/1 _THR

The transmit holding register is a write-only register that contains data to be transmitted on the serial output port (UART*n*_SOUT). Data can be written to MIO_UART*n*_THR any time that MIO_UART*n*_LSR[THRE] = 1. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	WO	—	—	Reserved
<7:0>	THR	WO	0x0	—	Transmit holding register.

- If FIFOs are not enabled and MIO_UART*n*_LSR[THRE] = 1, writing a single character to MIO_UART*n*_THR clears MIO_UART*n*_LSR[THRE]. Any additional writes to MIO_UART*n*_THR before MIO_UART*n*_LSR[THRE] is set again causes MIO_UART*n*_THR data to be overwritten.
- If FIFOs are enabled and MIO_UART*n*_LSR[THRE] = 1, and programmable THRE mode is disabled (i.e. MIO_UART*n*_IER[PTIME] = 0), 64 characters of data may be written to MIO_UART*n*_THR before the FIFO is full. Any attempt to write data when the FIFO is full results in the write data being lost.

NOTE: MIO_UART*n*_LCR[DLAB] must be clear to access this register.

The address in Table 16–1 is an alias to simplify these CSR descriptions. The MIO_UART*n*_THR, MIO_UART*n*_RBR, and MIO_UART*n*_DLL registers are one register with different uses at different times.

MIO UART0/1 FIFO Control Register MIO_UART0/1_FCR

The FIFO control register is a write-only register that controls the read- and write-data FIFO operation. When FIFOs and programmable-THRE-interrupt mode are enabled, this register also controls the THRE interrupt-empty threshold level. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	WO	—	—	Reserved.
<7:6>	RXTRIG	WO	0x0	—	RX trigger.If FIFOs are enabled (i.e. EN = 1), this field is active and set the trigger level in the receiver FIFO for the enable received-data-available interrupt (ERBFI). In autoflow-control mode, the trigger is used to determine when the rts_n signal will be deasserted. The trigger values are: 00 = 1 character in FIFO 01 = FIFO ¼ full 10 = FIFO ½ full 11 = FIFO is two characters less than full
<5:4>	TXTRIG	WO	0x0	—	TX trigger. If the FIFOs and programmable-THRE-interrupt mode are enabled, the values in this field control the empty threshold level at which THRE interrupts are generated when the mode is active. 00 = empty FIFO 01 = 2 characters in FIFO 10 = FIFO ¼ full 11 = FIFO ½ full
<3>	—	WO	0	—	Reserved.
<2>	TXFR	WO	0	—	TX FIFO reset. Writing a 1 to this bit resets and flushes data in the transmit FIFO. This bit is self-clearing, so it is not necessary to clear this bit.
<1>	RXFR	WO	0	—	RX FIFO reset. Writing a 1 to this bit resets and flushes data in the receive FIFO. This bit is self-clearing, so it is not necessary to clear this bit.
<0>	EN	WO	0	—	FIFO enable. Writing a 1 to this bit enables the transmit and receive FIFOs. Whenever the value of this bit is changed both the TX and RX FIFOs are reset.

NOTE: The address in Table 16–1 is an alias to simplify these CSR descriptions. The MIO_UART*n*_FCR and MIO_UART*n*_IIR registers are one register with different uses at different times.

MIO UART0/1 Divisor Latch Low Register MIO_UART0/1_DLL

The 8-bit divisor latch high register in conjunction with the 8-bit divisor latch low (MIO_UART*n*_DLL) register form a 16-bit, read/write, Divisor Latch register that contains the baud rate divisor for the UART. It is accessed by first setting MIO_UART*n*_LCR[DLAB] (bit 7) (refer to "MIO UART0/1 Line Control Register" on page 611). The output baud rate is equal to the ECLK frequency divided by sixteen times the value of the baud rate divisor, as follows:

baud rate = ECLK / (16 × divisor).

Note that once both divisor latch registers are set, at least 2 × divisor × 16 ECLK cycles should be allowed to pass before transmitting or receiving data.

See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved
<7:0>	DLL	R/W	0x0	—	Divisor latch low register

NOTE: The address in Table 16–1 is an alias to simplify these CSR descriptions. The MIO_UART*n*_DLL, MIO_UART*n*_RBR, and MIO_UART*n*_THR registers are one register with different uses at different times.

MIO UART0/1 Divisor Latch High Register MIO_UART0/1_DLH

The 8-bit divisor latch high register in conjunction with the 8-bit divisor latch low (MIO_UART*n*_DLL) register form a 16-bit, read/write, Divisor Latch register that contains the baud rate divisor for the UART. It is accessed by first setting MIO_UART*n*_LCR[DLAB] (bit 7) (refer to "MIO UART0/1 Line Control Register" on page 611). The output baud rate is equal to the ECLK frequency divided by sixteen times the value of the baud rate divisor, as follows:

baud rate = ECLK / (16 × divisor).

Note that once both divisor latch registers are set, at least 2 × divisor × 16 ECLK cycles should be allowed to pass before transmitting or receiving data.

See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved
<7:0>	DLH	R/W	0x0	—	Divisor latch high register

NOTE: The address in Table 16–1 is an alias to simplify these CSR descriptions. The MIO_UART*n*_IER and MIO_UART*n*_DLH registers are one register with different uses at different times.

MIO UART0/1 FIFO Access Register MIO_UART0/1_FAR

The FIFO access register is used to enable a FIFO-access mode for testing, so that the receive FIFO can be written by software and the transmit FIFO can be read by software when the FIFOs are enabled. When FIFOs are not enabled it allows the MIO_UART*n*_RBR to be written by software and the MIO_UART*n*_THR to be read by software. Note, that when the FIFO-access mode is enabled/disabled, the control portion of the receive FIFO and transmit FIFO is reset and the FIFOs are treated as empty. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	FAR	R/W	0	—	FIFO-access mode enable.

MIO UART0/1 Transmit FIFO Read Register MIO_UART0/1_TFR

The transmit FIFO read register is only valid when FIFO-access mode is enabled (i.e. MIO_UART*n*_FAR[FAR] = 1). See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved.
<7:0>	TFR	RO	0x0	—	Transmit FIFO data. When FIFOs are enabled, this field gives the data at the top of the transmit FIFO. Each consecutive read pops the transmit FIFO and gives the next data value that is currently at the top of the FIFO. When FIFOs are not enabled, this field gives the data in MIO_UART*n*_THR.

MIO UART0/1 Receive FIFO Write Register MIO_UART0/1_RFW

The receive FIFO write register is only valid when FIFO-access mode is enabled (i.e. MIO_UART*n*_FAR[FAR] = 1). See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:10>	—	RAZ	—	—	Reserved.
<9>	RFFE	WO	0	—	Receive FIFO framing error.
<8>	RFPE	WO	0	—	Receive FIFO parity error.
<7:0>	RFWD	WO	0x0	—	Receive FIFO write data. When FIFOs are enabled, this field provides write data to the receive FIFO. Each consecutive write pushes the new data to the next write location in the receive FIFO. When FIFOs are not enabled, this field provides write data to the MIO_UART*n*_RBR.

MIO UART0/1 UART Status Register
MIO_UART0/1_USR

The receive FIFO write register is only valid when FIFO-access mode is enabled (i.e. MIO_UART*n*_FAR[FAR] = 1). See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:5>	—	RAZ	—	—	Reserved.
<4>	RFF	RO	0	—	RX FIFO full.
<3>	RFNE	RO	0	—	RX FIFO not empty.
<2>	TFE	RO	1	—	TX FIFO empty.
<1>	TFNF	RO	1	—	TX FIFO not full.
<0>	BUSY	RO	0x0	—	Busy. When set,this bit indicates that a serial transfer is in progress; when clear, it indicates that the UART is idle or inactive.

MIO UART0/1 Transmit FIFO Level Register
MIO_UART0/1_TFL

The transmit FIFO level register indicates the number of data entries in the transmit FIFO. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:7>	—	RAZ	—	—	Reserved.
<6:0>	TFL	RO	0x0	—	Transmit FIFO level. Indicates the number of data entries in the transmit FIFO

MIO UART0/1 Receive FIFO Level Register
MIO_UART0/1_RFL

The receive FIFO level register indicates the number of data entries in the receive FIFO. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:7>	—	RAZ	—	—	Reserved.
<6:0>	RFL	RO	0x0	—	Receive FIFO level. Indicates the number of data entries in the receive FIFO

MIO UART0/1 Software Reset Register
MIO_UART0/1_SRR

The software reset register is a write-only register that resets the UART and/or the receive FIFO and/or the transmit FIFO. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:3>	—	RAZ	—	—	Reserved.
<2>	STFR	WO	0	—	Shadow copy of the TX FIFO reset bit (MIO_UART*n*_FCR[TXFR]). This bit can be used to remove the burden on software having to store previously written FCR values (which are pretty static) just to reset the transmit FIFO
<1>	SRFR	WO	0	—	Shadow copy of the RX FIFO reset bit (MIO_UART*n*_FCR[RXFR]). This can be used to remove the burden on software having to store previously written FCR values (which are pretty static) just to reset the receive FIFO.
<0>	USR	WO	0	—	UART soft reset. Setting this bit resets the UART.

MIO UART0/1 Shadow Request to Send Register
MIO_UART0/1_SRTS

The shadow request to send register is a shadow register for the MIO_UART*n*_MCR[RTS] bit that can be used to remove the burden of having to perform a read-modify-write on MIO_UART*n*_MCR. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	SRTS	R/W	0	—	Shadow request to send.

MIO UART0/1 Shadow Break Control Register
MIO_UART0/1_SBCR

The shadow break control register is a shadow register for the MIO_UART*n*_LCR[BREAK] bit that can be used to remove the burden of having to perform a read-modify-write on MIO_UART*n*_LCR. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	SBCR	R/W	0	—	Shadow break control.

MIO UART0/1 Shadow FIFO Enable Register
MIO_UART0/1_SFE

The shadow FIFO enable register is a shadow register for MIO_UART*n*_FCR[EN] that can be used to remove the burden of having to store the previously written value to MIO_UART*n*_FCR in memory and having to mask this value so that only the FIFO enable bit gets updated. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	SFE	R/W	0	—	Shadow FIFO enable.

MIO UART0/1 Shadow RX Trigger Register

MIO_UART0/1_SRT

The shadow RX trigger register is a shadow register for the RX trigger bits (MIO_UART*n*_FCR[RXTRIG]) that can be used to remove the burden of having to store the previously written value to MIO_UART*n*_FCR in memory and having to mask this value so that only the RX trigger bits get updated. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1:0>	SRT	R/W	0x0	—	Shadow RX trigger.

MIO UART0/1 Shadow TX Trigger Register
MIO_UART0/1_STT

The shadow TX trigger register is a shadow register for the TX trigger bits (MIO_UART*n*_FCR[TXTRIG]) that can be used to remove the burden of having to store the previously written value to MIO_UART*n*_FCR in memory and having to mask this value so that only the TX trigger bits get updated. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved.
<1:0>	STT	R/W	0x0	—	Shadow TX trigger.

MIO UART0/1 Halt TX Register
MIO_UART0/1_HTX

The halt TX register is used to halt transmissions for testing, so that the transmit FIFO can be filled by software when FIFOs are enabled. If FIFOs are not enabled, setting the HTX register will have no effect. See Table 16–1 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	HTX	R/W	0	—	Halt TX.

Chapter 17

TWSI Interface

This chapter contains the following subjects:

Overview

The CN50XX two-wire serial interface (TWSI) supports a standard suite of features, including the following:

- Standard two-wire SCL/SDA protocols.
- Standard mode (100 Kbps) and fast mode (400 Kbps) support.
- 7-bit and 10-bit addressing.
- Master and slave support.
  - TWSI masters initiate operations on the TWSI interface
  - TWSI slaves only respond to operations initiated by TWSI masters
- Multi-master support.
- Clock stretching.
- Flexible bus reset support. (Software can drive arbitrary patterns on SCL/SDA to create any required reset sequence.)

Figure 17–1 shows the architecture of CN50XX's TWSI logic. It consists of a TWSI core that implements some low-level details of the protocol, together with a high-level controller (HLC) to implement various multibyte sequences. TWSI transactions may be generated by either directing the HLC or directing the low-level details of the TWSI core. It is possible for either cores or a remote PCI host to communicate with a TWSI device, though we assume only the cores for the remainder of this section. The TWSI HLC is enabled when TWSI_CTL[CE] is set (see Section 17.4.4).

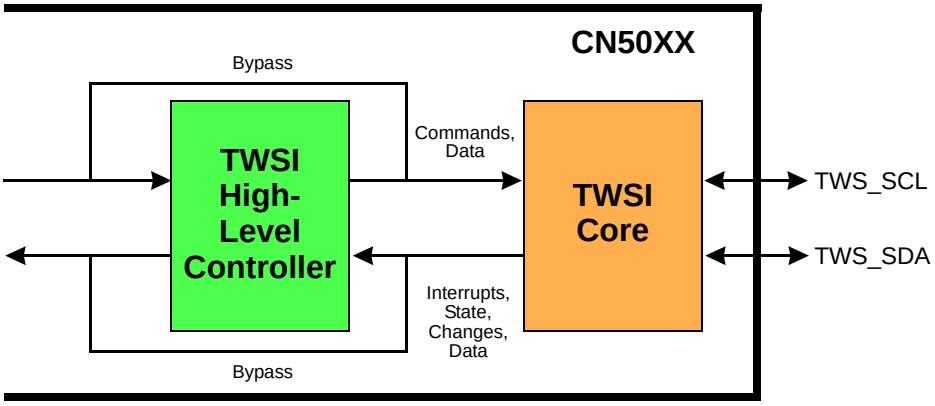


Figure 17–1 TWSI Architecture

Figure 17–2 displays the steps of the sequences supported by the TWSI HLC. The initial address identifies the device that will be the TWSI slave. CN50XX supports both 7-bit and 10-bit initial addresses. The address extension carries an optional internal address for the identified slave, and also completes the initial address for a read with a 10-bit address. CN50XX optionally supports an internal address of 8 or 16 bits. Finally, the read bytes or write bytes complete the sequence. CN50XX supports between zero- and eight-byte read/write sequences.

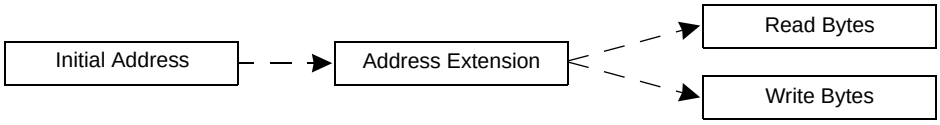


Figure 17–2 Supported TWS HLC Transaction Steps

CN50XX can communicate with another TWSI device as either master or slave. The steps in Figure 17–2 apply in both cases, though the details of each differ.

17.1 High-Level Controller as a Master

The CN50XX cores become master of HLC TWSI sequences by writing the MIO_TWS_SW_TWSI register. The HLC is in master mode when MIO_TWS_SW_TWSI[SLONLY] = 0 and both the TWSI_CTL[CE, AAK] bits are set to 1 (see Section 17.4.4).

- If the sequence is a write, the MIO_TWS_SW_TWSI[D] field (and MIO_TWS_SW_TWSI_EXT[D] if the write sequence is more than four bytes) contains the write data.
- If the sequence is a read, the MIO_TWS_SW_TWSI[D] field (and MIO_TWS_SW_TWSI_EXT[D] if the read sequence is more than four bytes) contains the read result after the HLC completes the transaction.

The cores either poll MIO_TWS_SW_TWSI or use interrupts to determine when the sequence is complete. CN50XX clears the MIO_TWS_SW_TWSI[V] bit and sets the MIO_TWS_INT[ST_INT] bit when the HLC completes the transaction.

If the mastered high-level sequence has an error, the MIO_TWS_SW_TWSI[R] bit is set to 0 and MIO_TWS_SW_TWSI[D<7:0>] contains an error code. In the case of an error, the TWSI high-level state controller aborts the operation on the TWSI bus and resets its internal state.

Refer to the "TWSI Status Register" on page 643. The error code is an unexpected TWSI core state found by the HLC.

- An example of an error would be **a NACK where an ACK was expected** by the HLC (0x20, 0x30, 0x48, 0xD8).
- Another example is where the **HLC loses arbitration** (0x38, 0x68, 0xB0, 0x78).
- A final error example is if the **master-mode operation attempts to initiate while the TWSI HLC is servicing a TWSI operation as a slave** (0x80, 0x88, 0xA0, 0xA8, 0xB8, 0xC0, 0xC8).

Figures 17–4 through 17–10 show some of the CN50XX-TWSI sequences (as defined in Figure 17–2). Figure 17–3 provides a legend for the symbols used in these examples.

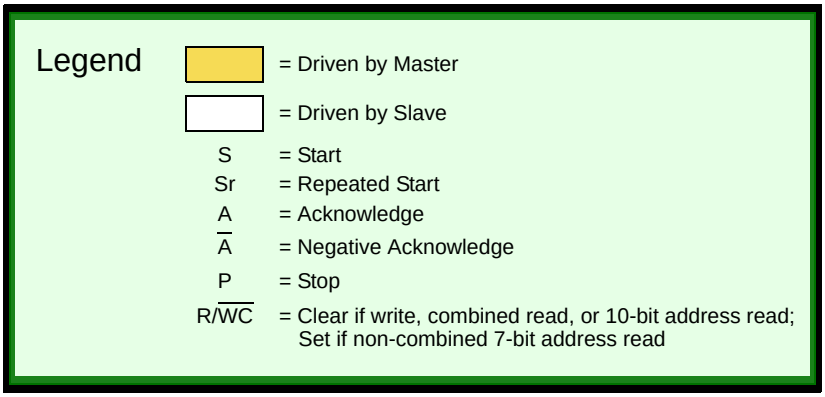
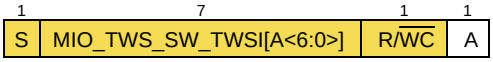


Figure 17–3 Legend for Diagrams

Figure 17–4 shows the possible initial-address steps for sequences mastered by the HLC. MIO_TWS_SW_TWSI[OP<1>] selects whether a 7-bit or 10-bit address is used, and MIO_TWS_SW_TWSI[A] contains the address of the remote slave.

MIO_TWS_SW_TWSI Write Operations, with [SLONLY] = 0, [OP<2>] = 0

7-bit address ([OP<1>] = 0)



10-bit Address ([OP<1>] = 1)

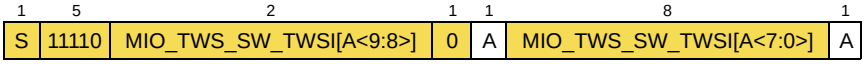


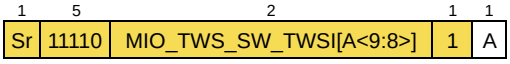
Figure 17–4 HLC as Master, Initial-Address Step

All possible address-extension steps for sequences mastered by the HLC are shown in Figure 17–5.

MIO_TWS_SW_TWSI Write Operations, with [SLONLY] = 0, [OP<2>] = 0

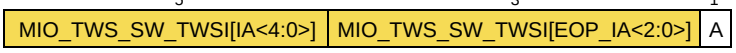
7-bit Non-Combined read ([OP<1:0>] = 00, [R] = 1)
Write without IA ([OP<0>] = 0, [R] = 0) None

10-bit Non-Combined Read ([OP<1:0>] = 2, [R] = 1)

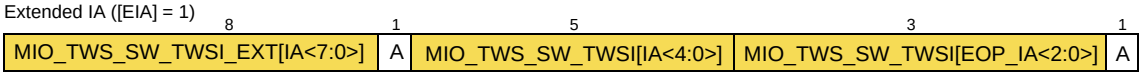


Write with IA ([OP<0>] = 1, [R] = 0)

[EIA] = 0



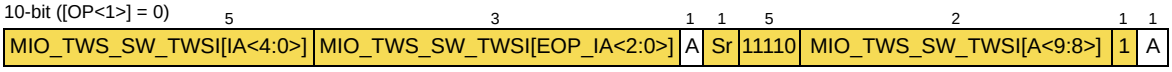
Extended IA ([EIA] = 1)



Combined Read ([OP<0>] = 1, [R] = 1, [EIA] = 0)

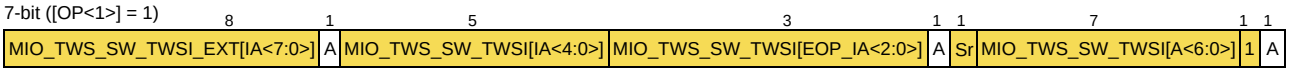
7-bit ([OP<1>] = 1)

10-bit ([OP<1>] = 0)



Combined Read with Extended IA ([OP<0>] = 1, [R] = 1, [EIA] = 1)

7-bit ([OP<1>] = 1)



10-bit ([OP<1>] = 0)

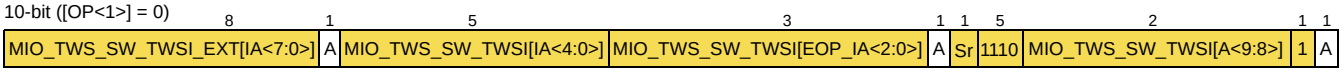


Figure 17–5 HLC as Master, Address-Extension Step

MIO_TWS_SW_TWSI[IA,EOP_IA] and MIO_TWS_SW_TWSI_EXT[IA] contain the internal address to be used by the external slave. The address extension component is empty for 7-bit address non-combined reads and for writes without internal addresses.

Figure 17–6 shows all possible read-bytes steps for sequences mastered by the HLC, while Figure 17–7 shows all possible write-bytes steps for sequences mastered by the HLC. CN50XX can master any size from one to eight bytes.

17.2 High-Level Controller as a Slave

The CN50XX cores can act as a slave to communicate with another TWSI device via the MIO_TWS_SW_TWSI and MIO_TWS_TWSI_SW registers and the TWSI HLC. The HLC is in slave mode when MIO_TWS_SW_TWSI[SLONLY] is set to 1 and both TWSI_CTL[CE,AAK] are set to 1.

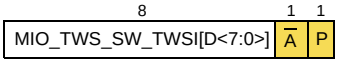
- To transfer up to four bytes from a core to the TWSI device, with the HLC acting as a slave:
  - The core writes the bytes into MIO_TWS_SW_TWSI, setting the MIO_TWS_SW_TWSI[V] bit.
  - The remote TWSI device reads MIO_TWS_SW_TWSI, retrieving the data and its corresponding vbyte. Vbyte<7> is set if MIO_TWS_SW_TWSI[V] is set, and the read clears MIO_TWS_SW_TWSI[V].
  - The core may either poll MIO_TWS_SW_TWSI[V] or wait for an interrupt (MIO_TWS_INT[ST_INT]) to determine when the data has been received, and when more data can be written into MIO_TWS_SW_TWSI.
- To transfer up to four bytes from a TWSI device to a core, with the HLC acting as a slave:
  - The remote TWSI device writes MIO_TWS_TWSI_SW, setting MIO_TWS_TWSI_SW[V].
  - The core either receives an interrupt (via MIO_TWS_INT[TS_INT]) or polls MIO_TWS_TWSI_SW[V] to determine when the bytes are present.
  - The core reads MIO_TWS_TWSI_SW, clearing MIO_TWS_TWSI_SW[V].
  - The remote TWSI device polls MIO_TWS_TWSI_SW to determine when it can write it again. Vbyte<7> is clear when MIO_TWS_TWSI_SW[V] is clear.

Figure 17–8 shows the possible initial-address steps (as shown in Figure 17–2) supported for sequences with the HLC as a slave. The TWSI core TWSI slave address register and TWSI slave extended address register contain the initial address that CN50XX will respond to.

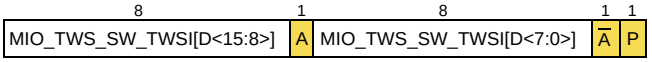
Figure 17–9 shows the possible address-extension steps supported for sequences with the HLC as a slave. Table 17–1 provides the format of CN50XX's slave internal-address (SIA) register. SIA selects between the MIO_TWS_SW_TWSI and MIO_TWS_TWSI_SW registers. The external TWSI device must use the SIA register.

MIO_TWS_SW_TWSI Read Operations, with [SLONLY] = 0, [OP<2>] = 0, [R] = 1

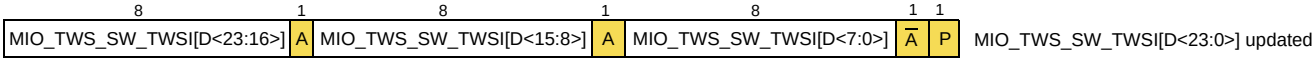
One-Byte Read, ([SOVR] = 0, [OP<3>] = 0) or ([SOVR] = 1, [SIZE] = 0)



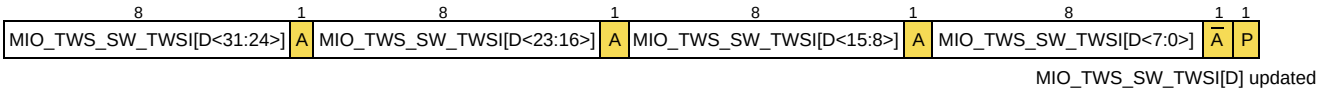
Two-Byte Read, ([SOVR] = 1, [SIZE] = 1)



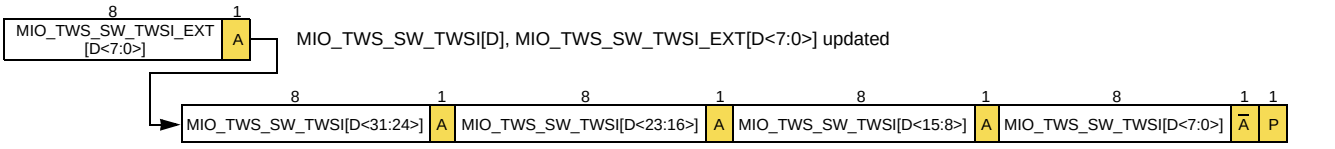
Three-Byte Read ([SOVR] = 1, [SIZE] = 2)



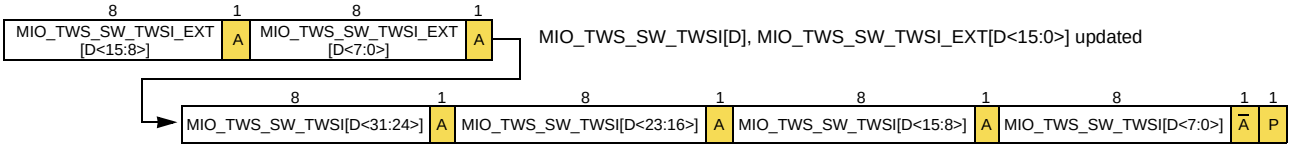
Four-Byte Read ([SOVR] = 0, [OP<3>] = 1) or ([SOVR] = 1, [SIZE] = 3)



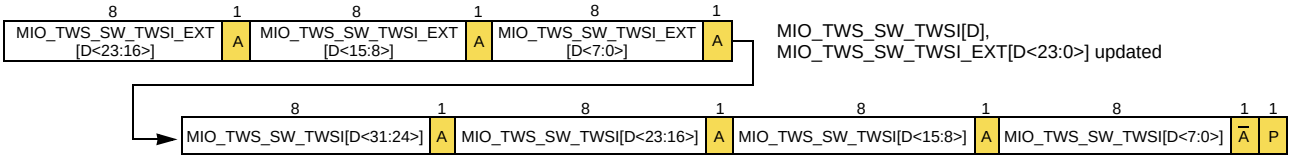
Five-Byte Read ([SOVR] = 1, [SIZE] = 4)



Six-Byte Read ([SOVR] = 1, [SIZE] = 5)



Seven-Byte Read ([SOVR] = 1, [SIZE] = 6)



Eight-Byte Read ([SOVR] = 1, [SIZE] = 7)

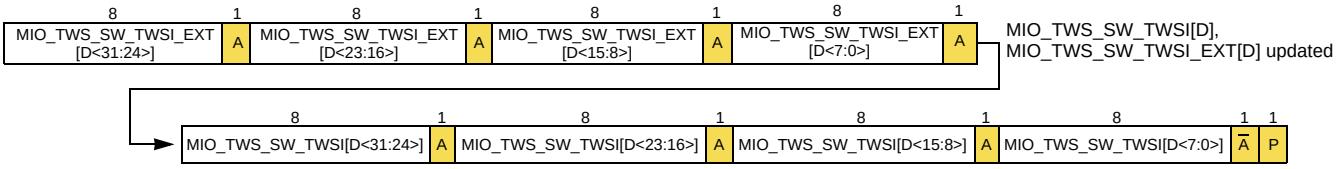
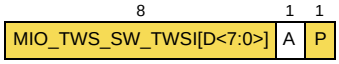


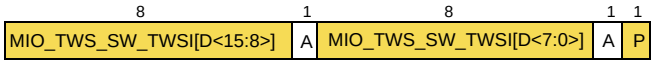
Figure 17–6 HLC as Master, Read-Bytes Step

MIO_TWS_SW_TWSI Write Operations, with [SLONLY] = 0, [OP<2>] = 0, [R] = 0)

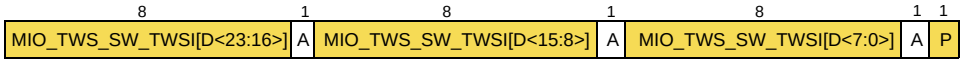
One-Byte Write ([SOVR] = 0, [OP<3>] = 0) or ([SOVR] = 1, [SIZE] = 0)



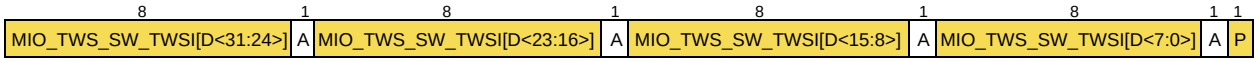
Two-Byte Write ([SOVR] = 1, [SIZE] = 1)



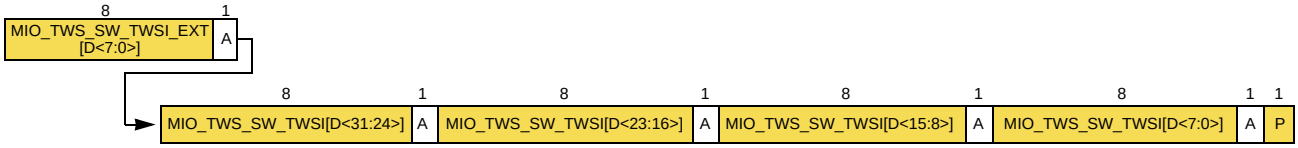
Three-Byte Write ([SOVR] = 1, [SIZE] = 2)



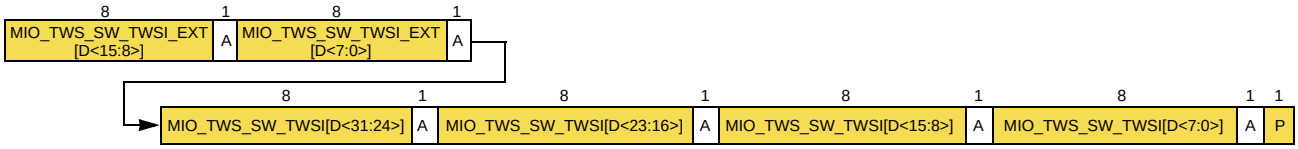
Four-Byte Write ([SOVR] = 0, [OP<3>] = 1) or ([SOVR] = 1, [SIZE] = 3)



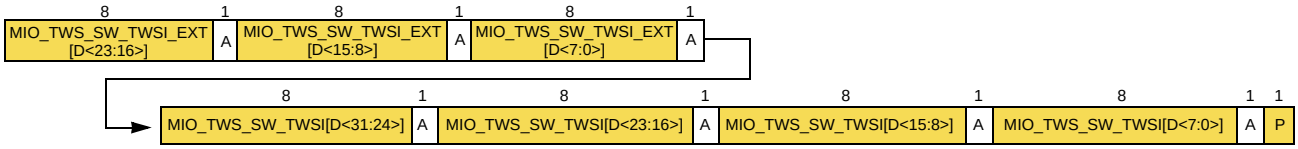
Five-Byte Write ([SOVR] = 1, [SIZE] = 4)



Six-Byte Write ([SOVR] = 1, [SIZE] = 5)



Seven-Byte Write ([SOVR] = 1, [SIZE] = 6)



Eight-Byte Write ([SOVR] = 1, [SIZE] = 7)

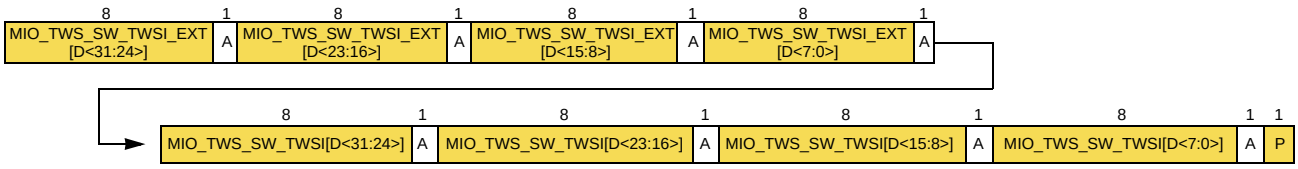


Figure 17–7 HLC as Master, Write-Bytes Step

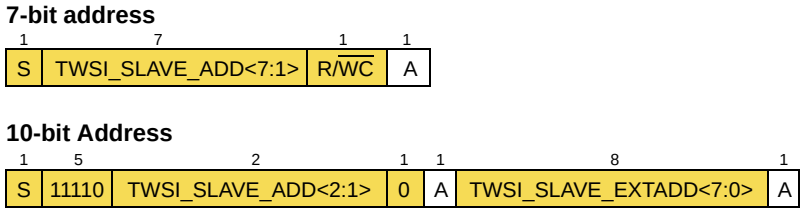


Figure 17–8 HLC as Slave, Initial-Address Step

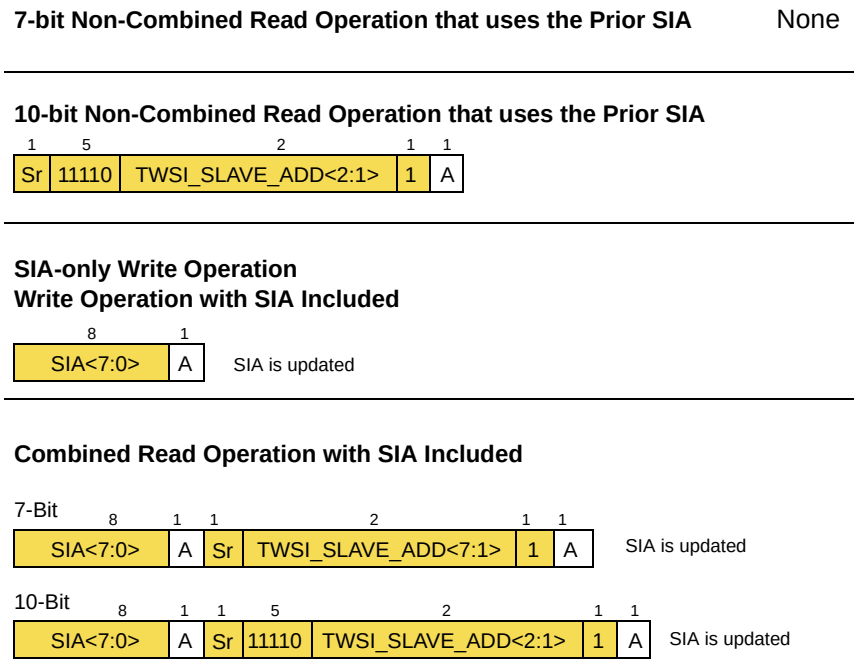


Figure 17–9 HLC as Slave, Address-Extension Step

Table 17–1 Slave Internal-Address Register

Bit Pos	Field Name	Description
<7:4>	—	Reserved
<3>	SIZE	Size of the data access: 0 = 8-bit access, 1 = 32-bit access. Must be 0 for an MIO_TWS_TWSI_SW read operation.
<2:1>	—	Reserved
<0>	REG	Identifies the register to be accessed: 0 = MIO_TWS_SW_TWSI register (read-only) 1 = MIO_TWS_TWSI_SW register (read/write)

Figure 17–10 shows the possible read and write bytes components supported for sequences with the HLC as a slave. SIA<3> (and the stop condition in the case of a write) allows the external TWSI device to vary the amount of data transferred in the subsequent read or write when N2 acts as a slave. For reads, it can be one or four bytes, plus one more byte for the Vbyte. For writes, it can be 0, 1, or four bytes. Note that the 8-bit write to MIO_TWS_TWSI_SW is equivalent to a 4-byte write with 0s for the first 3 bytes.

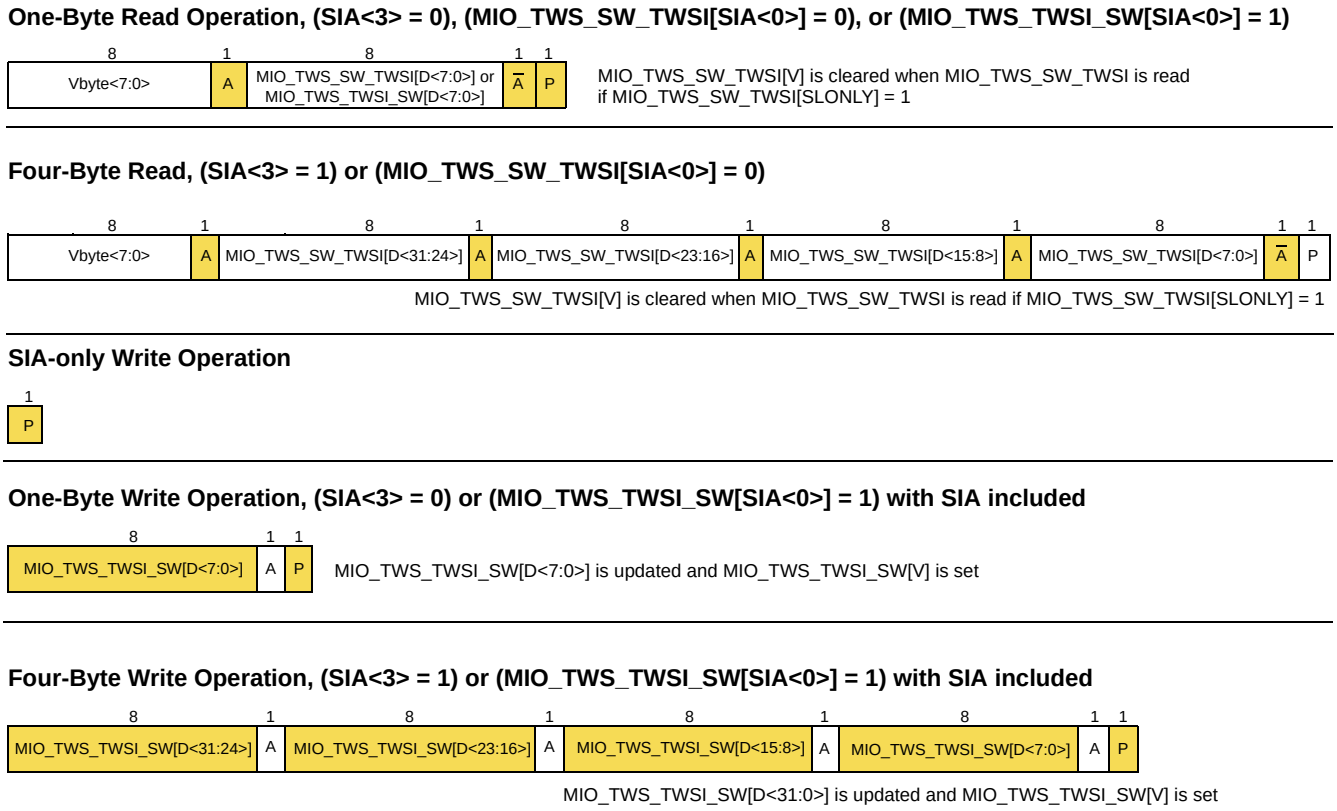


Figure 17–10 HLC as Slave, Read/Write-Bytes Step

Vbyte provides a validity indication on a read. Table 17–2 describes the Vbyte.

Table 17–2 Validity Indication for Slave Reads

Bit Pos	Field Name	Description
<7>	V	If MIO_TWS_SW_TWSI is read, V is: MIO_TWS_SW_TWSI[V] **AND** MIO_TWS_SW_TWSI[SLONLY]. If MIO_TWS_TWSI_SW is read, V is: MIO_TWS_TWSI_SW[V<1>] **AND** MIO_TWS_TWSI_SW[V<0>]
<6:0>	—	Unused, read as 0x0.

17.3 Direct TWSI Core Usage

The CN50XX cores can communicate directly with the TWSI core via CSR accesses through MIO_TWS_SW_TWSI (refer to Section 17.4). Direct TWSI-core accesses should be avoided when the TWSI HLC is enabled (i.e. when TWSI_CTL[CE] is set to 1). TWSI_CTL[CE] is assumed to be clear (thus disabling the HLC) for the rest of Section 17.3. This provides lower-level access to TWSI functions that can be used to create a wider variety of transaction sequences. (Refer to the TWSI Status Register description for more possible status conditions.)

The TWSI interface can be used when TWSI_CTL[ENAB] = 1 in the following modes:

- master transmit mode
- master receive mode
- slave transmit mode
- slave receive mode

Each of these modes is described in the following subsections.

17.3.1 Master Transmit Mode

In master transmit mode, the TWSI core transmits a number of bytes to a slave receiver.

A master-mode transaction starts when the software sets TWSI_CTL[STA] to 1. The TWSI core then tests the TWSI bus and transmits a START condition when the bus is free. When the START condition has been transmitted, TWSI_CTL[IFLG] is set to 1, and the status code in TWSI_STAT is 0x08 (*START condition transmitted*). Then TWSI_DATA must be loaded with either a 7-bit slave address or the first part of a 10-bit slave address, with the LSB cleared to 0 (i.e. with a write bit) to specify transmit mode. TWSI_CTL[IFLG] should then be cleared to 0 to prompt the transfer to continue.

After the 7-bit slave address (or the first part of a 10-bit address) plus the write bit have been transmitted, TWSI_CTL[IFLG] is set again. There are a number of status codes that might show up in the TWSI_STAT register. They are listed in Table 17–3.

Table 17–3 TWSI_STAT Status Codes (after 7-Bit Address or First Part of 10-Bit Address)

Status Code	TWSI Core State	Core Response	Next TWSI Core Action
0x18	Addr[1] + W[2] transmitted, ACK received	For a 7-bit address:	
		Write the byte to TWSI_DATA, clear TWSI_CTL[IFLG], **or**	Transmit data byte, receive ACK
		Set TWSI_CTL[STA], clear TWSI_CTL[IFLG], **or**	Transmit repeated START
		Set TWSI_CTL[STP], clear TWSI_CTL[IFLG], **or**	Transmit STOP
		Set TWSI_CTL[STA] and TWSI_CTL[STP], clear TWSI_CTL[IFLG]	Transmit STOP then START
		For a 10-bit address:	
		Write extended-address byte to TWSI_DATA, clear TWSI_CTL[IFLG]	Transmit extended-address byte.
0x20	Addr[1] + W[2] transmitted, $\overline{\text{ACK}}$ received	Same as for 0x18.	Same as for 0x18.

Table 17–3 TWSI_STAT Status Codes (after 7-Bit Address or First Part of 10-Bit Address) (Continued)

Status Code	TWSI Core State	Core Response	Next TWSI Core Action
0x38	Arbitration lost	Clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STA], clear TWSI_CTL[IFLG]	Return to idle Transmit START when bus is free
0x68	Arbitration lost, SLA[3] + W[2] received, ACK transmitted	Clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 0 **or** Clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 1	Receive data byte, transmit $\overline{ACK}$ Receive data byte, transmit ACK
0x78	Arbitration lost, general call addr received, ACK transmitted	Same as for 0x68.	Same as for 0x68.
0xB0	Arbitration lost, SLA[3] + R[4] received, ACK transmitted	Write the byte to TWSI_DATA, clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 0 **or** Write the byte to TWSI_DATA, clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 1	Transmit last byte, receive ACK Transmit data byte, receive ACK

1. Addr represents the contents of TWSI_DATA, either the 7-bit address or the first part of the 10-bit address.
2. W represents the write bit (i.e. LSB of the byte = 0).
3. SLA represents the slave address (first part), in the TWSI_SLAVE_ADD register.
4. R represents the read bit (i.e. LSB of the byte = 1).

If 10-bit addressing is used, then once the first part of a 10-bit address plus the write bit have been transmitted successfully, the status code is 0x18 or 0x20.

After this interrupt has been serviced, and the second part of the address transmitted, the TWSI_STAT register contains one of the codes shown in Table 17–4.

Table 17–4 TWSI_STAT Status Codes (after Second Part of 10-Bit Address)

Status Code	TWSI Core State	Core Response	Next TWSI Core Action
0x38	Arbitration lost	Clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STA], clear TWSI_CTL[IFLG]	Return to idle Transmit START when bus is free
0x68	Arbitration lost, SLAX[1] + W[2] received, ACK transmitted	Clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 0 **or** Clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 1	Receive data byte, transmit $\overline{ACK}$ Receive data byte, transmit ACK
0xB0	Arbitration lost, SLAX[1] + R[3] received, ACK transmitted	Write the byte to TWSI_DATA, clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 0 **or** Write the byte to TWSI_DATA, clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 1	Transmit last byte, receive ACK Transmit data byte, receive ACK

Table 17–4 TWSI_STAT Status Codes (after Second Part of 10-Bit Address) (Continued)

Status Code	TWSI Core State	Core Response	Next TWSI Core Action
0xD0	Addr[4] + W[2] transmitted, ACK received	Write the byte to TWSI_DATA, clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STA], clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STP], clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STA] and TWSI_CTL[STP], clear TWSI_CTL[IFLG]	Transmit data byte, receive ACK Transmit repeated START Transmit STOP Transmit STOP then START
0xD8	Addr[4] + W[2] transmitted, $\overline{\text{ACK}}$ received	Same as for 0xD0.	Same as for 0xD0.

1. SLAX represents the slave address (second part), in the TWSI_SLAVE_ADD_EXT register.
2. W represents the write bit (i.e. LSB of the byte = 0).
3. R represents the read bit (i.e. LSB of the byte = 1).
4. Addr represents the contents of TWSI_DATA, the second part of the 10-bit address.

If a repeated START condition has been transmitted, the status code is 0x10 instead of 0x08.

After each data byte has been transmitted, TWSI_CTL[IFLG] is set to 1 and one of the three listed in Table 17–5 is loaded into TWSI_STAT.

Table 17–5 TWSI_STAT Status Codes (after Repeated START Transmission)

Status Code	TWSI Core State	Core Response	Next TWSI Core Action
0x28	Data byte transmitted, ACK received	Write the byte to TWSI_DATA, clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STA], clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STP], clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STA] and TWSI_CTL[STP], clear TWSI_CTL[IFLG]	Transmit data byte, receive ACK Transmit repeated START Transmit STOP Transmit STOP then START
0x30	Data byte transmitted, $\overline{\text{ACK}}$ received	Same as for 0x28.	Same as for 0x28.
0x38	Arbitration lost	Clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STA], clear TWSI_CTL[IFLG]	Return to idle Transmit START when bus is free

When all bytes have been transmitted, TWSI_CTL[STP] should be set to 1. The TWSI core then does the following:

1. Transmits a STOP condition.
2. Clears TWSI_CTL[STP] to 0.
3. Returns to the idle state.

17.3.2 Master Receive Mode

In master receive mode, the TWSI core receives a number of bytes from a slave transmitter.

Refer to Section 17.3.1 and note that this is nearly identical. After the START condition has been transmitted, TWSI_CTL[IFLG] is set to 1 and a status code of 0x08 (*START condition transmitted*) is loaded into TWSI_STAT. Load the TWSI_DATA register with the 7-bit slave address (or the first part of the 10-bit slave address), with the LSB set to 1 to signify a read operation (if 7-bit). Then clear TWSI_CTL[IFLG] to 0 to prompt the transfer to continue.

When the 7-bit slave address (or the first part of the 10-bit slave address) and the read bit have been transmitted, TWSI_CTL[IFLG] is set again to 1. A number of status codes may possibly be in TWSI_STAT. They are shown in Table 17–6.

Table 17–6 TWSI_STAT Status Codes (Master Receive Mode)

Status Code	TWSI Core State	Core Response	Next TWSI Core Action
0x40	Addr[1] + R[2] transmitted, ACK received	For a 7-bit/10-bit address: Clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 0 **or** Clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 1	 Receive data byte, transmit $\overline{ACK}$ Receive data byte, transmit ACK
0x48	$\overline{Addr}$[1] + R[2] transmitted, $\overline{ACK}$ received	For a 7-bit/10-bit address: Set TWSI_CTL[STA], clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STP], clear TWSI_CTL[IFLG], **or** Set TWSI_CTL[STA] and TWSI_CTL[STP], clear TWSI_CTL[IFLG]	 Transmit repeated START Transmit STOP Transmit STOP then START
0x38	Same as for Master Transmit.	Same as for Master Transmit.	Same as for Master Transmit.
0x68	Same as for Master Transmit.	Same as for Master Transmit.	Same as for Master Transmit.
0x78	Same as for Master Transmit.	Same as for Master Transmit.	Same as for Master Transmit.
0xB0	Same as for Master Transmit.	Same as for Master Transmit.	Same as for Master Transmit.

1. Addr represents the contents of TWSI_DATA, either the 7-bit address or the first part of the 10-bit address.
2. R represents the read bit (i.e. LSB of the byte = 1).

If 10-bit addressing is being used, the slave is first addressed using the full 10-bit address plus the write bit. The master then issues a restart followed by the first part of the 10-bit address again except with the read bit, after which the status code is 0x40 or 0x48. It is the responsibility of the slave to remember that it had been selected prior to the restart.

If a repeated START condition is transmitted, the status code is 0x10 rather than 0x08.

After each data byte has been received, TWSI_CTL[IFLG] is set to 1 and one of the three codes listed in Table 17–7 is in TWSI_STAT.

Table 17–7 TWSI_STAT Status Codes (Master Receive – Data Received)

Status Code	TWSI Core State	Core Response	Next TWSI Core Action
0x50	Data byte received, ACK transmitted	Read TWSI_DATA, clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 0 **or** Read TWSI_DATA, clear TWSI_CTL[IFLG], TWSI_CTL[AAK] = 1	Receive data byte, transmit $\overline{\text{ACK}}$ Receive data byte, transmit ACK
0x58	Data byte received, $\overline{\text{ACK}}$ transmitted	Read TWSI_DATA, Set TWSI_CTL[STA], clear TWSI_CTL[IFLG], **or** Read TWSI_DATA, Set TWSI_CTL[STP], clear TWSI_CTL[IFLG], **or** Read TWSI_DATA, Set TWSI_CTL[STA] and TWSI_CTL[STP], clear TWSI_CTL[IFLG]	Transmit repeated START Transmit STOP Transmit STOP then START
0x38	Arbitration lost in $\overline{\text{ACK}}$ bit	Same as for Master Transmit.	Same as for Master Transmit.

When all bytes have been received (an $\overline{\text{ACK}}$ should have been transmitted), then set TWSI_CTL[STP]. The TWSI core then does the following:

1. Transmits a STOP condition.
2. Clears TWSI_CTL[STP] to 0.
3. Returns to the idle state.

17.3.3 Slave Transmit Mode

In slave transmit mode, the TWSI core transmits a number of bytes to a master receiver. TWSI_CTL[AAK] must be set to 1 to enable the core to initially respond as a slave.

The TWSI core enters slave transmit mode when it receives its own slave address and a read bit after a START condition. The TWSI core then transmits an acknowledge (ACK) bit and sets TWSI_CTL[IFLG] to 1. The TWSI_STAT register is set to 0xA8.

NOTE: When the TWSI core has an extended slave address (indicated by 11110 in TWSI_SLAVE_ADD[A<7:3>]), it transmits ACK after the first address byte is received after a restart. An interrupt is then generated, TWSI_CTL[IFLG] is set, and TWSI_STAT is set to 0xA8. No second address byte is sent by the master – it is up to the slave to remember that it had been selected prior to the restart.

Slave transmit mode can also be entered directly from a master mode if arbitration is lost in master mode during the transmission of an address and the TWSI core's slave address and read bit are received.

In this situation, TWSI_STAT is set to 0xB0.

The data byte to be transmitted should then be loaded into TWSI_DATA and TWSI_CTL[IFLG] cleared to 0. When the TWSI core has transmitted the byte and received ACK, it sets TWSI_CTL[IFLG] to 1 and sets TWSI_STAT to 0xB8. Once the last byte to be transmitted has been loaded into TWSI_DATA, both TWSI_CTL[IFLG, AAK] should be cleared to 0. After the last byte has been transmitted, TWSI_CTL[IFLG] is set to 1 and TWSI_STAT is set to 0xC8.

The TWSI core then returns to the idle state and TWSI_CTL[AAK] must be set to 1 before slave mode can be entered again.

If no ACK is received after transmitting a byte, the TWSI core sets TWSI_CTL[IFLG] to 1 and TWSI_STAT to 0xC0, then returns to the idle state.

If the STOP condition is detected after an ACK, the TWSI core returns to idle.

17.3.4 Slave Receive Mode

In slave receive mode, the TWSI core receives a number of bytes from a master transmitter. TWSI_CTL[AAK] must be set to 1 to enable the TWSI core to respond initially as a slave.

The TWSI core enters slave receive mode when it receives its own slave address and a write bit after a START condition. The TWSI core then transmits ACK and sets TWSI_CTL[IFLG] to 1 and TWSI_STAT to 0x60.

It also enters slave receive mode when it receives the general call address 0x00 if TWSI_SLAVE_ADD[GCE] is set to 1. The status code in TWSI_STAT is then set to 0x70.

NOTE: When the TWSI core has an extended slave address (indicated by 11110 in TWSI_SLAVE_ADD[A<7:3>]), it transmits ACK after the first address byte is received after a restart. No interrupt is generated, TWSI_CTL[IFLG] is not set, and TWSI_STAT is not changed.

Only after the second address byte is received does the TWSI core generate an interrupt, set TWSI_CTL[IFLG] to 1, and set the appropriate status code (either 0x60 or 0x70).

Slave transmit mode can also be entered directly from a master mode if arbitration is lost in master mode during the transmission of an address and either the TWSI core's slave address and write bit are received, or the general call address is received when TWSI_SLAVE_ADD[GCE] = 1.

In this situation, TWSI_STAT is set to either 0x68 (slave address) or 0x78 (general call address). TWSI_CTL[IFLG] must be cleared to 0 to allow the data transfer to continue.

If TWSI_CTL[AAK] is set to 1, then after each byte is received, an ACK is transmitted, TWSI_CTL[IFLG] is set to 1, and TWSI_STAT is set to 0x80 (slave address) or 0x90 (general call address). The received-data byte can be read from TWSI_DATA, then TWSI_CTL[IFLG] must be cleared to 0 to allow the transfer to continue.

When the STOP condition or a repeated START condition is detected after ACK, TWSI_CTL[IFLG] is set to 1 and TWSI_STAT is set to 0xA0.

If TWSI_CTL[AAK] = 0 during a transfer, the TWSI core transmits an $\overline{ACK}$ after the next byte is received, sets TWSI_CTL[IFLG] to 1, and sets TWSI_STAT to either 0x88 (slave address) or 0x98 (general call address). When TWSI_CTL[IFLG] is cleared to 0, the TWSI core returns to the idle state.

17.3.5 TWSI Core Flow Diagrams

Figure 17–11 shows the master-mode flow, and Figure 17–12 shows the slave-mode flow.

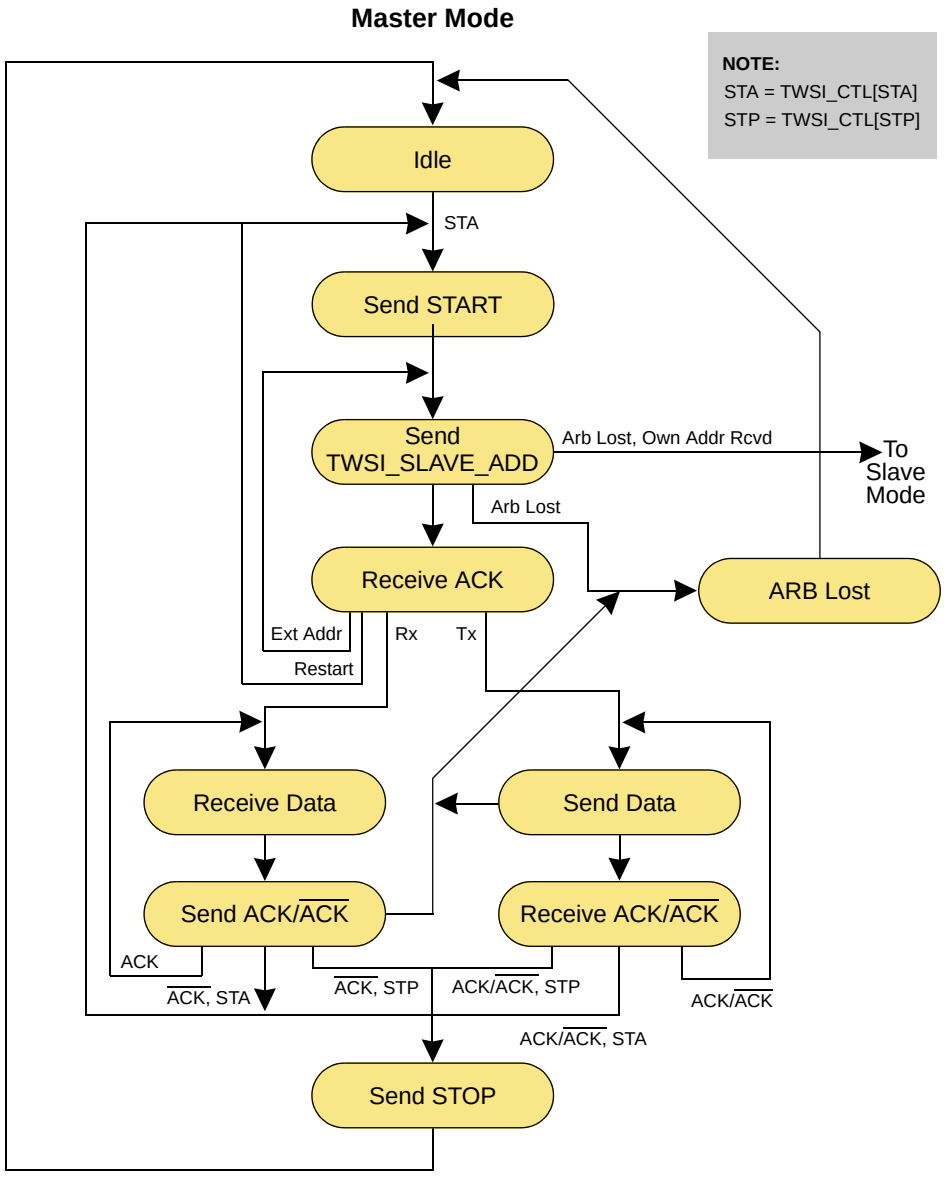


Figure 17–11 Master-Mode Flow Diagram

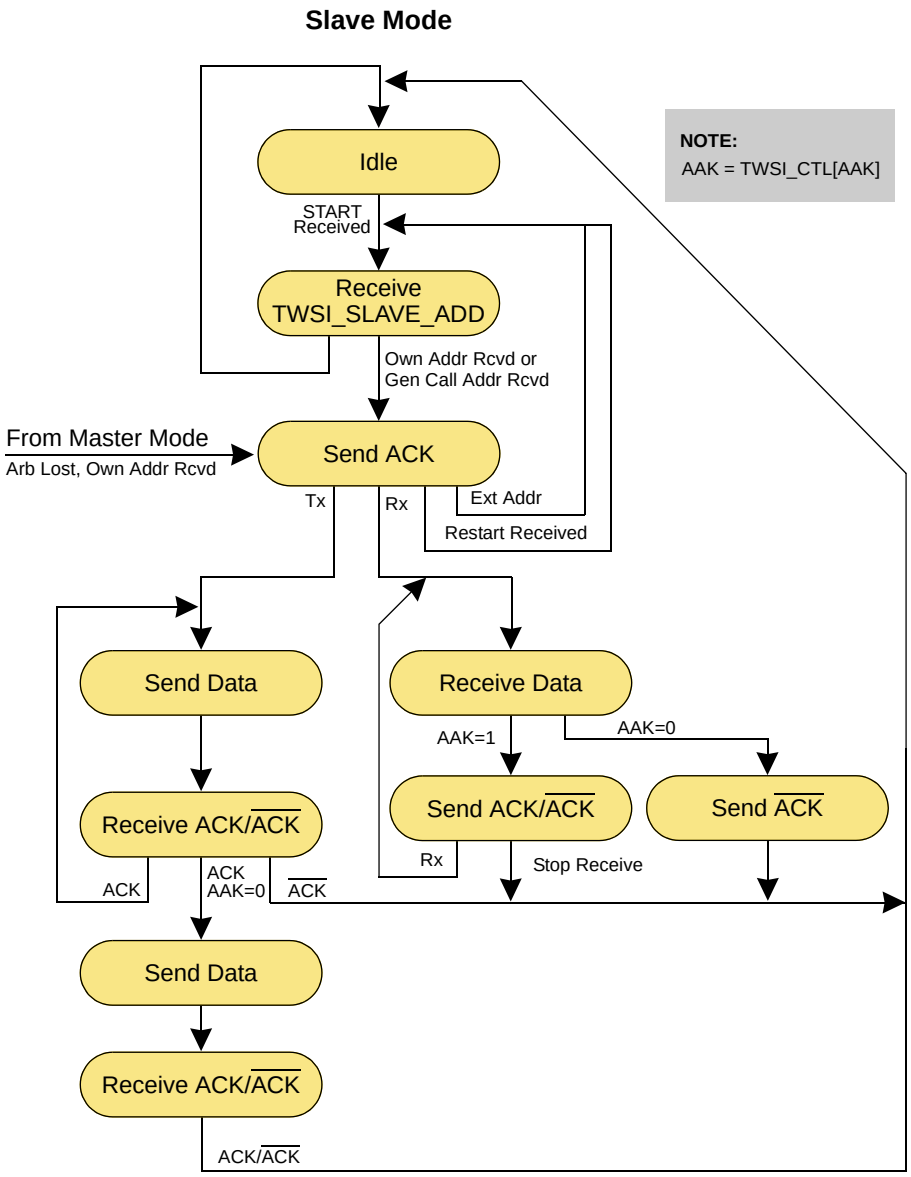


Figure 17–12 Slave-Mode Flow Diagram

17.4 TWSI Control Registers

This section describes the internal registers of the TWSI core. These registers can be read/written by write operations to MIO_TWS_SW_TWSI with [OP<2>] = 1 and [SLONLY] = 0.

The registers described are:

- TWSI slave address register (TWSI_SLAVE_ADD)
- TWSI slave extended-address register (TWSI_SLAVE_EXTADD)
- TWSI data register (TWSI_DATA)
- TWSI control register (TWSI_CTL)
- TWSI status register (TWSI_STAT)
- TWSI master clock register (TWSI_CLK)
- TWSI clock control register (TWSI_CLKCTL)
- TWSI software reset register (TWSI_RST)

17.4.1 TWSI Slave Address Register

The TWSI_SLAVE_ADD register is described in Table 17–8.

Table 17–8 TWSI_SLAVE_ADD Bit Description

Bit Pos	Value with 7-bit Addressing	Value with 10-bit Addressing	Reset Value
<7>	Slave address bit <6>	1	1
<6>	Slave address bit <5>	1	1
<5>	Slave address bit <4>	1	1
<4>	Slave address bit <3>	1	0
<3>	Slave address bit <2>	0	1
<2>	Slave address bit <1>	Slave address bit <9>	1
<1>	Slave address bit <0>	Slave address bit <8>	1
<0>	GCE general call address enable	GCE general call address enable	0

For 7-bit addressing:

Slave address is the 7-bit address of CN50XX when in slave mode. When CN50XX receives this address after a START condition, it sets TWSI_CTL[IFLG] (i.e. the TWSI core-interrupt bit) and the TWSI core enters slave mode, assuming TWSI_CTL[ENAB] is set to 1.

If GCE = 1, CN50XX also recognizes the general call address (0x00).

CN50XX resets to 7-bit addressing with an address of 0x77.

For 10-bit addressing:

When the received address has bits <7:3> = 11110, CN50XX recognizes this as the first part of a 10-bit address, and if the next two bits match TWSI_SLAVE_ADD<2:1> (i.e. slave address <9:8>), it sends an ACK. After the next byte of the address has been received, and if the address matches TWSI_SLAVE_EXTADD, CN50XX sets TWSI_CTL[IFLG] and the TWSI core enters slave mode. (TWSI_SLAVE_ADD<7:3> must be programmed as specified in order for CN50XX to respond as a slave to a 10-bit address.)

17.4.2 TWSI Slave Extended-Address Register

The TWSI_SLAVE_EXTADD register is described in Table 17–9.

Table 17–9 TWSI_SLAVE_EXTADD Bit Description

Bit Pos	Value with 7-bit Addressing	Value with 10-bit Addressing	Reset Value
<7>	—	Slave address bit <7>	0
<6>	—	Slave address bit <6>	0
<5>	—	Slave address bit <5>	0
<4>	—	Slave address bit <4>	0
<3>	—	Slave address bit <3>	0
<2>	—	Slave address bit <2>	0
<1>	—	Slave address bit <1>	0
<0>	—	Slave address bit <0>	0

17.4.3 TWSI Data Register

TWSI_DATA should not be accessed when the TWSI HLC is enabled (i.e. when TWSI_CTL[CE] = 1).

This register contains the data byte/slave address to be transmitted, or the data byte that has just been received.

- In transmit mode, the byte is sent MSB first
- In receive mode, the first bit received is placed in the MSB of the register.

After each byte is transmitted, TWSI_DATA contains the byte that was actually present on the bus, so in the event of lost arbitration, it will contain the received byte.

17.4.4 TWSI Control Register

The TWSI_CTL register is described in Table 17–10.

Table 17–10 TWSI_CTL Bit Description

Bit Pos	Field Name	Reset Value	Description
<7>	CE	1	High-level controller enable. When this bit is set to 1, the HLC interface is enabled and drives the sequence of the TWSI bus commands to transfer or receive data. When this bit is cleared to 0, the HLC interface is disabled. The software can still use the TWSI interface by issuing individual commands using TWSI control, status and data registers
<6>	ENAB	1	Bus enable. Must be set to 1 to use the TWSI interface (either via the HLC interface or via direct core usage).

Table 17–10 TWSI_CTL Bit Description (Continued)

Bit Pos	Field Name	Reset Value	Description
<5>	STA	0	Master-mode start. This bit should be clear when the HLC is enabled (i.e. when TWSI_CTL[CE] is set to 1). When this bit is set to 1, the TWSI core enters master mode and transmits a START condition on the bus when the bus is free. • If set to 1 when the controller is already in master mode and one or more bytes have been transmitted, then a repeated START condition is sent. • If this bit is set to 1 when the TWSI core is being accessed by an external device in slave mode, the core completes the data transfer in slave mode then enters master mode when the bus has been released. If both STA and STP bits are set, the TWSI core first transmits the STOP condition (if in master mode) then transmits the START condition. This bit is cleared automatically after a START condition has been sent. Writing a 0 to this bit has no effect.
<4>	STP	0	Master-mode stop. This bit should be clear when the HLC is enabled (i.e. when TWSI_CTL[CE] is set). • If this bit is set to 1 when the TWSI core is in master mode, a STOP condition is transmitted on the TWSI bus. • If this bit is set to 1 when the TWSI core is in slave mode, the HLC behaves as if a STOP condition has been received, but no STOP condition is transmitted on the TWSI bus. If both STA and STP bits are set, the TWSI core first transmits the STOP condition (if in master mode) then transmits the START condition. This bit is cleared automatically. Writing a 0 to this bit has no effect.
<3>	IFLG	0	Interrupt flag. This bit should be ignored when the HLC is enabled (i.e. when TWSI_CTL[CE] is set). This bit is automatically set to 1 when any of 28 (out of the possible 29) TWSI core states is entered. The only state that does not set this bit is state 0xF8 (see Section 17.4.5). When the TWSI core sets this bit, the low period of the TWSI bus clock line (SCL) is stretched and the data transfer is suspended. When 0 is written to this bit, the bit is cleared and the TWSI clock line is released.

Table 17–10 TWSI_CTL Bit Description (Continued)

Bit Pos	Field Name	Reset Value	Description
<2>	AAK	1	Assert acknowledge. This bit should be set when the HLC is enabled (i.e. when TWSI_CTL[CE] is set). When this bit is set to 1, an ACK is sent during the acknowledge clock pulse on the TWSI bus if any of the following occurs: • the whole of a matching 7-bit slave address or the first or the second byte of a matching 10-bit slave address has been received, • the general call address has been received and it was enabled, • a data byte has been received in master or slave mode. When this bit is cleared to 0, a "not acknowledge" ($\overline{ACK}$) is sent when a data byte is received in master or slave mode. • If this bit is cleared to 0 in slave transmitter mode, the byte in TWSI_DATA is assumed to be the last byte. After this byte has been transmitted, the TWSI core enters the idle state 0xC8. The TWSI core does not respond as a slave unless this bit is set.
<1:0>	—	00	Reserved.

17.4.5 TWSI Status Register

TWSI_STAT is a read-only register containing a state code.

- When this register contains the state 0xF8, no relevant status information is available and no interrupt bit is set.
- All the other 28 states correspond to a non-idle state of the TWSI.

When any of these non-idle states is entered, the corresponding state appears in this register and TWSI_CTL[IFLG] is set to 1. When TWSI_CTL[IFLG] is cleared to 0, the state returns to 0xF8.

If an illegal condition occurs on the TWSI bus, the bus-error state is entered (status code 0x00). To recover from this state, TWSI_CTL[STP] must be set to 1 and TWSI_CTL[IFLG] cleared to 0. The TWSI controller then returns to the idle state, and no STOP condition is transmitted on the TWSI bus.

The TWSI core state codes are listed in Table 17–11.

Table 17–11 TWSI Core-State Codes

Code	TWSI Core State
0x00	Bus error
0x08	START condition transmitted
0x10	Repeated START condition transmitted
0x18	Address + write bit transmitted, ACK received
0x20	Address + write bit transmitted, $\overline{\text{ACK}}$ received
0x28	Data byte transmitted in master mode, ACK received
0x30	Data byte transmitted in master mode, $\overline{\text{ACK}}$ received
0x38	Arbitration lost in address or data byte
0x40	Address + read bit transmitted, ACK received
0x48	Address + read bit transmitted, $\overline{\text{ACK}}$ received
0x50	Data byte received in master mode, ACK transmitted
0x58	Data byte received in master mode, $\overline{\text{ACK}}$ transmitted
0x60	Slave address + write bit received, ACK transmitted
0x68	Arbitration lost in address as master, slave address + write bit received, ACK transmitted
0x70	General call address received, ACK transmitted
0x78	Arbitration lost in address as master, general call address received, ACK transmitted
0x80	Data byte received after slave address received, ACK transmitted
0x88	Data byte received after slave address received, $\overline{\text{ACK}}$ transmitted
0x90	Data byte received after general call address received, ACK transmitted
0x98	Data byte received after general call address received, $\overline{\text{ACK}}$ transmitted
0xA0	STOP or repeated START condition received in slave mode
0xA8	Slave address + read bit received, ACK transmitted
0xB0	Arbitration lost in address as master, slave address + read bit received, ACK transmitted
0xB8	Data byte transmitted in slave mode, ACK received
0xC0	Data byte transmitted in slave mode, $\overline{\text{ACK}}$ received
0xC8	Last byte transmitted in slave mode, ACK received
0xD0	Second address byte + write bit transmitted, ACK received
0xD8	Second address byte + write bit transmitted, $\overline{\text{ACK}}$ received
0xF8	No relevant status information

The TWSI core prevents the software/hardware from causing a TWSI bus error in most of the cases (e.g. it does not let the firmware send a STOP condition immediately after a START condition, as it is illegal). However, the TWSI core does allow the firmware to cause a bus error by sending a STOP condition after the first byte of a 10-bit address. The controller can recover from the error state as described above.

17.4.6 TWSI Master Clock Register

The 8-bit value in TWSI_CLK controls the frequency ratio between CN50XX's core clock and the TWSI controller's clock (TCLK). The TWSI_CLK register is described in Table 17–12.

Table 17–12 TWSI_CLK Bit Description

Bits	Field Name	Reset Value	Description
<7:0>	THP	0x18	TCLK half period.

The period of TCLK is defined as: 2 × (THP+1) times the core clock, so the frequency of TCLK is defined as:

$$\text{TCLK frequency} = \frac{\text{core frequency}}{2 \times (\text{THP} + 1)}$$

TWSI_CLK must never be set to any of the values 0 – 4.

The reset value with 600-MHz core-clock cycle generates a 12-MHz TCLK cycle.

17.4.7 TWSI Clock Control Register

TWSI_CLKCTL is a write-only register. The bits <6:0> control the frequency at which the TWSI bus is sampled and the frequency of the TWSI clock line when the TWSI controller is in master mode. TWSI_CLKCTL is described in Table 17–13.

Table 17–13 TWSI_CLKCTL Bit Description

Bits	Field Name	Reset Value	Description
<7>	—	—	Reserved. Read as 0.
<6:3>	M	0x2	M divider
<2:0>	N	0x0	N divider

The TWSI clock frequency (TCLK) is first divided by a factor of 2^N, where N is the value defined by <2:0>. The output of this clock divider is FSAMP, which is then divided by a further factor of (M+1)×10, where M is the value defined by bits <6:3>. The output of this clock divider is FOSCL.

The TWSI bus is sampled by the controller at the frequency defined by FSAMP (N is <2:0> of TWSI_CLKCTL):

$$\text{FSAMP} = \text{TCLK} / 2^N$$

The TWSI OSCL output frequency, in master mode, is (N is <2:0> and M is <6:3> of TWSI_CLKCTL):

$$\text{FOSCL} = \frac{\text{FSAMP}}{(M+1)\times 10} = \frac{\text{TCLK}}{2^N \times (M + 1) \times 10}$$

The use of two separately programmable dividers allows the master mode output frequency to be set independently of the frequency at which the TWSI bus is sampled. This is particularly useful in multimaster systems because the frequency at which the TWSI bus is sampled must be at least 10 times the frequency of the fastest master on the bus to ensure that START and STOP conditions are always detected. By using two programmable clock divider stages, a high sampling frequency can be ensured while allowing the master mode output to be set to a lower frequency.

For a core-clock frequency of 600 MHz, immediately after reset CN50XX samples at the rate of 12 MHz (FSAMP), and CN50XX drives at the frequency of 400 KHz (FOSCL).

17.4.8 TWSI Software Reset Register

A software reset may be applied to the TWSI bus interface by writing any value to TWSI_RST. A software reset sets the TWSI core back to idle (TWSI_STAT = 0xF8) and sets TWSI_CTL[STA, STP, IFLG] to 0s.

NOTE: After this register is written, the TWSI controller is not available for any communication with the TWSI bus for three TCLK cycles (for the default value of the TWSI_CLK, it derives to 315 core-clock cycles).

17.5 TWSI Registers

The TWSI registers are listed in Table 17–14.

Table 17–14 TWSI Registers

Register	Address	CSR Type[1]	Detailed Description
MIO_TWS_SW_TWSI	0x0001180000001000	RSL	See page 647
MIO_TWS_TWSI_SW	0x0001180000001008	RSL	See page 648
MIO_TWS_INT	0x0001180000001010	RSL	See page 649
MIO_TWS_SW_TWSI_EXT	0x0001180000001018	RSL	See page 650

1. RSL-type registers are accessed indirectly across the I/O Bus.

Software to TWSI Register MIO_TWS_SW_TWSI

This register allows software to:

- Initiate master-mode operations with a write operation, and read the result with a read operation.
- Load four bytes for later retrieval (slave mode) with a write operation and check validity with a read operation.
- Launch a configuration read/write operation with a write operation and read the result with a read operation.

This register should be read or written by software, and read by the TWSI device. The TWSI device can use either two-byte or five-byte reads to reference this register.

The TWSI device considers this register valid when [V] = 1 and [SLONLY] = 1. See Table 17–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	V	RC/W	0	—	Valid bit: • set on a write operation (should always be written with a 1) • cleared when a TWSI master-mode operation completes • cleared when a TWSI configuration register access completes • cleared when the TWSI device reads the register if SLONLY = 1
<62>	SLONLY	R/W	—	—	Slave-only mode. When this bit is set, no operations are initiated with a write operation. Only the D field is updated in this case When this bit is clear, a write operation initiates either a master-mode operation or a TWSI configuration register access.
<61>	EIA	R/W	0	—	Extended internal address. Sends an additional internal address byte (the MSB of IA is from MIO_TWS_SW_TWSI_EXT[IA]).
<60:57>	OP	WO	—	—	Opcode field. When the register is written with SLONLY = 0, this field initiates one of the following read or write operations: 0000 = 7-bit byte master-mode operation 0001 = 7-bit byte combined-read master-mode operation, 7-bit byte write-with-IA master-mode operation 0010 = 10-bit byte master-mode operation 0011 = 10-bit byte combined-read master-mode operation, 10-bit byte write-with-IA master-mode operation 0100 = TWSI master-clock register, TWSI_CLK in 17.4.6 0110 = See EOP_IA field 1000 = 7-bit 4-byte master-mode operation 1001 = 7-bit 4-byte combined-read master-mode operation, 7-bit 4-byte write-with-IA master-mode operation 1010 = 10-bit 4-byte master-mode operation 1011 = 10-bit 4-byte combined-read master-mode operation 10-bit 4-byte write-with-IA master-mode operation
<56>	R	R/W	—	—	Read bit or result. If this bit is set on a write operation when SLONLY = 0, the operation is a read operation (if clear, it is a write operation). On a read operation, this bit returns the result indication for the most recent master=mode operation, 1 = success, 0 = failure.
<55>	SOVR	R/W	0	—	Size override. If this bit is set, use the SIZE field to determine the master-mode operation size rather than what OP specifies. For operations greater than four bytes, the additional data is contained in MIO_TWS_SW_TWSI_EXT[D].
<54:52>	SIZE	R/W	0x0	—	Size. Specifies the size in bytes of the master-mode operation if SOVR = 1. Specified in –1 notation (i.e. 0 = 1 byte, 1 = 2 bytes, ... 7 = 8 bytes)

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<51:50>	SCR	R/W	0x0	—	Scratch. Unused, but retain state.
<49:40>	A	R/W	0x0	—	Address field. The address of the remote device for a master-mode operation. A<9:7> are only used for 10-bit addressing.
<39:35>	IA	R/W	—	—	Internal address. Used when launching a combined master-mode operation.
<34:32>	EOP_IA	WO	—	—	Extra opcode, used when OP<3:0> = 0110 and SLONLY = 0. 000 = TWSI slave address register (TWSI_SLAVE_ADD) 001 = TWSI data register (TWSI_DATA) 010 = TWSI control register (TWSI_CTL) 011 = (when R = 0) TWSI clock control register (TWSI_CLKCTL) 011 = (when R = 1) TWSI status register (TWSI_STAT) 100 = TWSI extended slave register (TWSI_SLAVE_ADD_EXT) 111 = TWSI soft reset register (TWSI_RST) Also provides the lower 3 bits of internal address when launching a combined master-mode operation.
<31:0>	D	R/W	0x0	—	Data field. Used on a write operation when: • Initiating a master-mode write operation (SLONLY = 0) • Writing a TWSI configuration register (SLONLY = 0) • A slave-mode write operation (SLONLY = 1) The read value is updated by: • A write operation to this register • Master-mode completion (contains error code) • TWSI configuration-register read (contains result)

TWSI to Software Register
MIO_TWS_TWSI_SW

This register allows the TWSI device to transfer data to software and later check that software has received the information.

This register should be read or written by the TWSI device, and read by software. The TWSI device can use one-byte or four-byte payload write operations, and two-byte payload read operations. The TWSI device considers this register valid when V = 0x3. See Table 17–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:62>	V	RC/W	0x0	—	Valid bits. These bits are not directly writable. They are set to 11 on any write operation by the TWSI device. They are cleared to 00 on any read operation by software.
<61:32>	—	RAZ	—	—	Reserved.
<31:0>	D	R/W	—	—	Data field. Updated on a write operation by the TWSI device.

TWSI Interrupt Register
MIO_TWS_INT

This register contains the TWSI interrupt-enable mask and the interrupt-source bits. It also contains a read-only copy of the TWSI bus (SCL and SDA) as well as control bits to override the current state of the TWSI bus (SCL_OVR and SDA_OVR). Setting an override bit to 1 results in the open-drain driver being activated, thus driving the corresponding signal low.

NOTE: The interrupt-source bit for the TWSI core interrupt (CORE_INT) is read-only; the appropriate sequence must be written to the TWSI core to clear this interrupt.

The other interrupt-source bits are write-1-to-clear. TS_INT is set on the update of the MIO_TWS_SW_TWSI register (i.e. when it is written by a TWSI device). ST_INT is set whenever the valid bit of the MIO_TWS_SW_TWSI is cleared (see above for reasons).

See Table 17–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:12>	—	RAZ	—	—	Reserved
<11>	SCL	RO	—	—	SCL signal.
<10>	SDA	RO	—	—	SDA signal.
<9>	SCL_OVR	R/W	0	—	SCL override.
<8>	SDA_OVR	R/W	0	—	SDA override.
<7>	—	RAZ	—	—	Reserved
<6>	CORE_EN	R/W	0	—	TWSI core interrupt-enable. Cleared to 0 when the HLC is enabled.
<5>	TS_EN	R/W	0	—	MIO_TWS_TWSI_SW register-update interrupt-enable. Cleared to 0 when the HLC is disabled.
<4>	ST_EN	R/W	0	—	MIO_TWS_SW_TWSI register-update interrupt-enable. Cleared to 0 when the HLC is disabled.
<3>	—	RAZ	—	—	Reserved
<2>	CORE_INT	RO	0	—	TWSI core interrupt. Ignored when the HLC is enabled.
<1>	TS_INT	R/W1C	0	—	MIO_TWS_TWSI_SW register-update interrupt. Ignored when the HLC is disabled.
<0>	ST_INT	R/W1C	0	—	MIO_TWS_SW_TWSI register-update interrupt. Ignored when the HLC is disabled.

Software to TWSI Extension Register MIO_TWS_SW_TWSI_EXT

This register contains an additional byte of internal address and four additional bytes of data to be used with TWSI master-mode operations.

The IA field is sent as the first byte of internal address when performing master-mode combined-read/write-with-IA operations and MIO_TWS_SW_TWSI[EIA] is set. The D field extends the data field of MIO_TWS_SW_TWSI for a total of 8 bytes (SOVR must be set to perform operations greater than 4 bytes).

See Table 17–14 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:40>	—	RAZ	0x0	—	Reserved.
<39:32>	IA	R/W	0x0	—	Extended internal address. Sent as the first byte of internal address when performing master-mode combined-read/write-with-IA operations and MIO_TWS_SW_TWSI[EIA] is set.
<31:0>	D	R/W	0x0	—	Extended data. Extends the data field of MIO_TWS_SW_TWSI for a total of eight bytes (MIO_TWS_SW_TWSI[SOVR] must be set to 1 to perform operations greater than four bytes).

Chapter 18

System Management Interface (SMI)

This chapter contains the following subjects:

Overview

The SMI is a two-wire, programmable-frequency, serial interface connecting CN50XX to its accompanying ethernet PHYs. It conforms to the IEEE802.3-2005 standard (refer to IEEE Std 802.3-2005, Sections 22.2.4, 22.3.4 and Section 45 cover SMI/MDIO).

The two signal names are:

- MDI_MDC (management data clock – an output clock signal)
- MDI_MDIO (management data input/output – a bidirectional signal exchanging control and status information between the SMI and the PHY.)

18.1 SMI/MDIO Interface

The SMI_EN[EN] bit must be set to 1 to enable any SMI/MDIO transactions and/or to enable any MDI_MDC clock transitions. During the idle time before and after transactions, CN50XX tri-states the MDI_MDIO data wire, and optionally transitions the MDI_MDC clock wire (if SMI_CLK[CLK_IDLE] = 0).

If SMI_CLK[MODE]=1 (i.e. Clause 45 enabled), the Start of Frame field takes on the value 00 and the opcode field can take on any of the four possible values: 00 (address), 01 (write), 11 (read), or 10 post-read-increment address); if SMI0/1_CLK[MODE]=0 (i.e. Clause 22 enabled), the Start of Frame field takes on the value 01 and the opcode field is either 01 (write) or 10 (read).

Figure 18–1 shows an SMI/MDIO transaction mastered by CN50XX.

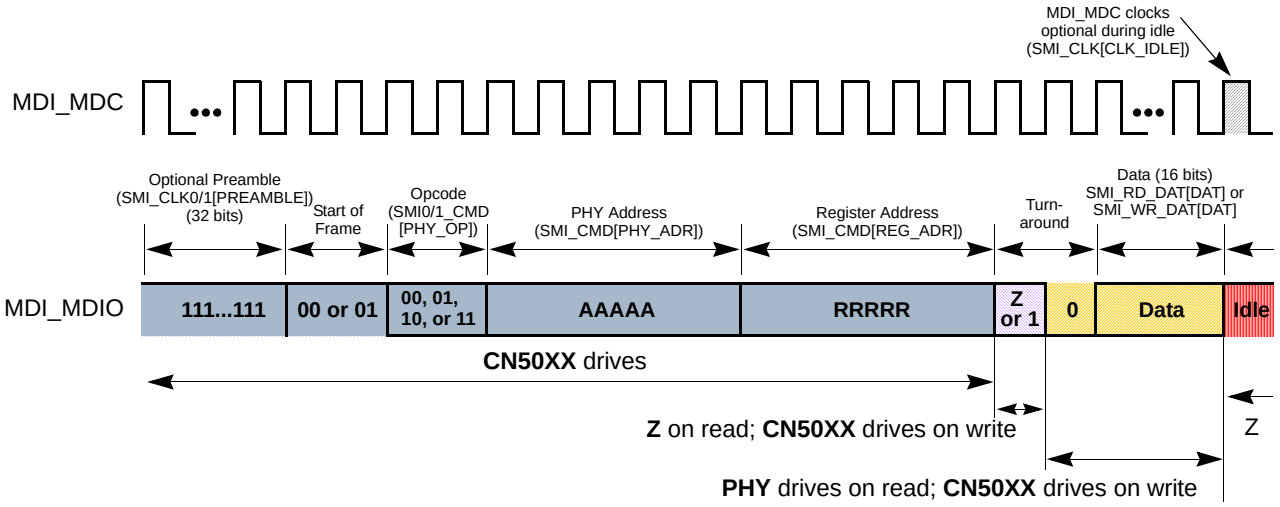


Figure 18–1 SMI/MDIO Transaction as Master

Software initiates a transaction by writing the SMI_CMD register, specifying the following:

- in Clause 22 mode (SMI_CLK[MODE] = 0), read or write operation (SMI_CMD[PHY_OP]); in Clause 45 mode (SMI_CLK[MODE] = 1), address, write, read, or post-read-increment-address operation
- PHY address (SMI_CMD[PHY_ADR])
- register address (SMI_CMD[REG_ADR]).

CN50XX optionally drives the 32-bit preamble (SMI_CLK[PREAMBLE]), then start of frame, operation code, PHY address, register address.

This is followed by the turnaround phase, and only the 16 data cycles remain.

- On a read or a post-read-increment address operation (e.g. SMI_CMD[PHY_OP] = 11 or 10 in Clause 45 mode), the PHY drives (and CN50XX samples) the response for the remaining 16 data cycles.

  According to the 802.3 specification, on read operations, the STA (CN50XX) transitions MDI_MDC and the PHY drives MDI_MDIO with some delay relative to that edge. This is **Edge1** (refer to Figure 18–2).

  The STA then samples MDI_MDIO on the next rising edge of MDI_MDC. This is **Edge2** (refer to Figure 18–2).

  CN50XX can sample the read data relative to Edge2.

- On a write or address operation (e.g. SMI_CMD[PHY_OP] = 1 or 00 in Clause 45 mode), CN50XX drives the write data for the remaining 16 cycles.

After software submits a read or a post-read-increment address operation via a write to SMI_CMD (e.g. with SMI_CMD[PHY_OP] = 11 or 10 in Clause 45 mode), it should poll SMI_RD_DAT for the result. SMI_RD_DAT[VAL] is set and SMI_RD_DAT[DAT] contains the 16-bits of read data when the transaction is complete. CN50XX clears SMI_RD_DAT[VAL] to 0 on a read operation, and SMI_RD_DAT[DAT] contains valid data only when SMI_RD_DAT[VAL] is set to 1. CN50XX asserts SMI_RD_DAT[PENDING] while the transaction is pending.

Software must write the 16-bits of write data to SMI_WR_DAT[DAT] before it initiates a write or address operation via a write to SMI_CMD (e.g. with SMI_CMD[PHY_OP] = 01 or 00 in Clause 45 mode). Software should then poll SMI_WR_DAT[VAL] to tell when the transaction is complete. CN50XX clears SMI_WR_DAT[VAL] to 0 once software has seen it set to 1. CN50XX asserts SMI_WR_DAT[PENDING] while the transaction is pending.

Figures 18–2 and 18–3 indicate timing characteristics of the interface.

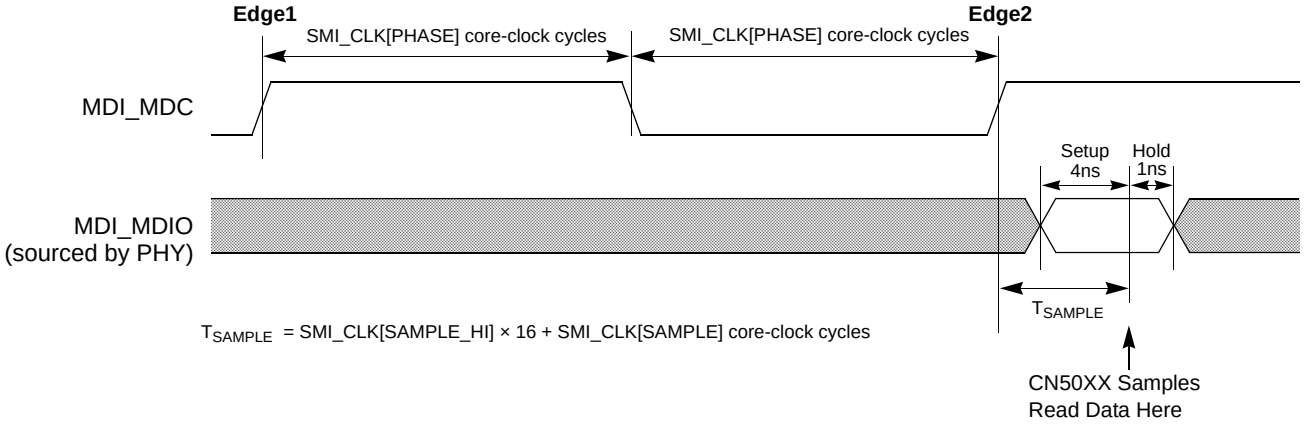


Figure 18–2 SMI/MDIO Interface Read Timing Characteristics

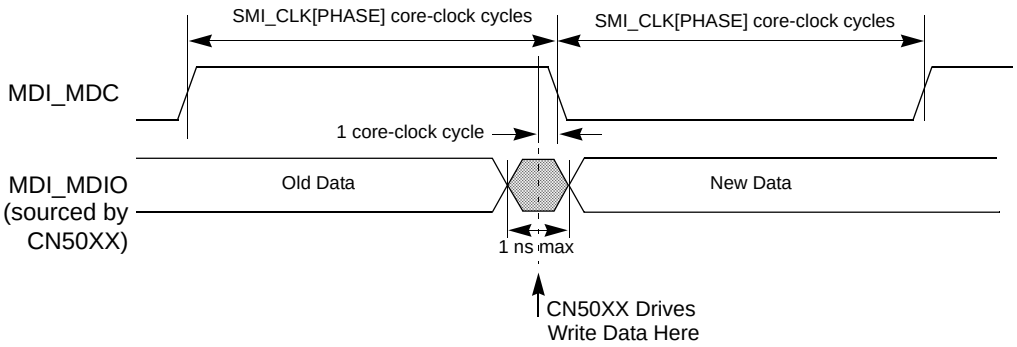


Figure 18–3 SMI/MDIO Interface Write Timing Characteristics

The MDI_MDC frequency is defined as follows:

$$\text{MDI\_MDC frequency} = \frac{\text{(core frequency)}}{\text{(SMI\_CLK[PHASE]} \times 2)}$$

CN50XX always drives write data one core-clock cycle before the falling edge of MDI_MDC, and the sample point for read data is programmable relative to the rising edge of MDI_MDC, based on SMI_CLK[SAMPLE_HI,SAMPLE].

18.2 SMI Registers

The SMI registers are shown in Table 18–1.

Table 18–1 SMI Registers

Register	Address	CSR Type[1]	Detailed Description
SMI0_CMD	0x0001180000001800	RSL	See page 655
SMI0_WR_DAT	0x0001180000001808	RSL	See page 655
SMI0_RD_DAT	0x0001180000001810	RSL	See page 655
SMI0_CLK	0x0001180000001818	RSL	See page 656
SMI0_EN	0x0001180000001820	RSL	See page 656

1. RSL-type registers are accessed indirectly across the I/O Bus.

SMI Command Control Register
SMI_CMD

This register forces a read or write command to the PHY. Write operations to this register create SMI transactions. Software will poll (depending on the transaction type).

See Table 18–1 for the register address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<17:16>	PHY_OP	R/W	0	—	PHY opcode, depending on SMI_CLK[MODE] setting. • If SMI_CLK[MODE] = 0 (≤1Gbs / Clause 22), 0 = write operation, encoded in the frame as 01 1 = read operation, encoded in the frame as 10. • If SMI_CLK[MODE] = 1 (>1Gbs / Clause 45), 00=address, 01=write, 11=read, 10=post-read-increment-address.
<15:13>	—	RAZ	—	—	Reserved.
<12:8>	PHY_ADR	R/W	0x0	—	PHY address.
<7:5>	—	RAZ	—	—	Reserved.
<4:0>	REG_ADR	R/W	0x0	—	PHY register offset.

SMI Write Data Register
SMI_WR_DAT

This register provides the data for a write operation. See Table 18–1 for the register address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:18>	—	RAZ	—	—	Reserved.
<17>	PENDING	RO	0	—	Write transaction pending. Indicates that an SMI write transaction is in flight.
<16>	VAL	RO	0	—	Write data valid. Asserts when the write transaction completes. A read to this register clears VAL.
<15:0>	DAT	R/W	0x0	—	Write data.

SMI Read Data Register
SMI_RD_DAT

This register contains the data in a read operation. See Table 18–1 for the register address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:18>	—	RAZ	—	—	Reserved.
<17>	PENDING	RO	0	—	Read transaction pending. Indicates that an SMI read transaction is in flight.
<16>	VAL	RO	0	—	Read data valid. Asserts when the read transaction completes. A read to this register clears VAL.
<15:0>	DAT	RO	0x0	—	Read data.

SMI Clock Control Register
SMI_CLK

This register determines the SMI timing characteristics. See Table 18–1 for the register address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:25>	—	RAZ	—	—	Reserved.
<24>	MODE	R/W	0	0	IEEE operating mode; 0 = Clause 22 compliant, 1 = Clause 45 compliant.
<23:21>	—	RAZ	—	—	Reserved.
<20:16>	SAMPLE_HI	R/W	0x0	0x0	Sample (extended bits). Specifies in core clock cycles when to sample read data.
<15:14>	—	RAZ	—	—	Reserved.
<13>	CLK_IDLE	R/W	0	0	MDI_MDC toggle. When set, this bit causes MDI_MDC not to toggle on idle cycles.
<12>	PREAMBLE	R/W	1	1	Preamble. When this bit is set, the 32-bit preamble is sent first on SMI transactions. This field must be set to 1 when MODE = 1 in order for the receiving PHY to correctly frame the transaction.
<11:8>	SAMPLE	R/W	0x2	0x2	Sample read data. Specifies the number of core clock cycles after the rising edge of MDI_MDC to wait before sampling read data. (SAMPLE_HI,SAMPLE) > 1 (SAMPLE_HI,SAMPLE) + 3 ≤ 2 × PHASE
<7:0>	PHASE	R/W	0x64	0x64	MDC clock phase. Specifies the number of core clock cycles that make up an MDI_MDC phase. PHASE > 2

SMI Enable Register
SMI_EN

Enables the SMI interface. See Table 18–1 for the register address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:1>	—	RAZ	—	—	Reserved.
<0>	EN	R/W	0	1	SMI/MDIO interface enable: 1 = enable interface 0 = disable interface: no transactions, no MDI_MDC transitions

Chapter 19

Random-Number Generator (RNG), Random-Number Memory (RNM)

This chapter contains the following subjects:

Overview

The CN50XX random-number generator/random-number memory (RNG/RNM) unit performs the following functions:

- generates random bits
- buffers random bits
- returns random bits to cores.

Figure 19–1 shows a block diagram of the random-number generator. The entropy is provided by the jitter of 125 of 128 free-running oscillators XORed into a 128-bit LFSR. The LFSR accumulates the entropy over 81 cycles, after which it is fed into a SHA-1 engine. The SHA-1 engine uses the following 16 32-bit words in order as input:

1. LFSR[31:0]
2. LFSR[63:32]
3. LFSR[95:64]
4. LFSR[127:96]
   (Note that the four LFSR words are all sampled on one cycle.)
5. digest a from the previous SHA-1 iteration
6. digest b from the previous SHA-1 iteration
7. digest c from the previous SHA-1 iteration
8. digest d from the previous SHA-1 iteration
9. digest e from the previous SHA-1 iteration
10. 0x80000000
11. 0x00000000
12. 0x00000000
13. 0x00000000
14. 0x00000000
15. 0x00000000
16. 0x00000120

The SHA-1 engine runs once every 81 cycles.

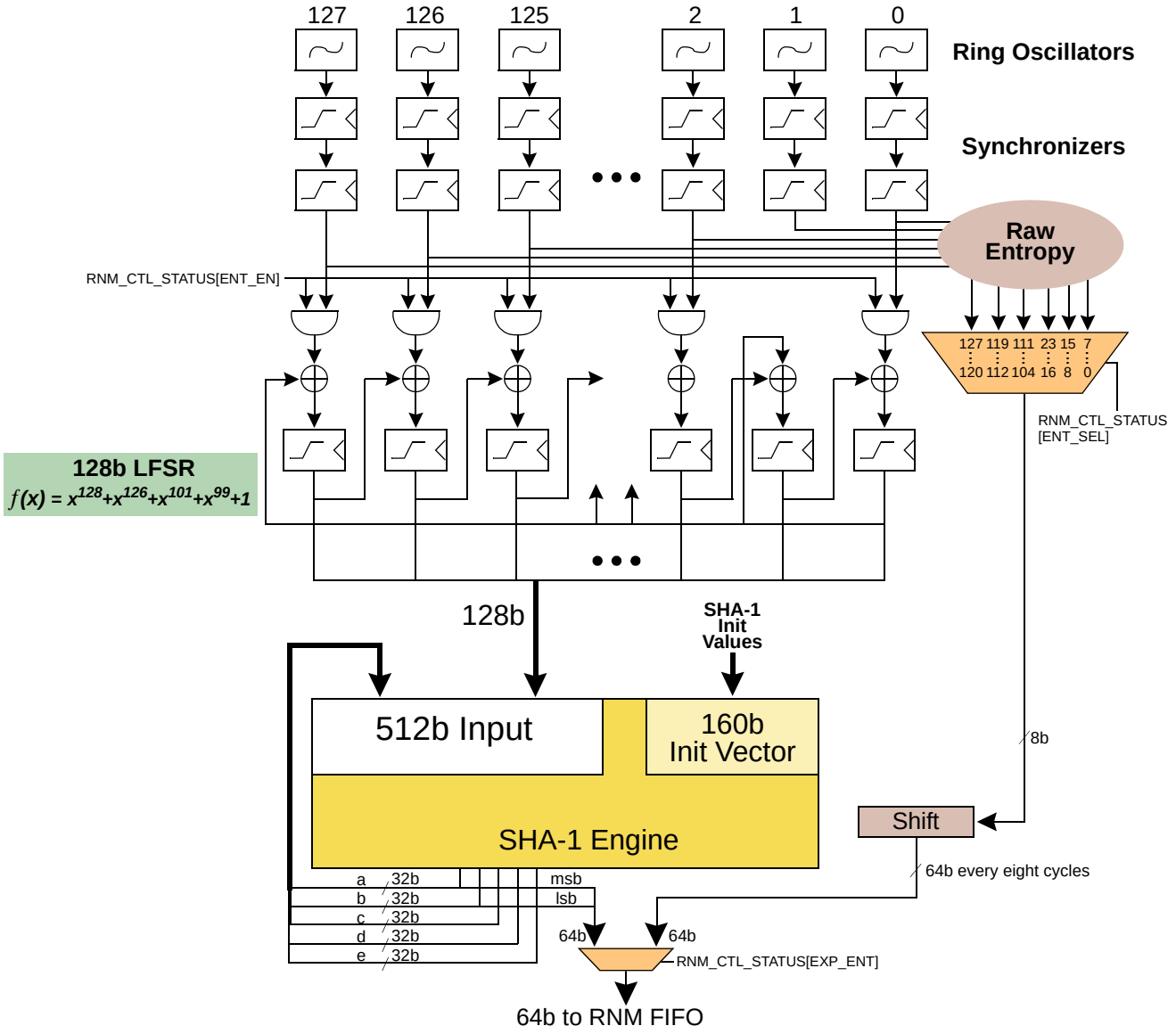


Figure 19–1 RNG Diagram

The SHA-1 engine uses the 512-bit input with its 160-bit initialization vector to run its algorithm. After completing the SHA-1 operation, the hardware sends the first 64 bits of the digest to the RNM FIFO. The hardware produces new 64-bit random number every 81 cycles.

The RNM FIFO can also be sourced from the raw entropy of the 128 oscillators. The entropy is divided into 16 groups of eight bits (0-7, 8-15, 16-23, ..., 112-119, 120-127) that are fed into a multiplexer, whose output (selected by RNM_CTL_STATUS[ENT_SEL]) is fed into a shift register, which produces the 64-bit input to the RNM FIFO every eight cycles.

The input to the RNM FIFO is determined by RNM_CTL_STATUS[EXP_ENT].

The RNG/RNM unit feeds the random numbers from the unit depicted in Figure 19–1 into a 512-byte FIFO eight bytes at a time. Load operations from the cores extract random bytes from the FIFO to drain it. Whenever the FIFO drains this way, RNG/RNM refills using the random numbers constantly generated by the hardware.

NOTE: Any load operation loads eight bytes of random numbers regardless of the type of load issued.

For chip-test purposes, the entropy source can be disabled from being fed into the LFSR mixing function, to ensure predictability of the random-number generator output. The entropy source is normally enabled by setting the RNM_CTL_STATUS[ENT_EN] bit. On power-up, this bit is clear, which ensures the entropy source is disabled.

When RNM_CTL_STATUS[RNG_EN] = 1, the random-number generator is enabled.

When RNM_CTL_STATUS[RNG_EN] = 0,

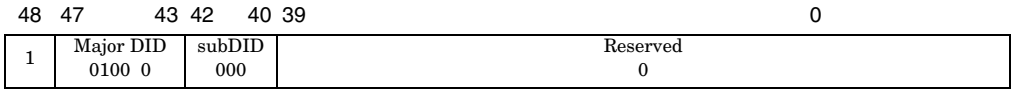
- no random numbers can be delivered to any cores
- the FIFO is cleared
- the LFSR mixing function is forced to its reset value (00…1).

If RNM_CTL_STATUS[ENT_EN] = 0, a predictable 512 bytes of numbers fill the FIFO after RNM_CTL_STATUS[RNG_EN] transitions from 0 to 1. Consequently, core operations can receive a predictable sequence of random numbers after enabling RNG/RNM when the entropy source is disabled.

19.1 RNG/RNM Operations

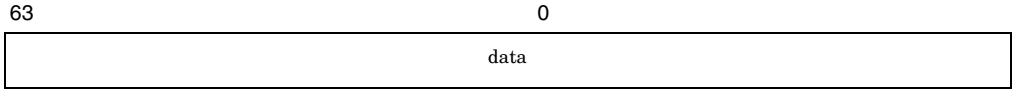
19.1.1 RNG/RNM Load Operation

Load Address Field



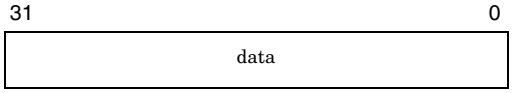
Load Result Field

64-bit operation result



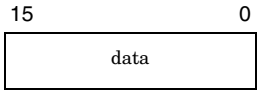
- **data** - eight random bytes.

32-bit operation result



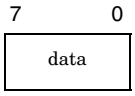
- **data** - four random bytes.

16-bit operation result



- **data** - two random bytes.

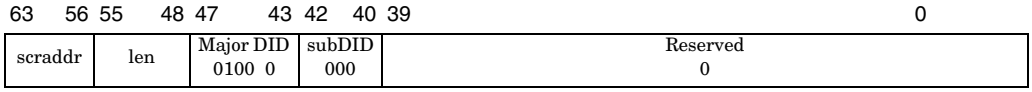
8-bit operation result



- **data** - random byte.

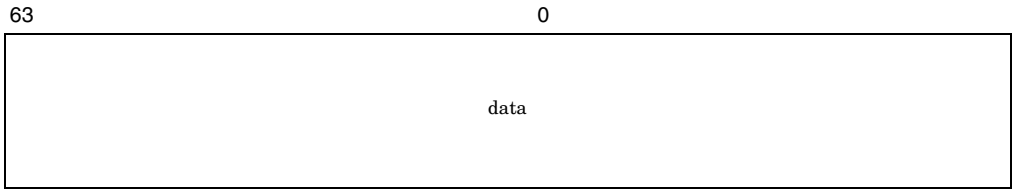
19.1.2 IOBDMA Operations

IOBDMA Address Field



- **scraddr** - Defined in “cnMIPS™ Cores” on page 143.
- **len** - Can be any value (1 to 255 8-byte words). Defined in “cnMIPS™ Cores” on page 143.

IOBDMA Result Field



- **data** - [8 × len] random bytes.

19.2 RNM Registers

The RNM registers are shown in Table 19–1.

Table 19–1 RNM Registers

Register	Address	CSR Type[1]	Detailed Description
RNM_CTL_STATUS	0x0001180040000000	RSL	See page 662
RNM_BIST_STATUS	0x0001180040000008	RSL	See page 662

1. RSL-type registers are accessed indirectly across the I/O Bus.

RNM BIST Status Register
RNM_BIST_STATUS

This register is the RNM memory BIST status register, indicating status of built-in self-tests. See Table 19–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:2>	—	RAZ	—	—	Reserved
<1>	RRC	RO	0	0	Status of the RRC memory block BIST: 1 = failed BIST, 0 = passed BIST
<0>	MEM	RO	0	0	Status of MEM block BIST: 1 = failed BIST, 0 = passed BIST

RNM Control and Status Register
RNM_CTL_STATUS

This register is the RNM interrupt-enable register. See Table 19–1 for address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:9>	—	RAZ	—	—	Reserved
<8:5>	ENT_SEL	R/W	0	0	Select input to RNM FIFO. 0x0 = 0–7 0x1 = 8–15 0x2 = 16–23 0x3 = 24–31 0x4 = 32–39 0x5 = 40–47 0x6 = 48–55 0x7 = 56–63 0x8 = 64–71 0x9 = 72–79 0xA = 80–87 0xB = 88–95 0xC = 96–103 0xD = 104–111 0xE = 112–119 0xF = 120–127
<4>	EXP_ENT	R/W	0	0	Exported entropy enable for random-number generator.
<3>	RNG_RST	R/W	0	0	Reset RNG as core reset.
<2>	RNM_RST	R/W	0	0	Reset the RNM as core reset except for register logic.
<1>	RNG_EN	R/W	0	0	Enables the output of the RNG.
<0>	ENT_EN	R/W	0	0	Entropy enable for random number generator.

Chapter 20

MPI/SPI Unit

This chapter contains the following subjects:

Overview

The multiprocessor peripheral interface (MPI)/serial peripheral interface (SPI) is a highly flexible implementation capable of connecting to many variations on the SPI. The key features are the following:

- Clock generation to 16 MHz
- Ability to tristate transmit data wire for MPI-style single-wire transmit/receive
- MSB- or LSB-first transmission/reception
- Interrupt- or polling-based transactions
- Continuous clocking mode
- Flexible timing between commands
- Optional integrated chip-select

An overview of the interface is shown in Figure 20–1.

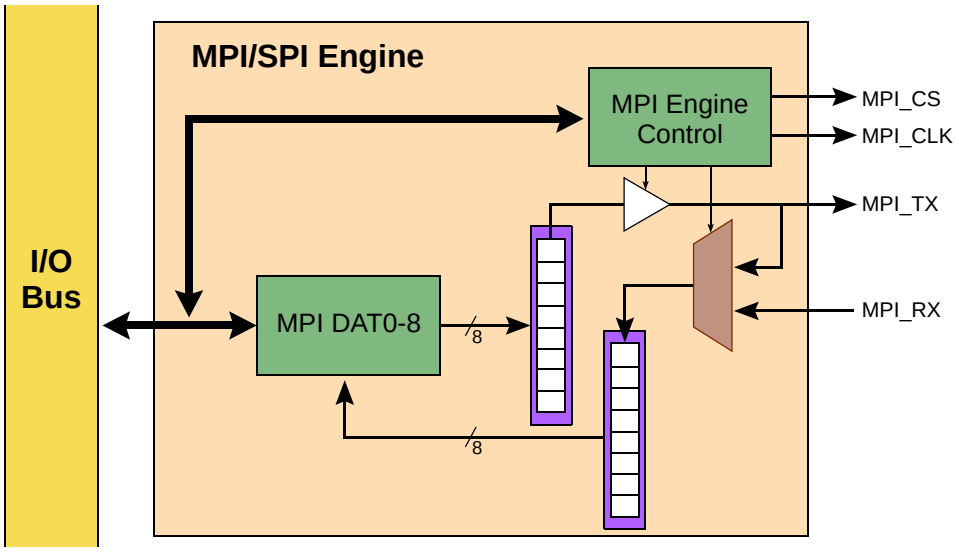


Figure 20–1 MPI/SPI Overview

20.1 Pin Usage

All the MPI/SPI pins, when not enabled for MPI/SPI use, function as GPIO pins. Refer to Chapter 15 for more details. Table 20–1 describes the MPI/SPI pins and the CSR fields that switch them from standard GPIO pins to their MPI/SPI functionality. In addition, GPIO_CFG*n*[TX_OE] must be 0 for any of these pins when using the MPI/SPI functionality associated with the given pin.

Table 20–1 MPI/SPI Signals

Pin Name	Description	Enable
GPIO_23/MPI_CLK	Serial interface clock (SCLK)	MPI_CFG[ENABLE] = 1
GPIO_22/MPI_CS	Chip select	MPI_CFG[CSENA] = 1
GPIO_21/MPI_TX	Transmit data wire	MPI_CFG[ENABLE] = 1
GPIO_20/MPI_RX	Receive data wire	Always received

For the remainder of the chapter the MPI/SPI signals will be referred to by the MPI/SPI part of their names.

20.2 MPI/SPI Configuration

20.2.1 Clock Generation

The MPI/SPI unit operates in master mode at all times. As such, it is responsible for generating SCLK for the MPI/SPI interface, which is accomplished through a simple clock divider.

MPI_CFG[CLKDIV] determines the MPI_CLK frequency using the following relation:

$$F_{SCLK} = \frac{F_{ECLK}}{2 \times MPI\_CFG[CLKDIV]}$$

In addition to the clock frequency, the MPI/SPI interface has two types of clocking strategies.

- When MPI_CFG[CLK_CONT] = 0, the MPI_CLK is stopped between MPI/SPI transactions. In this mode, MPI_CS is not required to deassert between transactions.
- When MPI_CFG[CLK_CONT] = 1, MPI_CLK runs continuously. In this mode, deassertion of MPI_CS is the only notification to the slave device that a transaction has completed.

MPI_CFG[IDLELO] specifies the value at which MPI_CLK idles when the clock is stopped.

20.2.2 Chip Select

Three fields in MPI_CFG control the handling of the MPI_CS signal: [CSENA], [CSLATE], and [CSHI]

- [CSENA] determines whether the MPI/SPI engine drives MPI_CS during MPI/SPI transactions.
  - If [CSENA] = 0, the MPI/SPI engine never drives MPI_CS.
  - If [CSENA] = 1, then MPI_CS is asserted at the start of every MPI transaction based on the settings of MPI_CFG[CSHI,CSLATE] and deasserted (or not) based on the value written to MPI_TX[LEAVECS].
- [CSHI] determines whether MPI_CS is treated as a low-asserted chip select ([CSHI] = 0) or a high-asserted chip select ([CSHI] = 1).
- [CSLATE] is provided to handle different device setup requirements for chip select.
  - If [CSLATE] = 0, the MPI/SPI engine asserts chip select one-half SCLK cycle before MPI_TX is driven.
  - If [CSLATE] = 1, chip select asserts coincident with MPI_TX driving.

A timing diagram displaying the relationship between MPI_CS and MPI_CFG[CSLATE] is shown in Figure 20–2.

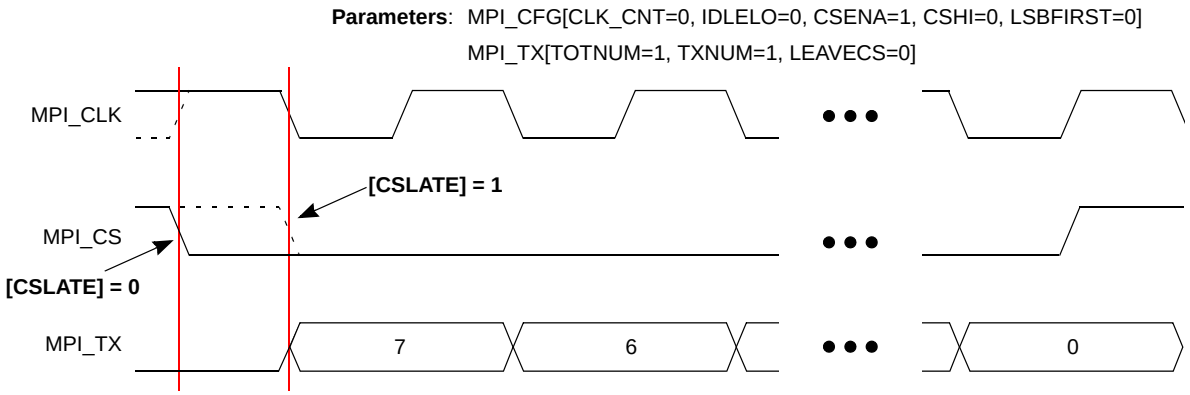


Figure 20–2 MPI_CS and MPI_CFG[CSLATE]

20.2.3 SPI/MPI Style

Typically, SPI uses separate TX and RX wires, while MPI uses a single wire for both TX and RX. MPI_CFG[WIREOR] tells the MPI/SPI engine which style to use.

- If MPI_CFG[WIREOR] = 0 (SPI style), data is sampled from MPI_RX and driven onto MPI_TX. In this case, MPI_TX is always driven when the MPI/SPI engine is enabled.
- If MPI_CFG[WIREOR] = 1 (MPI style), data is driven on MPI_TX but MPI_TX is tristated when the MPI/SPI engine expects the slave to drive data and data is sampled from MPI_TX. In this case the MPI_RX pin is not used at all.

  In addition, the MPI_CFG[TRITX] setting determines the following

  0 = the MPI/SPI engine drives MPI_TX during idle cycles (i.e. cycles in which the slave is not expected to drive)

  1 = the MPI/SPI engine tristates MPI_TX

20.2.4 Polling/Interrupt-Based Reception

MPI_CFG[INT_ENA] provides a mechanism to interrupt the processor upon completion of an MPI/SPI transaction.

1 = the MPI/SPI engine interrupts at the completion of an MPI/SPI transaction.

0 = software must poll MPI_STS[BUSY] to determine when the MPI/SPI transaction is complete.

20.2.5 Other Fields in MPI_CFG

MPI_CFG[IDLECLKS] specifies the number of idle SCLK cycles to be inserted between MPI/SPI transactions.

- When cleared to 0, it allows two transactions to follow as closely to each other as the protocol allows (i.e. it does not insert any idle cycles).
- When set to a non-zero value (1, 2, or 3), that many idle SCLK cycles are inserted between MPI/SPI transactions.

MPI_CFG[LSBFIRST] determines whether the LSB or MSB of a byte is sent first.

MPI_CFG[ENABLE] enables the MPI/SPI engine. When this field is cleared to 0, the engine is disabled and the pins are GPIO pins.

20.3 MPI/SPI Usage

Sending an MPI/SPI transaction requires the following steps:

1. Configure the interface via MPI_CFG.
2. Write transmission bytes to MPI_DAT0-8.
3. Write MPI_TX with transaction information.
4. Wait for interrupt/poll MPI_STS.
5. Read MPI_DAT0-8 to get received data

20.3.1 MPI_DAT(0..8) Registers

The MPI_DAT*n* registers hold the bytes to transmit and are written with the received bytes for any transaction performed by the MPI/SPI engine.

The value in MPI_DAT0 is the first byte transmitted, as well as the first register written with received data.

20.3.2 Using the MPI_TX Register

The MPI_TX register is written with transaction information to tell the MPI/SPI engine to start a transaction.

MPI_TX[TOTNUM] specifies the number of bytes in the transaction (transmit plus receive), while MPI_TX[TXNUM] specifies the number of bytes to transmit. MPI_TX[TOTNUM] should not exceed nine and typically is greater than one.

- An MPI/SPI transaction intended to write a register in a slave device would typically have [TOTNUM] = [TXNUM].
- An MPI/SPI transaction intended to read a register in a slave device would typically have [TOTNUM] > [TXNUM].

There are some devices that can support multibyte read/write operations larger than MPI_DAT0-8 can handle. For those devices, setting the MPI_TX[LEAVECS] bit tells the MPI/SPI engine to leave MPI_CS asserted after completing the transaction. This allows a subsequent MPI_TX write operation to continue the transaction from the slave device's perspective. This method only works in noncontinuous clocking mode (i.e. MPI_CFG[CLK_CONT] = 0).

20.3.3 Using the MPI_STS Register

Software may read the MPI_STS register to determine the status of the MPI/SPI engine.

- MPI_STS[BUSY] reads 1 if the MPI/SPI engine is currently processing a transaction.
- MPI_STS[RXNUM] reflects the number of bytes received during the transaction. If [BUSY] = 1, [RXNUM] reflects the number of bytes received so far.

Note that the MPI/SPI engine receives bytes the entire time it is shifting. Because of this MPI_STS[RXNUM] always equals MPI_TX[TOTNUM] at the end of a transaction.

20.4 Examples

The following four examples show the steps involved in reading and writing the MPI/SPI unit.

20.4.1 Example 1: Reading a Single Byte From Device Address 0x04

Assumptions: device expects first byte transmitted to be a command byte. Bit[7] of the command byte determines whether the transaction is a read (0) or write (1).

1. Write 0x04 to MPI_DAT0
   a. [7] = 0 indicates read (device specific)
   b. [6:0] = 0x04 indicates address 0x04
2. Write MPI_TX with [TOTNUM] = 0x2, [TXNUM] = 0x1, [LEAVECS] = 0

   Meaning: shift for a total of two bytes, transmitting during the first byte only, deassert MPI_CS after the last data bit is received.
3. Read MPI_STS until [BUSY] = 0 (or wait for an interrupt)
4. Read MPI_DAT1 to get the data byte sent from the slave device.

NOTE: MPI_DAT0 contains junk because it was latched while shifting the read command into the device. The second byte shifted from the device is the response to the read command.

20.4.2 Example 2: Writing a Single Byte to Register 0x04

Assumptions: device expects first byte transmitted to be a command byte. Bit[7] of the command byte determines whether the transaction is a read (0) or write (1).

1. Write 0x84 to MPI_DAT0
   a. [7] = 1 indicates write.
   b. [6:0] = 0x04 indicates address 0x04.
2. Write a byte to MPI_DAT1. This is the value to write to address 0x04.
3. Write MPI_TX with [TOTNUM] = 0x2, [TXNUM] = 0x2, [LEAVECS] = 0.

   Meaning: shift for a total of two bytes, transmitting during both bytes, deassert MPI_CS after the last data bit.
4. Read MPI_STS until [BUSY] = 0 (or wait for an interrupt).

20.4.3 Example 3: Writing Ten Bytes to Registers 0x09–0x00

This is a mode used by some devices that allows reading/writing from register N down to register 0 with one MPI command.

The first data byte is for register N, the second is for register N-1, etc. In this case, an MPI_CS deassertion aborts the transaction, so special care is taken here to not deassert MPI_CS during the transfer. Note that a maximum of eight bytes can be written at once because there are only nine MPI_DAT*n* registers, and one must be used for the command to start the transaction. So this operation must be split into two transactions.

NOTE: This can not work in continuous clocking mode. SCLK must idle between the two transactions or the data will be shifted out before the second transaction is ready to receive it. In continuous clocking mode a maximum of eight bytes may be read or written with one MPI/MPI command.

Assumptions: device expects first byte transmitted to be a command byte. Bit[7] of the command byte determines whether the transaction is a read (0) or write (1).

1. Write 0x89 to MPI_DAT0
   a. [7] = 1 indicates write.
   b. [6:0] = 0x09 indicates address 0x09.
2. Write eight bytes to MPI_DAT1–8. These are the bytes to write to addresses 0x09, 0x08, 0x07, 0x06, 0x05, 0x04, 0x03, and 0x02.
3. Write MPI_TX with [TOTNUM] = 0x9, [TXNUM] = 0x9, [LEAVECS] = 1.

   Meaning: shift for a total of nine bytes, transmitting during all bytes, do not deassert MPI_CS after the completion of the nine bytes.
4. Read MPI_STS until [BUSY] = 0 (or wait for an interrupt).
5. Write two bytes to MPI_DAT0–1. These are the bytes to write to addresses 0x01 and 0x00.
6. Write MPI_TX with [TOTNUM] = 0x2, [TXNUM] = 0x2, [LEAVECS] = 0.

   Meaning: shift for a total of two bytes, transmitting during both bytes, deassert MPI_CS after the last data bit.
7. Read MPI_STS until [BUSY] = 0 (or wait for an interrupt).

20.4.4 Example 4: Reading 17 Bytes From Registers 0x11–0x00

This is the reading equivalent of example 3.

NOTE: This can not work in continuous clocking mode. SCLK must idle between the two transactions or the data will be shifted out before the second transaction is ready to receive it. In continuous clocking mode a maximum of eight bytes may be read or written with one MPI/MPI command.

Assumptions: device expects first byte transmitted to be a command byte. Bit[7] of the command byte determines whether the transaction is a read (0) or write (1).

1. Write 0x11 to MPI_DAT0
   a. [7] = 0 indicates read.
   b. [6:0] = 0x11 indicates address 0x11 (17).
2. Write MPI_TX with [TOTNUM] = 0x9, [TXNUM] = 0x1, [LEAVECS] = 1.

   Meaning: shift for a total of nine bytes, transmitting only during the first byte, do not deassert MPI_CS after completion of the nine bytes.
3. Read MPI_STS until [BUSY] = 0 (or wait for an interrupt).
4. Read MPI_DAT1–8 to get the first eight data bytes.

NOTE: MPI_DAT0 contains junk because it was latched while shifting the read command into the device. The second byte shifted from the device is the response to the read command.

5. Write MPI_TX with [TOTNUM] = 0x9, [TXNUM] = 0x0, [LEAVECS] = 0.

   Meaning: shift for a total of nine bytes, never transmitting, deassert MPI_CS after completion of the transaction.
6. Read MPI_STS until [BUSY] = 0 (or wait for an interrupt).
7. Read MPI_DAT0–8 to get the next nine bytes. Because this transaction was simply receiving data from the end of a previously started command, all nine data bytes are valid.

20.5 MPI/SPI Registers

The MPI/SPI registers are listed in Table 20–2.

Table 20–2 MPI/SPI Registers

Register	Address	CSR Type[1]	Detailed Description
MPI_CFG	0x0001070000001000	NCB	See page 672
MPI_STS	0x0001070000001008	NCB	See page 673
MPI_TX	0x0001070000001010	NCB	See page 673
MPI_DAT0 ... MPI_DAT8	0x0001070000001080 ... 0x00010700000010C0	NCB	See page 673

1. NCB-type registers are accessed directly across the I/O Bus.

MPI Configuration Register
MPI_CFG

This register provides configuration for the MPI interface. See Table 20–2 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:29>	—	RAZ	—	—	Reserved.
<28:16>	CLKDIV	R/W	0x0	0x0	Clock divisor. FSCLK = FECLK / (2 × CLKDIV) CLKDIV = FECLK / (2 × FSCLK)
<15:12>	—	RAZ	—	—	Reserved.
<11>	CSLATE	R/W	0	0	MPI_CS late. Used only when CSENA = 1 1 = MPI_CS asserts coincident with the transaction. 0 = MPI_CS asserts ½ SCLK cycle before the transaction.
<10>	TRITX	R/W	0	0	Tristate TX. Used only when WIREOR = 1 1 = MPI_TX pin is tristated when not transmitting. 0 = MPI_TX pin is driven when slave is not expected to be driving.
<9:8>	IDLECLKS	R/W	0x0	0x0	Idle clocks. When set, guarantees idle SCLK cycles between commands.
<7>	CSHI	R/W	0	0	MPI_CS high: 1 = MPI_CS is asserted high, 0 = MPI_CS is asserted low.
<6>	CSENA	R/W	0	1	MPI_CS enable: 1 = MPI_CS is driven per MPI_TX instruction 0 = MPI_CS is a GPIO pin, not used by MPI_TX
<5>	INT_ENA	R/W	0	0	MPI interrupt enable: 1 = MPI engine interrupts at the end of the transaction 0 = polling is required
<4>	LSBFIRST	R/W	0	0	Shift LSB first: 1 = shift LSB first, 0 = shift MSB first.
<3>	WIREOR	R/W	0	0	Wire OR TX and RX. 1 = MPI_TX handles both transmit and receive and is tristated when not transmitting (MPI mode); MPI_RX pin is not used by the MPI engine. 0 = MPI_TX and MPI_RX are separate wires (SPI mode); MPI_TX pin is always driven.
<2>	CLK_CONT	R/W	0	0	Clock control. 1 = clock never idles, requires MPI_CS deassertion/assertion between commands. 0 = clock idles to value given by IDLELO after completion of MPI transaction.
<1>	IDLELO	R/W	0	0	Clock idle low. 1 = MPI_CLK idles low, first transition is low-to-high. 0 = MPI_CLK idles high, first transition is high-to-low.
<0>	ENABLE	R/W	0	0	MPI enable. 1 = MPI signals are driven, 0 = all MPI pins are GPIO signals.

MPI STS Register
MPI_STS

See Table 20–2 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:13>	—	RAZ	—	—	Reserved.
<12:8>	RXNUM	RO	0x0	0x0	Number of bytes written for the transaction.
<7:1>	—	RAZ	—	—	Reserved.
<0>	BUSY	RO	0	0	Busy. 1 = MPI engine is processing a transaction. 0 = no MPI transaction in progress.

MPI Transmit Register
MPI_TX

See Table 20–2 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:17>	—	WO	—	—	Reserved.
<16>	LEAVECS	WO	—	0x0	Leave MPI_CS asserted. 1 = leave MPI_CS asserted after the transaction is completed. 0 = deassert MPI_CS asserted after the transaction is completed.
<15:13>	—	WO	—	—	Reserved.
<12:8>	TXNUM	WO	—	0x1	Number of bytes to transmit.
<7:5>	—	WO	—	—	Reserved.
<4:0>	TOTNUM	WO	—	0x2	Total number of bytes to shift (transmit and receive).

MPI Data Registers
MPI_DAT(0..8)

See Table 20–2 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:8>	—	RAZ	—	—	Reserved.
<7:0>	DATA	R/W	—	—	Data to transmit/receive.

Chapter 21

USB Unit (USB)

This chapter contains the following subjects:

Overview

The USB interface is a dual-role device (DRD) controller that supports both host and device functions and is fully compliant with the USB 2.0 specification. The USB controller (USBC) supports the following features:

- USB 2.0 compliant.
- Host and device modes (depending on whether it is attached to an A or B connector respectively).
- Operates in high-speed (HS, 480Mbs), full-speed (FS, 12-Mbs), and low-speed (LS, 1.5-Mbs) modes (LS is not supported when the USB core is operating as a device).
- Supports up to eight host channels.
- Supports up to four bidirectional endpoints, including control endpoint 0.
- Supports periodic transfers in host and device mode.
- Supports slave mode.
- Supports dynamic FIFO sizing for application-specific configurations.
- Supports a generic root hub.
- Includes automatic ping capabilities.
- Supports big- and little-endian modes.

This implementation does not currently support the On-The-Go Supplement to the USB 2.0 specification or the Enhanced Host Controller Interface Specification (EHCI 1.0).

21.1 Architecture

This section describes the general USB controller (USBC) architecture and its major components, including its host architecture, its device architecture, its address map, and protocol and transaction handling.

21.1.1 Host Architecture

This section describes the USB controller when it is operating in host mode.

The host uses one transmit FIFO for all nonperiodic OUT transactions and one transmit FIFO for all periodic OUT transactions. These transmit FIFOs are used as transmit buffers to hold the data (payload of the transmit packet) to be transmitted over USB. The host pipes the USB transactions through request queues (one for periodic and one for nonperiodic). Each entry in the request queue holds the IN or OUT channel number along with other information to perform a transaction on the USB. The order in which the requests are written into the queue determines the sequence of transactions on the USB. The host processes the periodic request queue first, followed by the nonperiodic request queue, at the beginning of each microframe.

The host uses one receive FIFO for all periodic and nonperiodic transactions. The FIFO is used as a receive buffer to hold the received data (payload of the received packet) from the USB until it is transferred to the system memory. The status of each packet received also goes into the FIFO. The status entry holds the IN channel number along with other information, such as received byte count and validity status.

Figure 21–1 shows the bus interface architecture of the USBC in host mode.

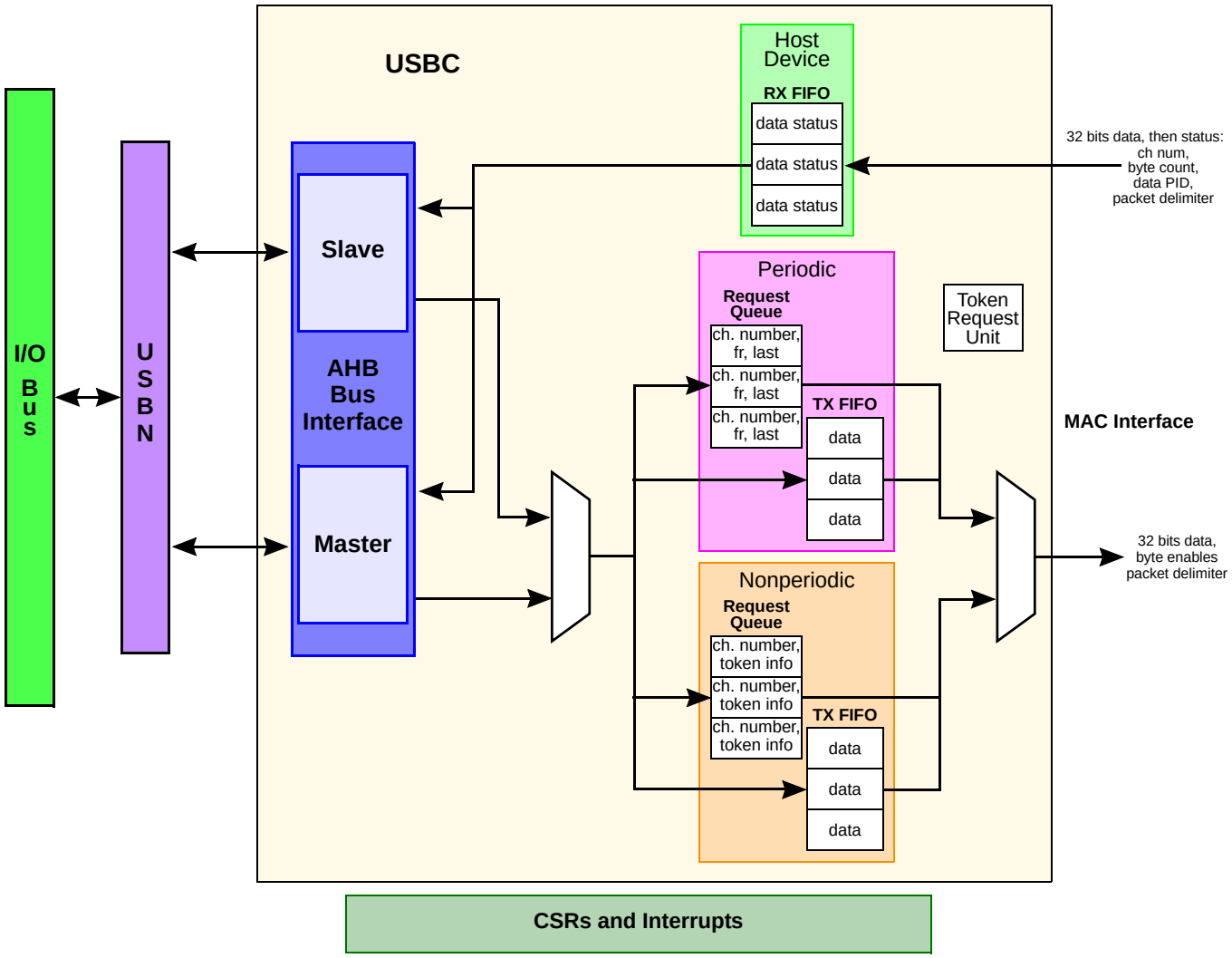


Figure 21–1 Bus Interface Block Diagram (Host Mode)

21.1.2 Device Architecture

The USB device uses a single transmit FIFO to store data for all nonperiodic endpoints, and one transmit FIFO per periodic endpoint to store data to be transmitted in the next microframe. The data is fetched by the DMA engine or is written by the application into the transmit FIFOs and is transmitted on the USB when the IN token is received. The request queue contains the number of the endpoint for which the data is written into the data FIFO.

To improve performance, the application can use the learning queue to help predict the order in which the USB host will access the nonperiodic endpoints and writes the data into the nonperiodic FIFO accordingly. Since each periodic IN endpoint has its own FIFO, no order prediction is needed for periodic IN transfers.

The USB device uses a single receive FIFO to receive the data and status for all OUT endpoints. The status of the packet includes the size of the received OUT data packet, data PID, and validity of the received data. The data in the receive FIFO is read by the application when the data is received.

Figure 21–2 shows the bus interface architecture of the USBC in device mode.

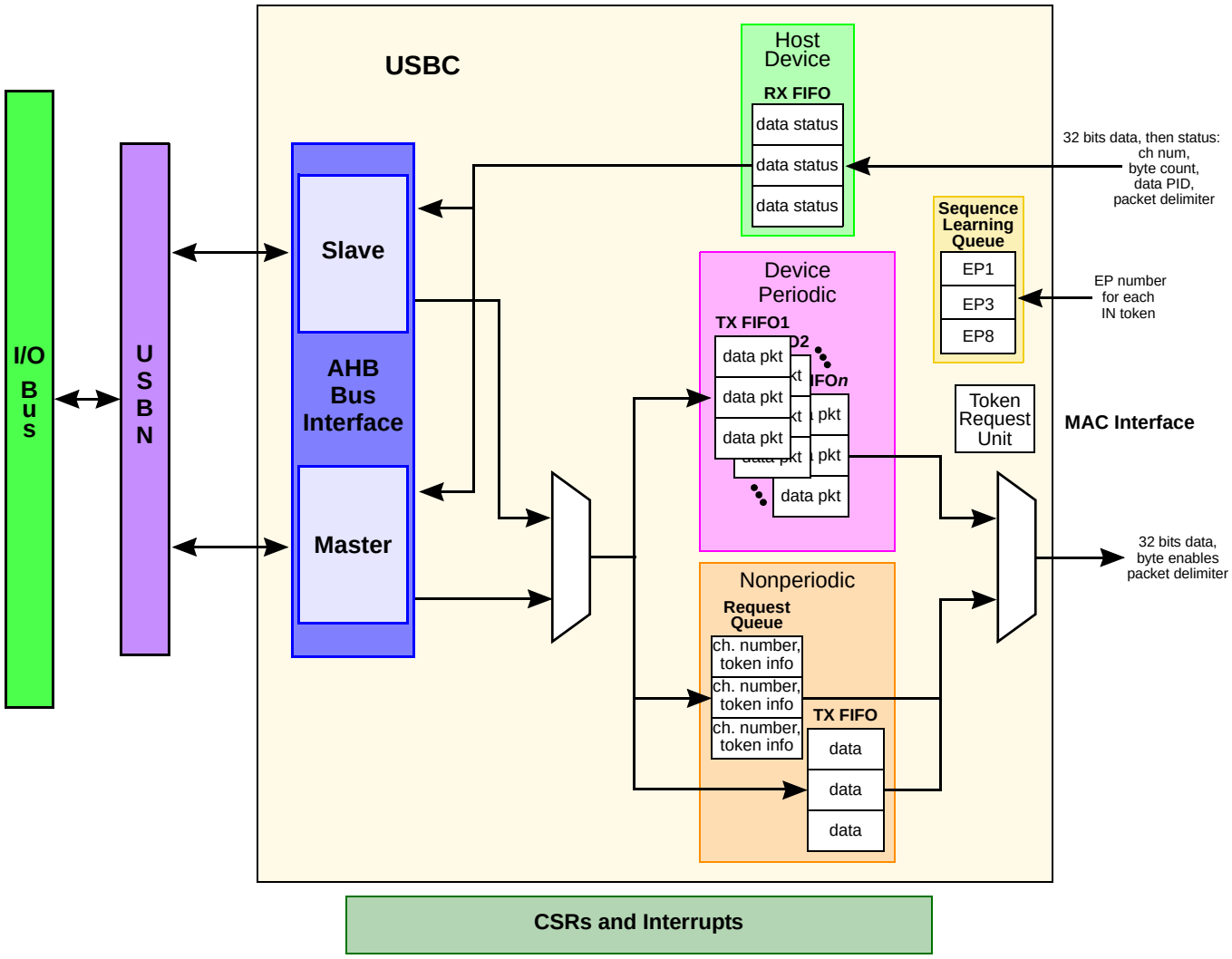


Figure 21–2 Bus Interface Block Diagram (Device Mode)

21.1.3 Address Map

To make the software transparent to the hardware implementation, one region is mapped to each host channel or device endpoint. The controller has the following elements:

- One common RxFIFO, used in host and device modes.
- One common nonperiodic TxFIFO, used in host and device modes
- One common periodic TxFIFO, used in host mode
- One periodic TxFIFO for each periodic IN endpoint in device mode

Within the controller, the following occur:

- Read accesses to any one of the 4 KB regions are mapped to the RxFIFO.
- Write operations to any nonperiodic IN endpoint or OUT channel are mapped to the nonperiodic TxFIFO.
- In host mode, write operations to any periodic OUT channel are mapped to the common periodic TxFIFO.
- In device mode, write accesses to a periodic IN endpoint are mapped to the corresponding endpoints periodic TxFIFO (Bits <30:27> of a device endpoint control register map an endpoint to a specific device periodic FIFO).

In device mode, an IN interrupt endpoint can either be mapped to the common non-periodic TxFIFO or assigned to a separate periodic TxFIFO. When the device endpoint-*n* control register's TxFIFO Number field is 0, the endpoint is mapped either to the common nonperiodic TxFIFO or to the FIFO number selected by these fields.

Hardware maintains the periodic and nonperiodic Tx queues for internal operation. For debugging, software can read the top of the queue information. Because these queues' read domain is in the PHY domain, no debug pop access is provided to these queues' saving area; access is provided only to the top of the queue.

In device mode, because each periodic IN endpoint has a separate buffer allocated to it that holds only one packet at a time, there is no device-mode periodic queue.

Figure 21–3 shows the host-mode FIFO address mapping.

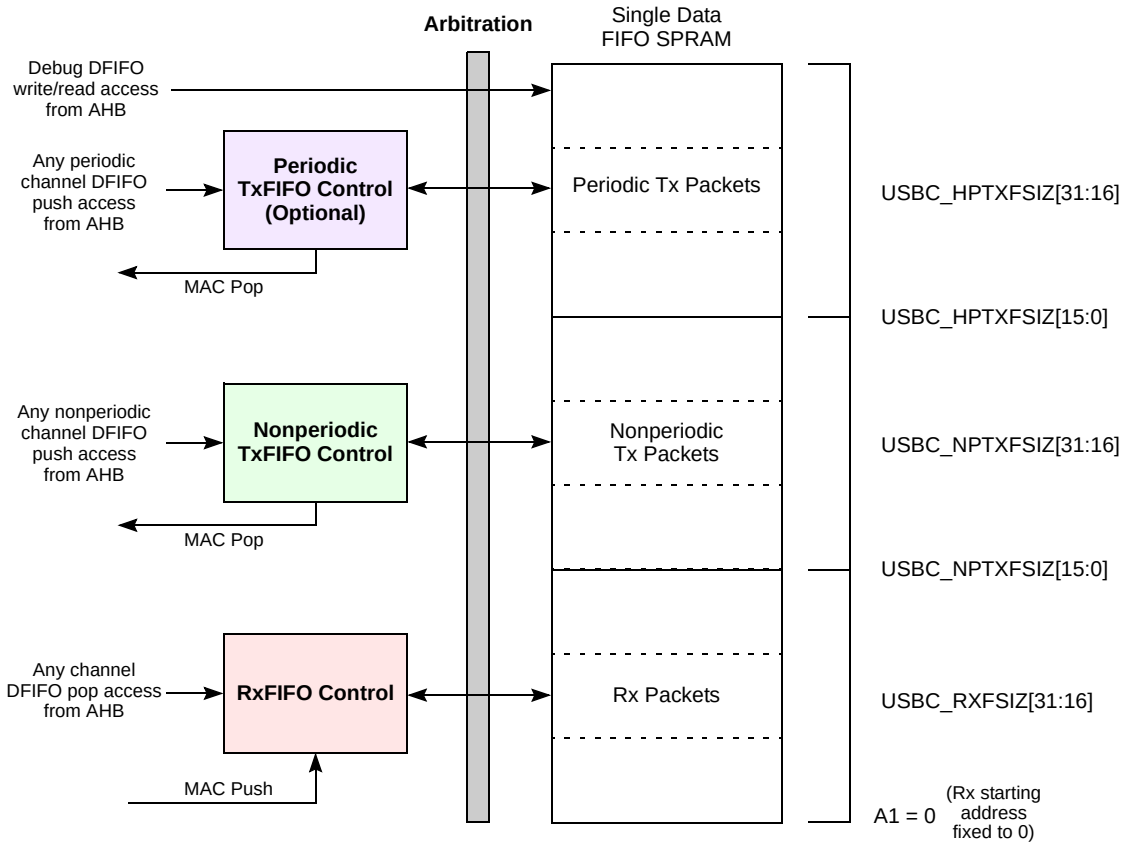


Figure 21–3 FIFO Mapping (Host Mode)

Figure 21–4 shows the device-mode FIFO address mapping.

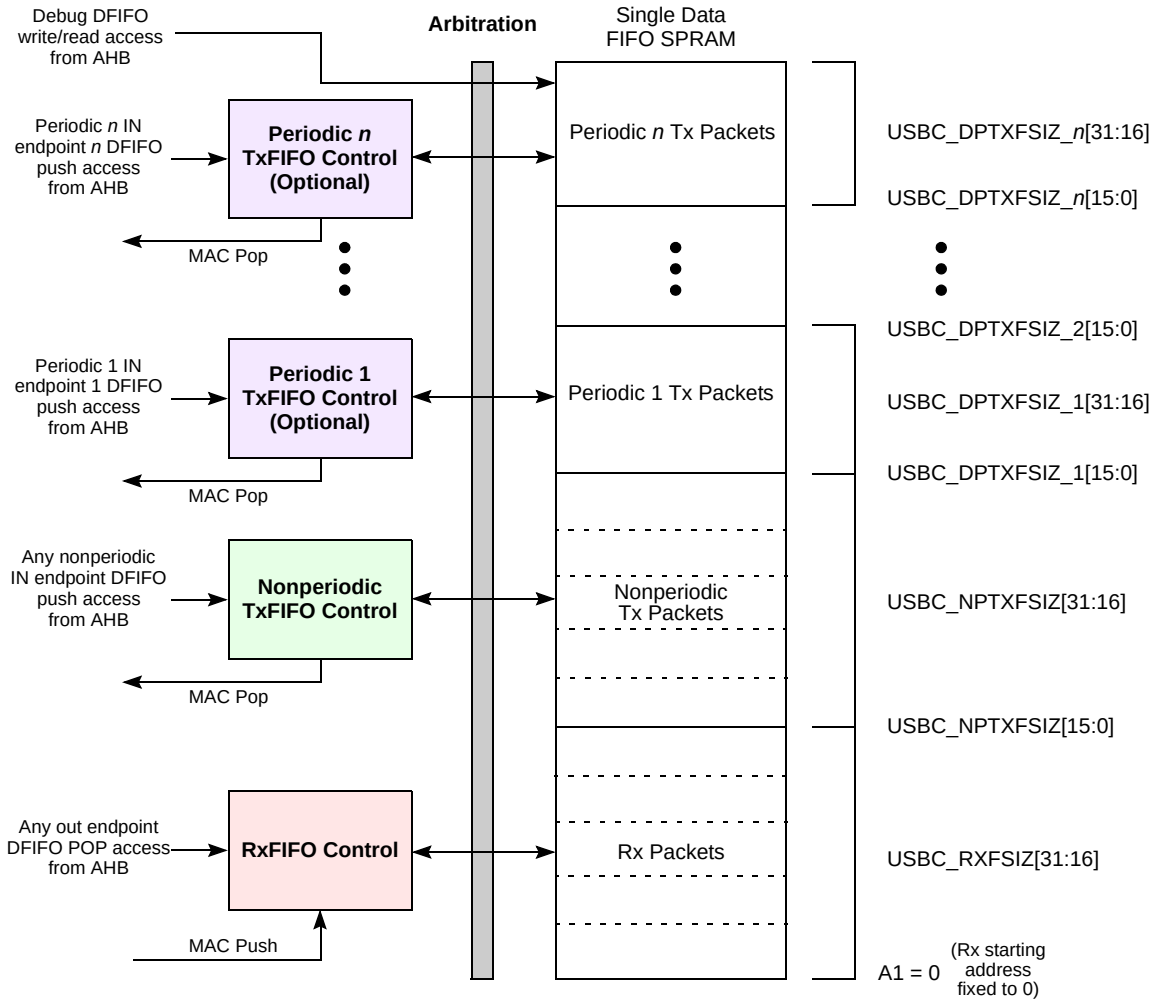


Figure 21–4 FIFO Mapping (Device Mode)

21.1.4 USB Protocol and Transaction Handling

The media-access controller (MAC) module handles USB transactions, and the host or device protocols.

21.1.4.1 USB Transaction Handling

Host Mode

The MAC receives a token request to start a USB transaction. After receiving a token request, the MAC builds and sends the requested token packet.

For OUT or SETUP transactions, the MAC reads data from the TxFIFO, builds and sends a data packet, waits for a handshake packet (if any) from the device, then updates the transaction status.

For an IN or PING transaction, the MAC waits for a data or handshake packet from the device. If it receives a handshake packet, the MAC updates the status. If it receives a data packet with the correct PID, the MAC writes the data into the RxFIFO, checks the data's integrity, sends a handshake packet, if required, to the device, then updates the status.

Device Mode

The MAC decodes and checks the integrity of token packets as it receives them from the host. If the received token is a valid OUT or SETUP token, the MAC waits and checks the PID of the following data packet, then writes the data to the RxFIFO if it is available. After the packet is completed, the MAC checks the data integrity, sends the appropriate handshake if required to the host, and writes the transaction status to the receiving status queue.

If the OUT token is received and the RxFIFO is not available, the MAC sends the host a NAK handshake. If the received token is a valid PING token, the MAC sends the appropriate handshake, based on the FIFO status and CSR control information.

If an IN token is received and the data is available in the FIFO, the MAC reads the data, builds the data packet and sends it, waits for a handshake packet, if any, from the host, then updates the transaction status. If an IN token is received and the data is not available in the FIFO, the MAC sends the host a NAK handshake.

21.1.4.2 Protocol Handling

In host mode, the MAC detects the device connect and disconnect, handles the USB reset and speed enumeration process, initiates USB suspend and resume, detects remote wakeup, generates SOF packets, and handles high-speed test modes.

In device mode, the MAC handles the USB reset sequence and speed enumeration process to determine the USB operating speed. The MAC detects USB suspend and resume signaling from the host, initiates remote wakeup, handles soft connect and disconnect, decodes and tracks SOF packets, and handles high-speed test modes.

21.1.5 Endian Swapping

When data is moved from/to the USB from/to the CN50XX, it can be swapped according to the application's needs. Note that the data swapping mentioned here does not affect read/write operations to the registers in the USB, but does affect access to the memory of the USB.

For all cases, assume the internal byte order of data is: A-B-C-D-E-F-G-H, where A is located at address 0 and H is located at address 7.

- When in PASS_THRU MODE, the order of the data will not be modified before sending/receiving to/from the USB.
- When in 64b_BYTE_SWAP MODE, the order of the data is modified to: H-G-F-E-D-C-B-A before sending/receiving to/from the USB.

21.2 Initialization

This section explains the initialization of the USB core after power-on. The initialization is broken down into three steps: power-on-reset, core initialization, and host initialization.

21.2.1 Power On Reset and PHY Initialization

This section describes the steps for USB power on reset and PHY initialization. Clock and reset requirements are shown in Figure 21–5.

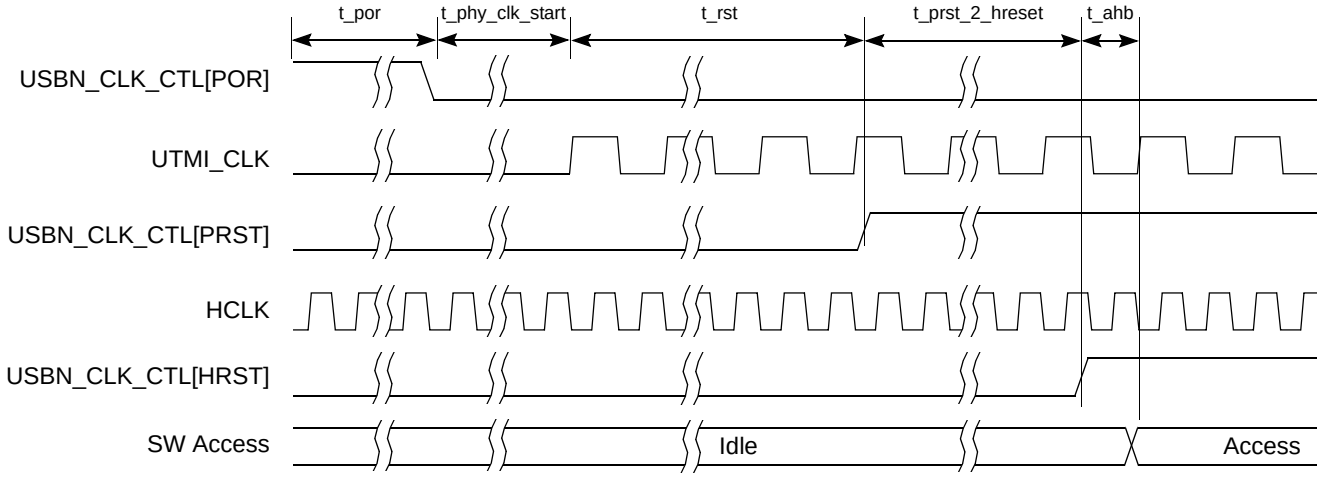


Figure 21–5 Clock and Reset Requirements

The following definitions are used in Figure 21–5:

- **t_por**: PHY power-on reset time
- **t_phy_clk_start**: PHY reset removal to clock start (typically in microseconds)
- **t_rst**: DWC_otg PHY clock domain reset and AHB HCLK domain reset over lap time (a minimum of 12 cycles of the slowest clock is recommended.)
- **t_prst_2_hreset**: prst_n removal to hreset_n remove (a minimum of six cycles of the slowest clock is recommended.)
- **t_ahb**: hreset_n removal to AHB access start (1 clock)

Overlap between PHY and AHB domain reset time (t_rst) is required so that interface signals between the two domains are correctly reset.

Setting the t_prst_2_hreset delay or setting t_ahb to 6 clock cycles is recommended to ensure that the PHY domain is ready immediately after reset.

The reset procedure is as follows:

1. Wait for DCOK to assert – all voltages have reached a stable state. There should be no action required for this, as all voltages should be stable before software can bring up the USB via the following steps.
2. Configure the reference clock, PHY, and HCLK:
   a. Write USBN_CLK_CTL[POR] = 1 and USBN_CLK_CTL[HRST,PRST,HCLK_RST] = 0
   b. Select the USB reference clock/crystal parameters by writing appropriate values to USBN_CLK_CTL[P_C_SEL, P_RTYPE, P_COM_ON]. Refer to Sections 21.7.4 and 21.7.5.
   c. Select the HCLK via writing USBN_CLK_CTL[DIVIDE, DIVIDE2] and setting USBN_CLK_CTL[ENABLE] = 1.
   d. Write USBN_CLK_CTL[HCLK_RST] = 1.

e. Wait 64 core-clock cycles for HCLK to stabilize.

3. Program the power-on reset field in the USBN clock-control register: USBN_CLK_CTL[POR] = 0
4. Wait 1 ms for PHY clock to start.
5. Program the Reset input from automatic test equipment field in the USBP control and status register: USBN_USBP_CTL_STATUS[ATE_RESET] = 1
6. Wait 10 cycles.
7. Clear ATE_RESET field in the USBN clock-control register: USBN_USBP_CTL_STATUS[ATE_RESET] = 0
8. Program the PHY reset field in the USBN clock-control register: USBN_CLK_CTL[PRST] = 1
9. Program the USBP control and status register to select host or device mode.

   USBN_USBP_CTL_STATUS[HST_MODE] = 0 for host, = 1 for device
10. Wait 1 µs.
11. Program the hreset_n field in the USBN clock-control register: USBN_CLK_CTL[HRST] = 1
12. Proceed to USB core initialization.

21.2.2 USB Core Initialization

Perform the following steps to initialize the USB core:

1. Read USBC_GHWCFG1, USBC_GHWCFG2, USBC_GHWCFG3, USBC_GHWCFG4 to determine USB core configuration parameters.
2. Program the following fields in the global AHB configuration register (USBC_GAHBCFG)
   - Slave mode, USBC_GAHBCFG[DMAEn]: 0 = slave mode
   - Burst length, USBC_GAHBCFG[HBSTLEN] = 0
   - Nonperiodic TxFIFO empty level, USBC_GAHBCFG[NPTXFEMPLVL]
   - Periodic TxFIFO empty level, USBC_GAHBCFG[PTXFEMPLVL]
   - Global interrupt mask, USBC_GAHBCFG[GLBLINTRMSK] = 1
3. Program the following fields in USBC_GUSBCFG register.
   - HS/FS timeout calibration, USBC_GUSBCFG[TOUTCAL] = 0
   - ULPI DDR select, USBC_GUSBCFG[DDRSEL] = 0
   - USB turnaround time, USBC_GUSBCFG[USBTRDTIM] = 0x5
   - PHY low-power clock select, USBC_GUSBCFG[PHYLPWRCLKSEL] = 0
4. The software must unmask the following bits in the USBC_GINTMSK register.
   - OTG interrupt mask, USBC_GINTMSK[OTGINTMSK] = 1
   - Mode mismatch interrupt mask, USBC_GINTMSK[MODEMISMSK] = 1
5. The software can read the USBC_GINTSTS[CURMOD] bit to determine whether the controller is operating in host or device mode. The software the follows either the host-initialization or device-initialization sequence.

21.2.3 Host Initialization

Host initialization consists of:

- power-on-reset (see Section 21.2.1)
- USB core initialization (see Section 21.2.2)
- host port initialization (this section)
- host channel initialization (see Section 21.5.1).

To initialize the USB core as host, the application must perform the following steps.

1. Program the host-port interrupt-mask field to unmask, USBC_GINTMSK[PRTINT] = 1
2. Program the USBC_HCFG register to select full-speed host or high-speed host.
3. Program the port power bit to drive VBUS on the USB, USBC_HPRT[PRTPWR] = 1
4. Wait for the USBC_HPRT0[PRTCONNDET] interrupt, indicating that a device is connect to the port.
5. Program the port reset bit to start the reset process., USBC_HPRT[PRTRST] = 1
6. Wait at least 10 ms for the reset process to complete.
7. Program the port reset bit to 0, USBC_HPRT[PRTRST] = 0
8. Wait for the USBC_HPRT[PRTENCHNG] interrupt.
9. Read the port speed field to get the enumerated speed, USBC_HPRT[PRTSPD].
10. Wait for an SOF transmission.
    - Read USBC_HFNUM while USBC_HFNUM[FRNUM] == 0x3FFF.
11. Program the host frame interval field, USBC_HFIR[FRINT]: 0x3750 for HS, 0x30000 for FS/LS.
12. Program the USBC_HFIR register with a value corresponding to the selected PHY clock. At this point, the host is up and running and the port register will begin to report device disconnects, etc. The port is active with SOFs occurring down the enabled port.
13. Program the USBC_GRXFSIZ register to select the size of the receive FIFO.
14. Program the USBC_GNPTXFSIZ register to select the size and the start address of the non- periodic transmit FIFO for nonperiodic transactions.
15. Program the USBC_HPTXFSIZ register to select the size and start address of the periodic transmit FIFO for periodic transactions.

To communicate with devices, the system software must initialize and enable at least one channel as described in Channel Initialization section (Section 21.5.1).

21.2.4 Device Initialization

The application must perform the following steps to initialize the USB core as a device on power-up. Device initialization consists of USB core initialization followed by device endpoint initialization

1. Program the following fields in the USBC_DCFG register:
   - Device speed, USBC_DCFG[DEVSPD] = 0 (high speed)
   - Non-zero-length status OUT handshake, USBC_DCFG[NZSTSOUTHSHK] = 0
   - Periodic frame interval (if periodic endpoints are supported), USBC_DCFG[PERFRINT] = 1
2. Program the USBC_GINTMSK register to unmask the following interrupts:
   - USB Reset, USBC_GINTMSK[USBRSTMSK] = 1
   - Enumeration done, USBC_GINTMSK[ENUMDONEMSK] = 1
   - SOF, USBC_GINTMSK[SOFMSK] = 1
3. Wait for the USBC_GINTSTS[USBRESET] interrupt, which indicates a reset has been detected on the USB and lasts for about 10 ms. On receiving this interrupt, the application must perform the steps listed in Section 21.6.1.1, "Initialization on USB Reset".
4. Wait for the USBC_GINTSTS[ENUMERATIONDONE] interrupt, which indicates the end of reset on the USB. On receiving this interrupt, the application must read the USBC_DSTS register to determine the enumeration speed and perform the steps listed in Section 21.6.1.2, "Initialization on Enumeration Completion".

At this point, the device is ready to accept SOF packets and perform control transfers on control endpoint 0.

21.3 Modes of Operation

The application operates the USB core in slave mode, where the application initiates transfers for data fetch and store.

21.3.1 Slave Mode

In slave mode, the application can operate the USBC either in transaction-level (packet-level) operation or in pipelined transaction-level operation.

21.3.1.1 Transaction-Level Operation

The application handles one data packet at a time per channel/endpoint in transaction-level operations. Based on the handshake response received on the USB, the application determines whether to retry the transaction or proceed with the next, until the end of the transfer. The application is interrupted on completion of every packet. The application performs transaction-level operations for a channel/endpoint for a transmission (host mode: OUT/ device mode: IN) or reception (host mode: IN/ device mode: OUT) as shown in Figure 21–6.

Host Mode

For an OUT transaction, the application enables the channel and writes the data packet into the corresponding (periodic or nonperiodic) transmit FIFO. The USB core automatically writes the channel number into the corresponding (periodic or nonperiodic) request queue, along with the last Dword write of the packet.

For an IN transaction, the application enables the channel and the USB core automatically writes the channel number into the corresponding request queue. The application must wait for the packet-received interrupt, then empty the packet from the receive FIFO.

Device Mode

For an IN transaction, the application enables the endpoint, writes the data packet into the corresponding transmit FIFO, and waits for the packet completion interrupt from the USB core.

For an OUT transaction, the application enables the endpoint, waits for the packet received interrupt from the USB core, then empties the packet from the receive FIFO.

The application has to finish writing one complete packet before switching to a different channel/endpoint FIFO. Violating this rule will result in an error.

Figure 21–6 shows the transaction-level operations in slave mode.

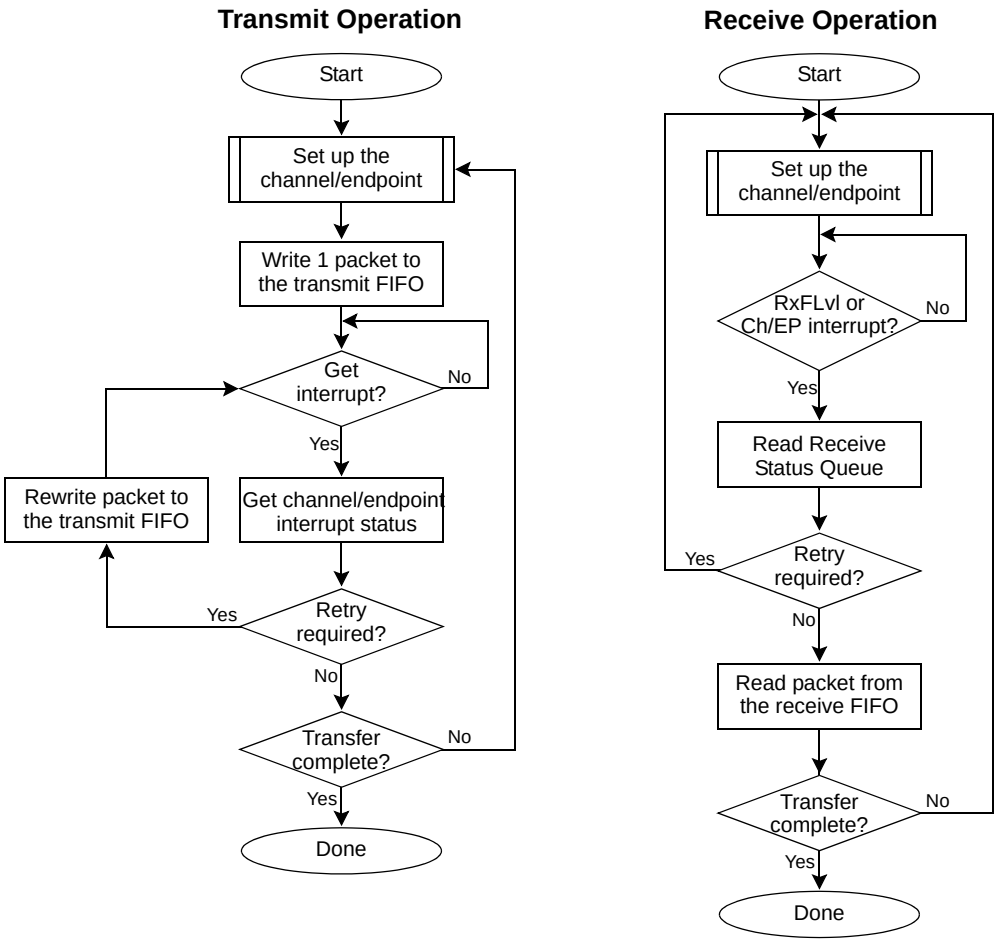


Figure 21–6 Transaction-Level Operations (Slave Mode)

21.3.1.2 Pipelined Transaction-Level Operation

The application can pipeline more than one transaction (IN or OUT) with pipelined transaction-level operations. In pipelined transaction-level operations, the application can program the USB core to perform multiple transactions. The advantage of this mode of operation compared to transaction-level operation is that the application is not interrupted on a packet basis.

Host Mode

For an OUT transaction, the application sets up a transfer and enables the channel. The application can write multiple packets back-to-back for the same channel into the transmit FIFO, based on the space availability. It can also pipeline OUT transactions for multiple channels by writing into the USBC_HCHAR*n* register, followed by a packet write to that channel. The USB core writes the channel number, along with the last Dword write for the packet, into the request queue and schedules transactions on the USB in the same order.

For an IN transaction, the application sets up a transfer and enables the channel, and the USBC writes the channel number into the request queue. The application can schedule IN transactions on multiple channels, provided space is available in the request queue. The USB core initiates an IN token on the USB only when there is enough space to receive at least of one maximum-packet-size packet of the channel in the top of the request queue.

Device Mode

For an IN transaction, the application sets up a transfer and enables the endpoint. The application can write multiple packets back-to-back for the same endpoint into the transmit FIFO, based on available space. It can also pipeline IN transactions for multiple channels by writing into the USBC_DIEPCTL*n* register followed by a packet write to that endpoint. The USB core writes the endpoint number, along with the last Dword write for the packet into the request queue. The USB core transmits the data in the transmit FIFO when an IN token is received on the USB.

For an OUT transaction, the application sets up a transfer and enables the endpoint. The USB core receives the OUT data into the receive FIFO, if it has available space. As the packets are received into the FIFO, the application must empty data from it.

From this point on in this chapter, the terms pipelined-transaction mode and transfer mode are used interchangeably.

21.3.2 Speed Mode

When the USB core is in host mode, all three speeds, high-speed (HS), full-speed (FS) and low-speed (LS), are supported. When the USB core is in device mode, high-speed (HS) and full-speed (FS) are supported. Low-speed (LS) is not supported in device mode given that a high-speed capable device cannot support low-speed device operation per the USB 2.0 specification.

Integrated 45Ω resistors set the HS input and output impedance: 1.5kΩ pull-up resistor on a D+ line for FS operation in device mode, and 15kΩ pull-down resistors on D+ and D- lines for HS/FS and LS operation in host mode. The USB core does not integrate a 1.5kΩ pull-up resistor on a D- line in LS device mode operation since LS device mode is not supported for a HS-capable device.

Once the enumerated speed is detected during host/device operation, the USB core automatically enables or disables the 45Ω impedance for HS operation and 1.5kΩ pull-up resistor on a D+ line for FS device operation, depending on the speed condition and host/device operation. The 15kΩ pull-down resistors are controlled through USBN_USBP_CTL_STATUS[DP_PULLD:DM_PULLD]. They are enabled by default in host mode, and in device mode should be disabled through USBN_USBP_CTL_STATUS[DP_PULLD:DM_PULLD].

21.4 Interrupt Handler

In order to initialize and process traffic from the USB interface, the USB-to-cnMIPS interrupt must be enabled by programming the central interrupt unit interrupt-enable register (CIU_INT*n*_EN0, where n = PPid × 2) as follows:

1. Program CIU_INT*n*_EN0[USB] = 1.

Once a cnMIPS core establishes that a USB interrupt has occurred, the interrupt handler takes the following actions.

1. Read USBC_GINTSTS and USBC_GINTMSK to determine which interrupt occurred.
2. If host port interrupt, read USBC_HPRT to further classify interrupt.
3. If host channel interrupt, read USBC_HAINT and USBC_HAINTMSK to further classify interrupt.
4. Take appropriate interrupt action and clear interrupt by writing appropriate interrupt status bit.

Figure 21–7 shows the USB interrupt handler.

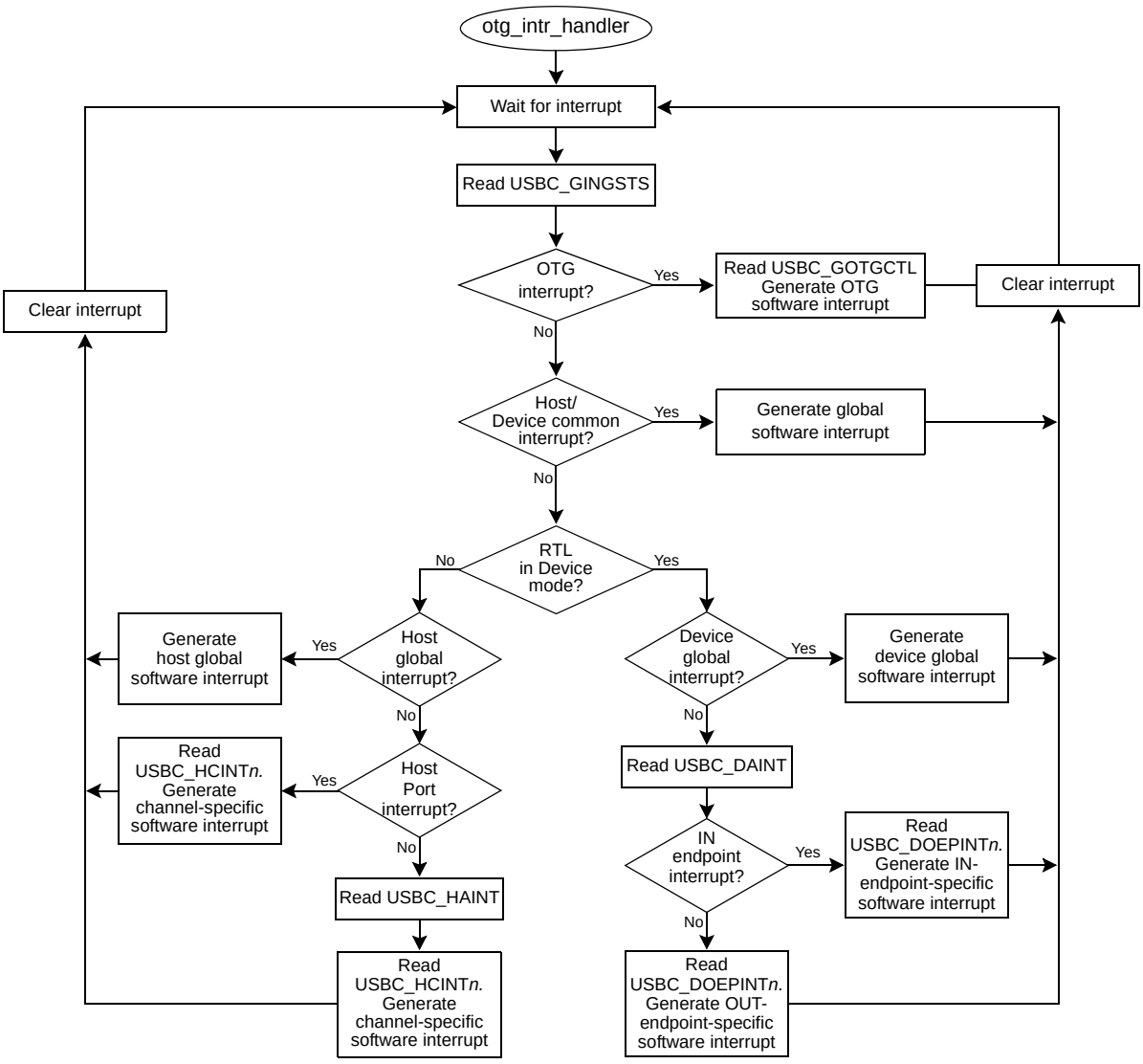


Figure 21–7 USB Interrupt Handler

21.5 Host-Mode Programming Model

The following sections describe the steps required to program the USB unit in host mode.

21.5.1 Channel Initialization

The application must initialize one or more channels before it can communicate with connected devices. To initialize and enable a channel *n*, the application must perform the following steps.

1. Program the USBC_GINTMSK register to unmask the following:
   - Channel interrupt field, [HCHINTMSK] = 1
   - Nonperiodic transmit FIFO empty field for OUT transactions (applicable for slave mode that operates in pipelined transaction-level with the Packet Count field programmed with more than one), [NPTXFEMPMSK].
   - Nonperiodic transmit FIFO half-empty field for OUT transactions (applicable for slave mode that operates in pipelined transaction-level with the packet-count field programmed with more than one), [NPTXFEMPMSK].
   - Receive FIFO non-empty mask field (if in slave mode), [RXFLVLMSK] = 1.
2. Program the USBC_HAINTMSK register to unmask the selected channels interrupts.
   - Channel Interrupt Mask, [HAINTMSK] = (1<<n)
3. Program the USBC_HCINTMSK*n* register to unmask the transaction-related interrupts of interest given in the host-channel interrupt register. Typical set of interrupts of interest are as follows.
   - transfer-completed mask, [XFERCOMPLMSK] = 1
   - channel-halted mask, [CHHLTDMSK] = 1
   - AHB error mask, [AHBERRMSK] = 1
   - STALL response-received interrupt mask, [STALLMSK] = 1
   - NAK response-received interrupt mask, [NAKMSK] = 1
   - NYET response-received interrupt mask, [NYETMSK] = 1
   - transaction-error mask, [XACTERRMSK] = 1
   - babble-error mask, [BBLERRMSK] = 1
   - frame-overrun mask, [FRMOVRUNMSK] = 1
   - data-toggle-error mask, [DATATGLERRMSK] = 1
4. Program the selected channel's USBC_HCTSIZ*n* register with the total transfer size (in bytes) and the expected number of packets, including short packets. The application must program the PID field with the initial data PID (to be used on the first OUT transaction or to be expected from the first IN transaction).
5. Program the selected channel's USBC_HCSPLT*n* registers with the hub and port addresses (split transactions only).
6. Program the USBC_HCCHAR*n* register of the selected channel with the device's endpoint characteristics, such as type, speed, direction, etc. (The channel can be enabled by setting the channel-enable bit to 1 only when the application is ready to transmit or receive any packet).

21.5.2 Halting a Channel

The application can disable any channel by setting USBC_HCCHAR*n*[CHDIS, CHENA] both to 1. This enables the USBC to flush the posted requests (if any) and generates a channel-halted interrupt. The application must wait for the USBC_HCINT*n*[CHHLTD] interrupt before reallocating the channel for other transactions. The USBC does not interrupt the transaction that has been already started on USB.

Before disabling a channel, the application must ensure that there is at least one free space available in the nonperiodic request queue (when disabling a nonperiodic channel) or the periodic request queue (when disabling a periodic channel). The application can simply flush the posted requests when the request queue is full (before disabling the channel), by setting USBC_HCCHAR*n*[CHDIS] to 1, and USBC_HCCHAR*n*[CHENA] to 0.

The application is expected to disable a channel on any of the following conditions:

1. When a USBC_HCINT*n*[XFERCOMPL] interrupt is received during a nonperiodic IN transfer or high-bandwidth interrupt IN transfer.
2. When a USBC_HCINT*n*[STALL, XACTERR, BBLERR, OR DATATGLERR] interrupt is received for an IN or OUT channel.

   For high-bandwidth interrupt INs in slave mode, once the application has received a [DATATGLERR] interrupt it must disable the channel and wait for a channel-halted interrupt. The application must be able to receive other interrupts (DATATGLERR, NAK, DATA, XACTERR, BABBLEERR) for the same channel before receiving the halt.
3. When a disconnect-device interrupt (USBC_GINTSTS[DISCONNINT]) is received. (The application is expected to disable all enabled channels).
4. When the application aborts a transfer before normal completion.

21.5.3 Ping Protocol

When the USBC operates in high speed, the application must initiate the ping protocol when communicating with high-speed bulk or control (data and status stage) OUT endpoints. The application must initiate the ping protocol when it receives a NAK/NYET/XACTERR interrupt. When the USBC receives one of the above responses, it does not continue any transaction for a specific endpoint, drops all posted or fetched OUT requests (from the request queue), and flushes the corresponding data (from the transmit FIFO).

The application can send a ping token either by setting USBC_HCTSIZ*n*[DOPNG] to 1 before enabling the channel or by just writing the USBC_HCTSIZ*n* register with DoPng bit set if the channel is already enabled. This enables the DWC_otg host to write a ping request entry to the request queue. The application must wait for the response to the ping token (a NAK, ACK, or XACTERR interrupt) before continuing the transaction or sending another ping token. The application can continue the data transaction only after receiving an ACK from the OUT endpoint for the requested ping. The channel-specific interrupt-service routine (ISR) for the ping protocol in Slave mode is shown in Example 21–1.

Example 21–1 ISR for Ping Protocol in Slave Mode

```
Unmask (ACK/NAK/XACTERR/CHHLTD/STALL)
if (ACK)
{
Reset Error Count
Re-initialize Channel (Do data transactions)
{
else if (NAK)
{
Reset Error Count
Send Ping
}
else if (STALL)
{
Disable Channel
}
else if (XACTERR)
{
Increment Error Count
    if (Error_count < 3)
    {
    Send Ping
    }
    else
    {
    Disable Channel
    }
}else if (CHHLTD)
{
De-allocate Channel
}
```

21.5.4 Sending a Zero-Length Packet

To send a zero-length data packet, the application must initialize an OUT channel as follows.

1. Program USBC_HCTSIZ*n* of the selected channel with a correct PID: [XFERSIZE] = 0, and [PKTCNT] = 1.
2. Program USBC_HCCHAR*n*[CHENA] = 1 and the device's endpoint characteristics, such as type, speed, and direction.

The application must treat a zero-length data packet as a separate transfer, and cannot combine it with a non-zero-length transfer.

21.5.5 Selecting the Queue Depth

Choose the periodic and nonperiodic request-queue depths carefully to match the number of periodic/nonperiodic endpoints accessed.

- The **nonperiodic request-queue depth** affects the performance of nonperiodic transfers. The deeper the queue (along with sufficient FIFO size), the more often the USB core is able to pipeline nonperiodic transfers. If the queue size is small, the USB core is able to put in new requests only when the queue space is freed up.
- The **periodic request-queue depth** is critical to performing periodic transfers as scheduled. Base the depth you select for the periodic queue on the number of transfers scheduled in a microframe.

  In slave mode, however, the application must also take into account the disable entry that must be put into the queue.

  - If there are two non-high-bandwidth periodic endpoints, the periodic-request queue depth must be at least four.
  - If at least one high-bandwidth endpoint is supported, the queue depth must be eight.
  - If the periodic request-queue depth is smaller than the periodic transfers scheduled in a microframe, a frame overrun condition results.

21.5.6 Handling Babble Conditions

The USBC handles two cases of babble: packet babble and port babble.

- Packet babble occurs if the device sends more data than the maximum packet size for the channel.
- Port babble occurs if the USB core continues to receive data from the device at EOF2 (the end of frame 2, which is very close to SOF).

When the USBC detects a packet babble, it stops writing data into the Rx buffer and waits for the end of packet (EOP). When it detects an EOP, it flushes already-written data in the Rx buffer and generates a Babble interrupt to the application.

When the USBC detects a port babble, it flushes the RxFIFO and disables the port. The USB core then generates a port-disabled interrupt (USBC_GINTSTS[PrtInt], USBC_HPRT[PrtEnChng]). On receiving this interrupt, the application must determine that this is not due to an overcurrent condition (another cause of the port-disabled interrupt) by checking USBC_HPRT[PrtOvrCurrAct], then perform a soft reset. The USB core does not send any more tokens after it has detected a port babble condition.

21.5.7 Host Mode Slave Transactions

The application must initialize a channel as described in Section 21.5.1 before communicating to the connected device. This section explains the sequence of operation to be performed for different types of USB transactions.

Writing the Transmit FIFO in Slave Mode

Figure 21–8 shows the flow diagram for writing to the transmit FIFO in slave mode. The USBC automatically writes an entry (OUT request) to the periodic/nonperiodic request queue, along with the last Dword write of a packet. The application must ensure that at least one free space is available in the periodic/nonperiodic request queue before starting to write to the transmit FIFO. The application must always write to the transmit FIFO in Dwords. If the packet size is non-Dword aligned, the application must use padding. The USBC host determines the actual packet size based on the programmed maximum packet size and transfer size.

Reading the Receive FIFO in Slave Mode

Figure 21–8 shows the flow diagram for reading the receive FIFO in slave mode. The application must ignore all packet statuses other than IN data packet.

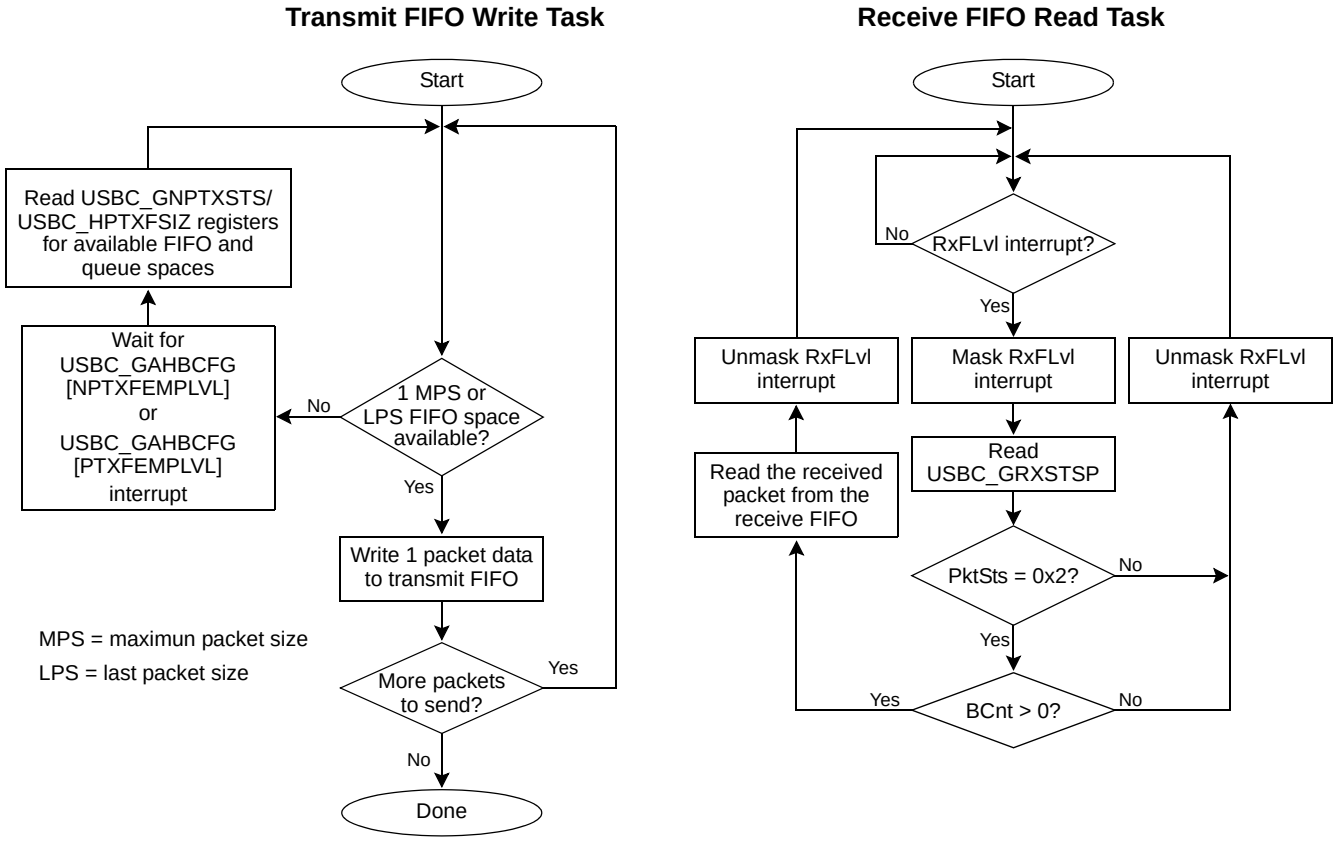


Figure 21–8 FIFO Task Flow Diagrams (Slave Mode)

21.5.7.1 Host Bulk and Control OUT/SETUP in Slave Mode

The normal sequence of operations for a bulk or control OUT/SETUP transactions to channel *n* are as follows:

1. Initialize and enable channel *n* as described in Section 21.5.1.

   Host channel *n* transfer size register – USBC_HCTSIZ*n*

   - [DOPNG] = 1 if USBC_HPRT[PRTSPD] = 0x0 (high speed)
   - [PID] = 0x0 (data0)
   - USBC_HCTSIZ0[PKTCNT] = (XFERSIZE + (MPS–1)) / MPS
   - [XFERSIZE] = total number of bytes to transfer

   Host channel *n* characteristics register – USBC_HCCHAR*n*

   - [DEVADDR]
   - [EPNUM]
   - [EPTYPE] = 0x2 (bulk)
   - [EPDIR] = 0 (out)
   - [MPS]
   - [CHENA] = 1

2. Write the first packet for channel *n*.
3. Along with the last Dword write, the USB core writes an entry to the nonperiodic request queue.
4. As soon as the nonperiodic queue becomes non-empty, the USB core attempts to send an OUT token in the current frame/microframe.
5. Wait for the PING protocol response (high-speed mode only), which should raise an ACK, NACK, or NYET interrupt.
6. Continue writing packet data until the entire XFERSIZE has been written to the nonperiodic FIFO.
7. The USB core generates the XFERCOMPL interrupt as soon as the last transaction is completed successfully.
8. In response to the XFERCOMPL interrupt, deallocate the channel for other transfers.

Handling Non-ACK Responses

The channel-specific interrupt service routine for bulk and control OUT/SETUP transactions in slave mode is shown in Example 21–2.

Example 21–2 ISR for Bulk/Control Out/Setup Transactions in Slave Mode

```
Unmask (NAK/XACTERR/NYET/STALL/XFERCOMPL)
if (XFERCOMPL)
{
Reset Error Count
Mask ACK
Deallocate Channel
}
else if (STALL)
{
Transfer Done = 1
Unmask CHHLTD
Disable Channel
}
else if (NAK or XACTERR or NYET)
{
Rewind Buffer Pointers
Unmask CHHLTD
Disable Channel
   if (XACTERR)
   {
   Increment Error Count
   Unmask ACK
   }
   else
   {
   Reset Error Count
   }
}
else if (CHHLTD)
{
Mask CHHLTD
   if (Transfer Done or (Error_count == 3))
   {
   Deallocate Channel
   }
   else
   {
   Reinitialize Channel (Do ping protocol for HS)
   }
}
else if (ACK)
{
Reset Error Count
Mask ACK
}
```

21.5.7.2 Host Bulk and Control IN Transactions in Slave Mode

The normal sequence of operations for a bulk or control IN transactions to channel n are as follows:

1. Initialize and enable channel *n* as described in Section 21.5.1.

   Host channel *n* transfer size register – USBC_HCTSIZ*n*

   - [DOPNG] = 0
   - [PID] = 0x1 (data1)
   - USBC_HCTSIZ0[PKTCNT] = (XFERSIZE + (MPS–1)) / MPS
   - [XFERSIZE] = total number of bytes to transfer

2. Set USBC_HCCHAR*n*[CHENA] to 1, to write an IN request to the nonperiodic request queue.

   Host channel *n* characteristics register – USBC_HCCHAR*n*

   - [DEVADDR]
   - [EPNUM]
   - [EPTYPE]
   - [EPDIR] = 1 (in)
   - [MPS]
   - [CHENA] = 1

3. The USB core attempts to send an IN token.

4. The USB core generates an RXFLVL interrupt as soon as the received packet is written to the receive FIFO. Each received packet generates an RXFLVL interrupt.

5. In response to the RXFLVL interrupt, mask the RXFLVL interrupt and read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. Following this, unmask the RXFLVL interrupt.

   - Set USBC_GINTMSK[RXFLVLMSK] = 0
   - If (USBC_GRXSTSPH[PKTSTS] = 0x2) && (USBC_GRXSTSPH[BCNT] ≠ 0x0)
     for i=0; i<USBC_GRXSTSPH.BCnt; i+=4
     Read USBC_NPTXDFIFO0, this is the IN packet data word i
   - Set USBC_GINTMSK.RxFlvlMsk = 1

6. The USB core generates an additional RXFLVL interrupt for the transfer completion status entry in the receive FIFO.

7. The application must read and ignore the receive packet status if the receive packet status is not an IN data packet (USBC_GRXSTSR[PKTSTS] ≠ 0x2).

8. The USB core generates the XFERCOMPL interrupt as soon as the receive packet status is read.

9. In response to the XFERCOMPL interrupt, disable the channel (as explained in Section 21.5.2) and stop writing the USBC_HCCHAR*n* register for further requests. The USB core writes a channel disable request to the nonperiodic request queue as soon as USBC_HCCHAR*n* is written.

   Make sure there is space in the nonperiodic request queue by waiting until USBC_GNPTXSTS[NPTXQSPCAVAIL] ≠ 0x0.

   Halt the channel
   - USBC_HCCHAR*n*[CHENA] = 1
   - USBC_HCCHAR*n*[CHDIS] = 1

10. The USB core generates the RXFLVL interrupt as soon as the halt status is written to the receive FIFO.

11. Read and ignore the receive packet status.

12. The USB core generates a CHHLTD interrupt as soon the halt status is popped from the receive FIFO.

13. In response to the CHHLTD interrupt, deallocate the channel for other transfers.

Handling Non-ACK Responses

The channel-specific interrupt service routine for bulk and control IN transactions in Slave mode is shown in Example 21–2.

Example 21–3 ISR for Bulk/Control IN Transactions in Slave Mode

```
Unmask (XACTERR/XFERCOMPL/BBLERR/STALL/DATATGLERR)
if (XFERCOMPL)
{
Reset Error Count
Unmask CHHLTD
Disable Channel
Reset Error Count
Mask ACK
}
else if (XACTERR or BBLERR or STALL)
{
Unmask CHHLTD
Disable Channel
   if (XACTERR)
   {Increment Error Count
   Unmask ACK
   }
}
else if (CHHLTD)
{
Mask CHHLTD
   if (Transfer Done or (Error_count == 3))
   {
   Deallocate Channel
   }
   else
   {
   Reinitialize Channel
   }
}
else if (ACK)
{
Reset Error Count
Mask ACK
}
else if (DATATGLERR)
{
Reset Error Count
}
```

21.6 Device Programming Model

21.6.1 Endpoint Initialization

21.6.1.1 Initialization on USB Reset

1. Set USBC_DOEPCTL*n*[SNAK] = 1 (for all OUT endpoints, *n* = 0-4).
2. Unmask the following interrupt bits:
   - USBC_DAINTMSK[INEPMSK] = 1 (control 0 IN endpoint)
   - USBC_DAINTMSK[OUTEPMSK] = 1 (control 0 OUT endpoint)
   - USBC_DOEPMSK[SETUPMSK] = 1
   - USBC_DOEPMSK[XFERCOMPLMSK] = 1
   - USBC_DIEPMSK[XFERCOMPLMSK] = 1
   - USBC_DIEPMSK[TIMEOUTMSK] = 1
3. To transmit or receive data, the device must initialize more registers as specified in Section 21.6.1.7.
4. Set up the data FIFO RAM for each of the FIFOs:

- Program USBC_GRXFSIZ to be able to receive control OUT data and SETUP data. This must equal at least one maximum packet size of control endpoint 0 + 2 Dwords (for the status of the control OUT data packet) + 10 Dwords (for SETUP packets).
- Program USBC_GNPTXFSIZ to be able to transmit control IN data. This must equal at least one maximum packet size of control endpoint 0.

5. Program the following fields in the endpoint-specific registers for control OUT endpoint 0 to receive a SETUP packet
   - USBC_DOEPTSIZ0[SUPCNT] = 0x3 (to receive up to three back-to-back SETUP packets)
   - In DMA mode, USBC_DOEPDMA0 register with a memory address to store any SETUP packets received

At this point, all initialization required to receive SETUP packets is done.

21.6.1.2 Initialization on Enumeration Completion

1. On receiving the enumeration-done interrupt (USBC_GINTSTS[ENUMDONE] = 1), read USBC_DSTS to determine the enumeration speed.
2. Program USBC_DIEPCTL0[MPS] to set the maximum packet size. This step configures control endpoint 0. The maximum packet size for a control endpoint depends on the enumeration speed.
3. In DMA mode, set USBC_DOEPCTL0[EPENA] = 1 to enable control OUT endpoint 0, in order to receive a SETUP packet.

At this point, the device is ready to receive SOF packets and is configured to perform control transfers on control endpoint 0.

21.6.1.3 Initialization on SetAddress Command

This section describes what the application must do when it receives a SetAddress command in a SETUP packet.

1. Program USBC_DCFG with the device address received in the SetAddress command.
2. Program the USB core to send out a status IN packet.

21.6.1.4 Initialization on SetConfiguration/SetInterface Command

This section describes what the application must do when it receives a SetConfiguration or SetInterface command in a SETUP packet.

1. When a SetConfiguration command is received, the application must program the endpoint registers to configure them with the characteristics of the valid endpoints in the new configuration.
2. When a SetInterface command is received, the application must program the endpoint registers of the endpoints affected by this command.
3. Some endpoints that were active in the prior configuration or alternate setting may not be valid in the new configuration or alternate setting. These invalid endpoints must be deactivated.
4. For details on a particular endpoints activation or deactivation, see Sections 21.6.1.5 and 21.6.1.6.

5. Unmask the interrupt for each active endpoint and mask the interrupts for all inactive endpoints in USBC_DAINTMSK.
6. Set up the data FIFO RAM for each FIFO. See Section 21.7.1 for more detail.
7. After all required endpoints are configured, the application must program the USB core to send a status IN packet.

At this point, the device core is configured to receive and transmit any type of data packet.

21.6.1.5 Endpoint Activation

This section describes the steps required to activate a device endpoint or to configure an existing device endpoint to a new type.

1. Program the characteristics of the required endpoint into the following fields of USBC_DIEPCTL*n* (for IN or bidirectional endpoints) or USBC_DOEPCTL*n* (for OUT or bidirectional endpoints).
   - Maximum packet size, [MPS]
   - USB active endpoint, [USBACTEP] = 1
   - Endpoint start data toggle, [.DPID] (for interrupt and bulk endpoints)
   - Endpoint type, [EPTYPE]
   - TxFIFO number, [TXFNUM] = 1
2. Once the endpoint is activated, the USB core starts decoding the tokens addressed to that endpoint and sends out a valid handshake for each valid token received for the endpoint.

21.6.1.6 Endpoint Deactivation

This section describes the steps required to deactivate an existing endpoint.

1. In the endpoint to be deactivated, clear USBC_DIEPCTL*n*[USBACTEP] = 0 (for IN or bidirectional endpoints) or USBC_DOEPCTL*n*[USBACTEP] = 0 (for OUT or bidirectional endpoints).
2. Once the endpoint is deactivated, the USB core ignores tokens addressed to that endpoint, resulting in a timeout on the USB.

21.6.1.7 Device Slave Mode Initialization

The application must meet the following conditions to set up the device core to handle traffic.

- USBC_GINTMSK[NPTXFEMPMSK], and USBC_GINTMSK[RXFLVLMSK] must be unset.

21.7 Miscellaneous Topics

21.7.1 Data FIFO Allocation

The USBC has 1824, four-byte words (or 7296 bytes) that must be allocated among different FIFOs before any transactions can start. The application must follow this procedure every time it changes core FIFO RAM allocation.

The application must allocate storage per FIFO based on a number of system parameters including PHY clock frequency, available bandwidth, and performance required on the USB. Based on these criteria, the application must provide a table in each mode (see Tables 21–1 and 21–2) with sizes for each FIFO.

21.7.1.1 Host Mode Allocation

The FIFO sizes in host mode are shown in Table 21–1.

Table 21–1 Data FIFO Sizes in Host Mode

FIFO Name	Data Size
Receive-data FIFO	rx_fifo_size
Nonperiodic transmit FIFO	tx_fifo_size[0]
Periodic transmit FIFO	tx_fifo_size[1]

Using the values in Table 21–1, the following registers must be programmed:

1. Receive FIFO size register (USBC_GRXFSIZ)
   [RXFDEP] = rx_fifo_size

2. Nonperiodic transmit FIFO size register (USBC_GNPTXFSIZ)
   [NPTXFDEP] = tx_fifo_size[0]
   [NPTXFSTADDR] = rx_fifo_size

3. Host periodic transmit FIFO size register (USBC_HPTXFSIZ)
   [PTXFSIZE] = tx_fifo_size[1]
   [PTXFSTADDR] = USBC_GNPTXFSIZ[NPTXFSTADDR] + tx_fifo_size[0]

4. After RAM allocation, the transmit FIFOs/receive FIFO must be flushed for proper FIFO function, using the USB core-reset register (USBC_GRSTCTL)
   [TXFNUM] = 0x10
   [TXFFLSH] = 1
   [RXFFLSH] = 1

   The application must wait until the [TXFFLSH] and [RXFFLSH] bits are cleared before performing any operation on the USB core.

21.7.1.2 Device Mode Allocation

The FIFO sizes in device mode are shown in Table 21–2.

Table 21–2 Data FIFO Sizes in Device Mode

FIFO Name	Data Size
Receive-data FIFO	rx_fifo_size. This must include RAM for the following: • SETUP packets • OUT-endpoint control information • data OUT packets
Nonperiodic transmit FIFO	tx_fifo_size[0]
Periodic transmit FIFO1 Periodic transmit FIFO2 ... Periodic transmit FIFOi	tx_fifo_size[1] tx_fifo_size[2] ... tx_fifo_size[i]

The following are considerations for allocating storage for some of the FIFOs in device mode:

- Receive FIFO storage allocation:
  - storage for SETUP packets: (4 × n + 6) locations must be reserved in the receive FIFO to receive up to n SETUP packets on control endpoints, where n is the number of control endpoints the device controller supports. The controller does not use these locations, which are reserved for SETUP packets, to write any other data.
  - one location for global OUT NAK
  - Status information is written to the FIFO along with each received packet. Therefore, a minimum space of [(Largest_Packet_Size / 4) + 1] must be allotted to receive packets.

  If a high-bandwidth endpoint is enabled, or multiple isochronous endpoints are enabled, then at least two [(Largest_Packet_Size / 4) + 1] spaces must be allotted to receive back-to-back packets. Typically, two [(Largest_Packet_Size / 4) + 1] spaces are recommended so that when the previous packet is being transferred to I/O bus, the USB can receive the subsequent packet. If I/O-bus latency is high, you must allocate enough space to receive multiple packets. This is critical to prevent dropping any isochronous packets.

  - Along with each endpoints last packet, transfer complete status information is also pushed to the FIFO. Typically, one location for each OUT endpoint is recommended.
- Transmit FIFO RAM allocation:
  - The size for the periodic-transmit FIFO must equal the maximum amount of data that can be transmitted in a single microframe. The controller does not use any data storage allocated over this requirement, and if data storage allocated is less than this requirement, the controller may malfunction.
  - The minimum amount of storage required for the nonperiodic-transmit FIFO is the largest maximum packet size among all supported nonperiodic IN endpoints.

- More space allocated in the transmit-nonperiodic FIFO results in better performance on the USB and can hide I/O-bus latencies. Typically, two Largest_Packet_Size's worth of space is recommended, so that when the current packet is under transfer to the USB, the I/O bus can get the next packet. If the I/O-bus latency is large, then you must allocate enough space to buffer multiple packets.
- It is assumed that *i* number of periodic FIFOs are implemented in device mode (refer to Table 21–2).

With these considerations and the values in Table 21–2, the following registers must be programmed:

1. Receive FIFO size register (USBC_GRXFSIZ)
   [RXFDEP] = rx_fifo_size
2. Nonperiodic-transmit FIFO size register (USBC_GNPTXFSIZ)
   [NPTXFDEP] = tx_fifo_size[0]
   [NPTXFSTADDR] = rx_fifo_size
3. Device periodic-transmit FIFO 1 size register (USBC_DPTXFSIZ1)
   [DPTXFSTADDR] = USBC_GNPTXFSIZ[NPTXFSTADDR] + tx_fifo_size[0]
4. Device periodic-transmit FIFO 2 size register (USBC_DPTXFSIZ2)
   [DPTXFSTADDR] = USBC_DPTXFSIZ1[DPTXFSTADDR] + tx_fifo_size[1]
5. Device periodic-transmit FIFO *n* size register (USBC_DPTXFSIZ*n*)
   [DPTXFSTADDR] = USBC_DPTXFSIZ*n–1*[DPTXFSTADDR] + tx_fifo_size[*n–1*]
6. After RAM allocation, the transmit FIFOs/receive FIFO must be flushed for proper FIFO function, using the USB core-reset register (USBC_GRSTCTL)
   [TXFNUM] = 0x10
   [TXFFLSH] = 1
   [RXFFLSH] = 1

   The application must wait until the [TXFFLSH] and [RXFFLSH] bits are cleared before performing any operation on the USB core.

21.7.1.3 FIFO Programming Recommendations

In the following sections, the minimum FIFO-depth allocations are specified, followed by recommendations for minimum FIFO-depth allocations and optimum FIFO-depth allocations.

Minimum FIFO-Depth Allocation (32-bit words)

The allocations specified in the section are the bare minimums needed to run the USB core.

- **RxFIFO in host mode:**
  - When no high-bandwidth periodic channel is supported in host mode, the minimum allocation is:
    - **(largest-USB-packet-used / 4) + 2**
  - When a high-bandwidth channel is supported in host mode, the minimum allocation is:
    - **2 × ((largest-USB-packet-used / 4) + 1) + 1**

- **RxFIFO in device mode:**
  - When no high-bandwidth periodic endpoint and no more than one periodic endpoint is to be supported in device mode, the minimum allocation is:
    - **(4 × number-of-control-endpoints + 6) + ((largest-USB-packet-used/4) + 1) + M**

      where:
      M = (2 × number-of-OUT-endpoints) for transfer complete or
      M = 1 for global NAK.
  - When a high-bandwidth endpoint, or more than one periodic endpoints are to be supported in device mode, the minimum allocation is:
    - **(4 × number-of-control-endpoints + 6) + 2 × ((largest-USB-packet-used / 4) + 1) + M**

      where:
      M = (2 × number-of-OUT-endpoints) for transfer complete or
      M = 1 for global NAK.
- **Nonperiodic TxFIFO:**
  - The minimum allocation is:
    - **largest-nonperiodic-USB-packet-used / 4**
- **Periodic TxFIFO in host mode:**
  - When support for only one non-high-bandwidth periodic endpoint is required, or when there are high-bandwidth or multiple isochronous endpoints but no nonisochronous transfers planned to run concurrent to the isochronous transfers, the minimum allocation is:
    - **largest-periodic-USB-packet-used / 4**
  - When support for a high-bandwidth endpoint or multiple periodic endpoints is required and non-isochronous transfers are expected, the minimum allocation is:
    - **2 × largest-periodic-USB-packet-used / 4**
- **Periodic endpoint-specific TxFIFOs in device mode:**
  - The minimum allocation is:
    - (**largest-periodic-USB-packet-used-for-an-endpoint / 4) × maximum-number-of-periodic-data-packets-per-microframe**

Recommended Minimum FIFO-Depth Allocation (32-bit words)

The specified FIFO allocations guarantee that while a packet is being transferred on the USB, the previous (next) packet is simultaneously transferred to the I/O bus. This ensures that peak USB bandwidth is achieved and that back-to-back isochronous packets can be received or sent.

- **RxFIFO in host mode:**
  - The recommended minimum allocation is:
    - **2 × ((largest-USB-packet-used / 4) + 1) + 1**
- **RxFIFO in device mode:**
  - The recommended minimum allocation is:
    - **(4 × number-of-control-endpoints + 6) + 2 × ((largest-USB-packet-used / 4) + 1) + (2 × number-of-OUT-endpoints) + 1**
- **Nonperiodic TxFIFO:**
  - The recommended minimum allocation is:
    - **2 × largest-nonperiodic-USB-packet-used / 4**

- **Periodic TxFIFO in host mode:**
  - The recommended minimum allocation is:
    - **2 × largest-periodic-USB-packet-used / 4**
- **Periodic endpoint-specific TxFIFO in device mode:**
  - The recommended minimum allocation is:
    - **largest-periodic-USB-packet-used-for-an-endpoint / 4**

If I/O bus latency is large (i.e. a received packet is not reliably transferred to the I/O bus before the next packet is received from the USB), then allocating more FIFO to the RxFIFO, nonperiodic TxFIFO, and host periodic TxFIFO is recommended. This is critical for isochronous transfers.

Recommended Optimum FIFO-Depth Allocation (32-bit words)

For the following examples, X is an integer. If there is a fractional value, X is rounded up to the next integer. X has the following value:

$$X = \frac{\text{I/O bus latency + time to transfer largest packet on I/O bus}}{\text{time to transfer largest packet on USB}}$$

- **RxFIFO in host mode:**
  - The recommended allocation is:
    - **(X + 1) × (largest-USB-packet-used / 4) + 2**
- **RxFIFO in device mode:**
  - The recommended allocation is:
    - **(4 × number-of-control-endpoints + 6) + (X + 1) × [(largest-USB-packet-used/4) + 1] + (2 × number-of-OUT-device-mode-endpoints + 1)**
- **Nonperiodic TxFIFO:**
  - The recommended allocation is:
    - **(X + 1) × largest-nonperiodic-USB-packet-used / 4**
- **Periodic TxFIFO in host mode:**
  - The recommended allocation is:
    - **(X + 1) × largest-periodic-USB-packet-used / 4**
- **Periodic endpoint-specific TxFIFO in device mode:**
  - The recommended allocation is:
    - (**largest-periodic-USB-packet-used-for-an-endpoint / 4)**
      × maximum-number-of-periodic-data-packets-per-microframe

For example, it takes approximately 10 µs to transfer a 512-byte packet on the USB (in high-speed mode). If the largest packet is 512 bytes, there are only one control endpoint and two bulk OUT endpoints, and if the I/O-bus latency plus the time to transfer 512 bytes on the I/O bus equals 20µs, then the following allocation is recommended:

RxFIFO = $(4 \times 1 + 6) + \left((20/10 + 1) \times (512/4 + 1)\right) + (2 \times 2 + 1)$ **= 402 deep**

Nonperiodic TxFIFO = 384 deep

Periodic TxFIFO (host mode) = 384 deep

Because there is a separate FIFO for each device-periodic endpoint, each has an almost 1–microframe (125µs) latency to get the next packet. Hence, only one packet-size worth of data may be allocated to them.

21.7.2 Dynamic FIFO Allocation

With dynamic FIFO allocation, the application can change the RAM allocation for each FIFO during the operation of the USB core.

Host Mode

In host mode, before changing FIFO data RAM allocation, the application must determine the following.

- All channels are disabled
- All FIFOs are empty

Once these conditions are met, the application can reallocate FIFO data RAM as explained in Section 21.7.1.

After reallocating the FIFO data RAM, the application must flush all FIFOs in the USB core using the USBC_GRSTCTL[TXFFLSH, RXFFLSH] fields. Flushing is required to reset the pointers in the FIFOs for proper FIFO operation after reallocation.

Device Mode

In device mode, before changing FIFO data RAM allocation, the application must determine the following.

- All IN and OUT endpoints are disabled
- NAK mode is enabled in the USB core on all IN endpoints
- Global OUT NAK mode is enabled in the USB core
- All FIFOs are empty

Once these conditions are met, the application can reallocate FIFO data RAM as explained in Section 21.7.1. When NAK mode is enabled in the USB core, the core responds with a NAK handshake on all tokens received on the USB, except for SETUP packets.

After the reallocating the FIFO data RAM, the application must flush all FIFOs in the USB core using the USBC_GRSTCTL[TXFFLSH, RXFFLSH] fields. Flushing is required to reset the pointers in the FIFOs for proper FIFO operation after reallocation.

21.7.3 Power Saving Modes

In systems in which the USB is not present, software can clear USBN_CLK_CTL[ENABLE] to 0 to prevent the USB clock from being generated, providing some power-savings.

In systems in which USB is present, the USBC has no support for additional power saving modes.

21.7.4 Reference Clocks

The USB interface supports the following reference clock sources:

- **Crystal connected to the xi and xo pins:** The crystal must have a fundamental frequency of 12 MHz, with a frequency tolerance of ±400 ppm, peak jitter of ±100 ps, and an output differential voltage of no less than 500 mV with respect to the xi signal. Peak jitter is the absolute jitter over an 18µs period.
- **External clock connected to the xo pin:** The signal must have a fundamental frequency of 12/24/48 MHz, with a frequency tolerance of ±400 ppm, peak jitter of ±100 ps, duty cycle between 40/60 and 60/40 percent, and signal swing of 3.3V ± 7%. Peak jitter is the absolute jitter over an 18µs period. The xi pin should be tied to ground.

The external clock is set through USBN_CLK_CTL[P_C_SEL, P_RTYPE,]. Refer to Section 21.2.1.

21.7.5 Crystal Oscillators

Refer to *Cavium Networks OCTEON Plus USB Crystal Selection Application Note* (CN5XXX_USB-AN-0.9) for the selection of a crystal oscillator.

21.8 USB Registers

The USB controller supports two kinds of CSRs: registers that interface with the USB core (USBN) in Table 21–3 and registers that control the USB core (USBC) in Table 21–4.

21.8.1 USBN Registers

The USBN registers are shown in Table 21–3.

Table 21–3 USBN Registers

Register	Address	CSR Type[1]	Detailed Description
USBN_INT_SUM	0x0001180068000000	RSL	See page 709
USBN_INT_ENB	0x0001180068000008	RSL	See page 710
USBN_CLK_CTL	0x0001180068000010	RSL	See page 712
USBN_USBP_CTL_STATUS	0x0001180068000018	RSL	See page 713
USBN_BIST_STATUS	0x00011800680007F8	RSL	See page 715
USBN_CTL_STATUS	0x00016F0000000800	RSL	See page 715
USBN_DMA_TEST	0x00016F0000000808	RSL	See page 716
USBN_DMA0_INB_CHN0 ... USBN_DMA0_INB_CHN7	0x00016F0000000818 ... 0x00016F0000000850	RSL	See page 716
USBN_DMA0_OUTB_CHN0 ... USBN_DMA0_OUTB_CHN7	0x00016F0000000858 ... 0x00016F0000000890	RSL	See page 716

1. RSL-type registers are accessed indirectly across the I/O Bus.

Interrupt Summary Register
USBN_INT_SUM

This register contains the interrupt summary bits of the USBN. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:38>	—	RAZ	—	—	Reserved.
<37>	ND4O_DPF	R/W1C	—	0	I/O bus DMA out data FIFO push full.
<36>	ND4O_DPE	R/W1C	—	0	I/O bus DMA out data FIFO pop empty.
<35>	ND4O_RPF	R/W1C	—	0	I/O bus DMA out request FIFO push full.
<34>	ND4O_RPE	R/W1C	—	0	I/O bus DMA out request FIFO pop empty.
<33>	LTL_F_PF	R/W1C	0	0	L2C transfer-length FIFO push full.
<32>	LTL_F_PE	R/W1C	0	0	L2C transfer-length FIFO pop empty.
<31:26>	—	R/W1C	—	0x0	Reserved.
<25>	UOD_PF	R/W1C	—	0	UOD FIFO push full.
<24>	UOD_PE	R/W1C	—	0	UOD FIFO pop empty.
<23>	RQ_Q3_E	R/W1C	0	0	Request Queue 3 FIFO pushed when full.
<22>	RQ_Q3_F	R/W1C	0	0	Request Queue 3 FIFO pushed when full.
<21>	RQ_Q2_E	R/W1C	0	0	Request Queue 2 FIFO pushed when full.
<20>	RQ_Q2_F	R/W1C	0	0	Request Queue 2 FIFO pushed when full.
<19>	RG_FI_F	R/W1C	0	0	Register request FIFO pushed when full.
<18>	RG_FI_E	R/W1C	0	0	Register request FIFO pushed when full.
<17>	LT_FI_F	R/W1C	0	0	L2C request FIFO pushed when full.
<16>	LT_FI_E	R/W1C	0	0	L2C request FIFO pushed when full.
<15>	L2C_A_F	R/W1C	—	0	L2C credit count added when full.
<14>	L2C_S_E	R/W1C	—	0	L2C credit count subtracted when empty.
<13>	DCRED_F	R/W1C	0	0	Data credit FIFO pushed when full.
<12>	DCRED_E	R/W1C	0	0	Data credit FIFO pushed when full.
<11>	LT_PU_F	R/W1C	0	0	L2C transaction FIFO pushed when full.
<10>	LT_PO_E	R/W1C	0	0	L2C transaction FIFO popped when full.
<9>	NT_PU_F	R/W1C	0	0	NPI transaction FIFO pushed when full.
<8>	NT_PO_E	R/W1C	0	0	NPI transaction FIFO popped when full.
<7>	PT_PU_F	R/W1C	0	0	Core transaction FIFO pushed when full.
<6>	PT_PO_E	R/W1C	0	0	Core transaction FIFO popped when full.
<5>	LR_PU_F	R/W1C	0	0	L2C request FIFO pushed when full.
<4>	LR_PO_E	R/W1C	0	0	L2C request FIFO popped when empty.
<3>	NR_PU_F	R/W1C	0	0	NPI request FIFO pushed when full.
<2>	NR_PO_E	R/W1C	0	0	NPI request FIFO popped when empty.
<1>	PR_PU_F	R/W1C	0	0	Core request FIFO pushed when full.
<0>	PR_PO_E	R/W1C	0	0	Core request FIFO popped when empty.

Interrupt Enable Register
USBN_INT_ENB

This register contains the interrupt summary bits of the USBN. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:38>	—	RAZ	—	—	Reserved.
<37>	ND4O_DPF	R/W	0	0	When set to 1 and bit <37> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<36>	ND4O_DPE	R/W	0	0	When set to 1 and bit <36> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<35>	ND4O_RPF	R/W	0	0	When set to 1 and bit <35> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<34>	ND4O_RPE	R/W	0	0	When set to 1 and bit <34> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<33>	LTL_F_PF	R/W	0	0	When set to 1 and bit <33> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<32>	LTL_F_PE	R/W	0	0	When set to 1 and bit <32> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<31:26>	—	R/W	0x0	0x0	Reserved.
<25>	UOD_PF	R/W	0	0	When set to 1 and bit <25> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<24>	UOD_PE	R/W	0	0	When set to 1 and bit <24> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<23>	RQ_Q3_E	R/W	0	0	When set to 1 and bit <23> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<22>	RQ_Q3_F	R/W	0	0	When set to 1 and bit <22> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<21>	RQ_Q2_E	R/W	0	0	When set to 1 and bit <21> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<20>	RQ_Q2_F	R/W	0	0	When set to 1 and bit <20> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<19>	RG_FI_F	R/W	0	0	When set to 1 and bit <19> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<18>	RG_FI_E	R/W	0	0	When set to 1 and bit <18> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<17>	LT_FI_F	R/W	0	0	When set to 1 and bit <17> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<16>	LT_FI_E	R/W	0	0	When set to 1 and bit <16> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<15>	L2C_A_F	R/W	0	0	When set to 1 and bit <15> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<14>	L2C_S_E	R/W	0	0	When set to 1 and bit <14> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<13>	DCRED_F	R/W	0	0	When set to 1 and bit <13> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<12>	DCRED_E	R/W	0	0	When set to 1 and bit <12> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<11>	LT_PU_F	R/W	0	0	When set to 1 and bit <11> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<10>	LT_PO_E	R/W	0	0	When set to 1 and bit <10> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<9>	NT_PU_F	R/W	0	0	When set to 1 and bit <9> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<8>	NT_PO_E	R/W	0	0	When set to 1 and bit <8> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<7>	PT_PU_F	R/W	0	0	When set to 1 and bit <7> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<6>	PT_PO_E	R/W	0	0	When set to 1 and bit <6> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<5>	LR_PU_F	R/W	0	0	When set to 1 and bit <5> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<4>	LR_PO_E	R/W	0	0	When set to 1 and bit <4> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<3>	NR_PU_F	R/W	0	0	When set to 1 and bit <3> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<2>	NR_PO_E	R/W	0	0	When set to 1 and bit <2> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<1>	PR_PU_F	R/W	0	0	When set to 1 and bit <1> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.
<0>	PR_PO_E	R/W	0	0	When set to 1 and bit <0> of USBN_INT_SUM is asserted, the USBN asserts an interrupt.

Clock-Control Register
USBN_CLK_CTL

This register controls the frequency of the HCLK and the HRESET and PHY_RST signals. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:20>	—	RAZ	—	—	Reserved.
<19:18>	DIVIDE2	R/W	0x0	0x1	The HCLK used by the USB subsystem is derived from the ECLK (core clock). Also see [DIVIDE]. DIVIDE2<1> must currently be zero because it is not implemented, so the maximum ratio of ECLK/HCLK is currently 16. The actual divide number for HCLK is: (DIVIDE2 + 1) × (DIVIDE + 1).
<17>	HCLK_RST	R/W	1	1	HCLK reset. Active low. When set to 0, DIVIDE (used to generate the HCLK in the USB subsystem) is held in reset. This bit must be set to 0 before changing the value of DIVIDE. The reset to the DIVIDE is also asserted when core reset is asserted.
<16>	—	RAZ	—	—	Reserved.
<15:14>	P_RTYPE	R/W	0x0	0x0	PHY reference-clock type. 00 = uses 12MHz crystal as clock source at USB_XO and USB_XI. 01 = reserved. 10 = uses 12/24/48MHz 2.5V board clock source at USB_XO. USB_XI should be tied to GND. 11 = reserved.
<13>	P_COM_ON	R/W	1	1	Force USB-PHY XO bias, bandgap, and PLL to remain on in suspend mode. 0 = USB-PHY XO bias, bandgap, and PLL remain powered on during suspend. 1 = USB-PHY XO bias, bandgap, and PLL are powered down during suspend. This value must be set while POR is active.
<12:11>	P_C_SEL	R/W	0x2	0x0	PHY clock speed select. Selects the reference clock/crystal frequency. 00 = 12 MHz, 01 = 24 MHz, 10 = 48 MHz, 11 = Reserved This value must be set while POR is active. **NOTE:** If a crystal is used as a reference clock, this field must be set to 12 MHz. Values 01 and 10 are reserved when a crystal is used.
<10>	CDIV_BYP	R/W	0	0	Used to enable the bypass input to the USB_CLK_DIV.
<9:8>	SD_MODE	R/W	0x0	0x0	Scaledown mode for the USBC. Controls timing events in the USBC. For normal operation, this field must be 0x0.
<7>	S_BIST	R/W	0	1	Starts BIST on the HCLK memories, during the 0-to-1 transition.
<6>	POR	R/W	1	0	Power on reset for the PHY. Resets all the PHY's registers and state machines.
<5>	ENABLE	R/W	1	1	HCLK enable. When set to 1, allows the generation of the HCLK. When cleared to 0, the HCLK is not generated. See [DIVIDE] of this register.
<4>	PRST	R/W	0	1	PHY_CLK reset. This is the value for PHY_RST_N. When this field is '0' the logic associated with the PHY_CLK functionality in the USB subsystem is held in reset. This should not be set to 1 until the time it takes for six clock cycles (HCLK or PHY_CLK, whichever is slower) has passed. Under normal operation, once this bit is set to 1, it should not be set to 0.
<3>	HRST	R/W	0	1	HCLK reset. This is the value for HRESET_N. When this field is 0, the logic associated with the HCLK functionality in the USB subsystem is held in reset. This should not be set to 1 until 12 ms after PHY_CLK is stable. Under normal operation, once this bit is set to 1, it should not be set to 0.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<2:0>	DIVIDE	R/W	0x4	0x0	The frequency of HCLK used by the USB subsystem is the ECLK (core clock) frequency divided by the value of (DIVIDE2 + 1) × (DIVIDE + 1). Also see the DIVIDE2 field of this register. The HCLK frequency should be less than 125 Mhz. After writing a value to this field, the software should read the field for the value written. The [ENABLE] field of this register should not be set until after this field is set and then read.

USBP Control and Status Register USBN_USBP_CTL_STATUS

This register contains general control and status information for the USBN block. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63>	TXRISETUNE	R/W	0	0	HS transmitter rise/fall time adjustment.
<62:59>	TXVREFTUNE	R/W	0x7	0x7	HS DC voltage level adjustment.
<58:55>	TXFSLSTUNE	R/W	0x3	0x3	FS/LS source impedance adjustment.
<54:53>	TXHSXVTUNE	R/W	0x0	0x0	Transmitter high-speed crossover adjustment.
<52:50>	SQRXTUNE	R/W	0x3	0x3	Squelch threshold adjustment.
<49:47>	COMPDISTUNE	R/W	0x2	0x2	Disconnect threshold adjustment.
<46:44>	OTGTUNE	R/W	0x2	0x2	VBUS valid threshold adjustment.
<43>	OTGDISABLE	R/W	1	1	OTG block disable.
<42>	PORTRESET	R/W	0	0	Per-port reset.
<41>	DRVVBUS	R/W	0	0	Drive VBUS.
<40>	LSBIST	R/W	0	0	Low-speed BIST enable.
<39>	FSBIST	R/W	0	0	Full-speed BIST enable.
<38>	HSBIST	R/W	0	0	High-speed BIST enable.
<37>	BIST_DONE	RO	0	0	PHY BIST done. Asserted at the end of the PHY BIST sequence.
<36>	BIST_ERR	RO	0	0	PHY BIST error. Indicates an internal error was detected during the BIST sequence.
<35:32>	TDATA_OUT	RO	—	—	PHY test data out. Presents either internally generated signals or test register contents, based upon the value of TDATA_SEL.
<31>	—	R/W	0	0	Spare bit.
<30>	TXPREEMPHASISTUNE	R/W	0	0	HS transmitter pre-emphasis enable.
<29>	DMA_BMODE	R/W	0	0	DMA byte-count mode. When set to 1, the L2C DMA address is updated with byte counts between packets. When set to 0, the L2C DMA address is incremented to the next 4-byte aligned address after adding byte count.
<28>	USBC_END	R/W	0	0	Big-endian input to the USB core. This should be set to 0 for operation.
<27>	USBP_BIST	R/W	1	1	This is cleared 0 to run BIST on the USBP.
<26>	TCLK	R/W	0	0	PHY test clock, used to load TDATA_IN to the USBP.
<25>	DP_PULLD	R/W	1	1	DP_PULLDOWN input to the USB-PHY. This signal enables the pull-down resistance on the D+ line. 1 = pull-down resistance is connected to D+. 0 = pull-down resistance is not connected to D+. When an A/B device is acting as a host (downstream-facing port), DP_PULLDOWN and DM_PULLDOWN are enabled. This must not toggle during normal operation.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<24>	DM_PULLD	R/W	1	1	DM_PULLDOWN input to the USB-PHY. This signal enables the pull-down resistance on the D– line. 1 = pull-down resistance is connected to D–. 0 = pull-down resistance is not connected to D–. When an A/B device is acting as a host (downstream-facing port), DP_PULLDOWN and DM_PULLDOWN are enabled. This must not toggle during normal operation.
<23>	HST_MODE	R/W	0	0	Host mode: 0 = USB is acting as HOST, 1 = USB is acting as device. This field needs to be set while the USB is in reset.
<22:19>	—	R/W	0x0	0x0	Spares bits.
<18>	TX_BS_ENH	R/W	0	0	Transmit-bit stuffing on [15:8]. When set, enables bit stuffing on data[15:8] when bit-stuffing is enabled.
<17>	TX_BS_EN	R/W	0	0	Transmit-bit stuffing on [7:0]. When set, enables bit stuffing on data[7:0] when bit-stuffing is enabled.
<16>	LOOP_ENB	R/W	0	0	Loopback enable. 1 = During data transmission, the receive is enabled. 0 = During data transmission, the receive is disabled. Must be 0 for normal operation.
<15>	VTEST_ENB	R/W	0	0	Analog test-pin enable. 1 = The PHY's ANALOG_TEST pin is enabled for the input and output of applicable analog test signals. 0 = The ANALOG_TEST pin is disabled.
<14>	BIST_ENB	R/W	0	0	BIST enable.
<13>	TDATA_SEL	R/W	0	0	Test data out select. 1 = TDATA_OUT[3:0] are output. 0 = internally generated signals are output.
<12:9>	TADDR_IN	R/W	0x0	0x0	Mode address for test interface. Specifies the register address for writing to or reading from the PHY test interface register.
<8:1>	TDATA_IN	R/W	0x0	0x0	Internal testing register input data and select. This is a test bus. Data is present on [3:0], and its corresponding select (enable) is present on bits [7:4].
<0>	ATE_RESET	R/W	0	0	Reset input from automatic-test equipment. This is a test signal. When the USB core is powered up (not in suspend mode), an automatic tester can use this to disable PHY_CLOCK and FREE_CLK, then re-enable them with an aligned phase. 1 = PHY_CLK and FREE_CLK outputs are disabled. 0 = PHY_CLK and FREE_CLK outputs are available within a specific period after the deassertion.

BIST Status Register
USBN_BIST_STATUS

This register contains status information about the USB built-in selftests. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:7>	—	RAZ	—	—	Reserved.
<6>	U2NC_BIS	RO	0	0	BIST status U2N CTL FIFO Memory.
<5>	U2NF_BIS	RO	0	0	BIST status U2N FIFO Memory.
<4>	E2HC_BIS	RO	0	0	BIST status E2H CTL FIFO Memory.
<3>	N2UF_BIS	RO	0	0	BIST status N2U FIFO Memory.
<2>	USBC_BIS	RO	0	0	BIST status USBC FIFO memory.
<1>	NIF_BIS	RO	0	0	BIST status for inbound memory.
<0>	NOF_BIS	RO	0	0	BIST status for outbound memory.

Control and Status Register
USBN_CTL_STATUS

This register contains general control and status information for the USBN block. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:6>	—	RAZ	—	—	Reserved.
<5>	DMA_0PAG	R/W	0	0	DMA zero page. When set to 1, sets the DMA engine sets the zero-page bit in the L2C store operation to the IOB.
<4>	DMA_STT	R/W	0	0	DMA STT mode. When set to 1, sets the DMA engine to use STT operations.
<3>	DMA_TEST	R/W	0	0	DMA test mode. When set to 1, sets the DMA engine into test mode. For normal operation, this bit should be 0.
<2>	INV_A2	R/W	0	0	Address bit<2> invert. When set to 1, causes the address bit <2> driven on the AHB for USB-core FIFO access to be inverted. Also data written to and read from the AHB has its byte order swapped. If the original order was A-B-C-D the new byte order is D-C-B-A.
<1:0>	L2C_EMOD	R/W	0x1	0x1	Endian format for data from/to the L2C. IN: A-B-C-D-E-F-G-H OUT0: A-B-C-D-E-F-G-H OUT1: H-G-F-E-D-C-B-A OUT2: D-C-B-A-H-G-F-E OUT3: E-F-G-H-A-B-C-D

DMA Test Register
USBN_DMA_TEST

This register allows the external DMA engine (to the USB core) to make transfers to/from the L2C/USB FIFOs. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:40>	—	RAZ	—	—	Reserved.
<39>	DONE	R/W1C	0	0	This field sets to 1 when a DMA operation completes.
<38>	REQ	R/W1C	0	0	DMA request. Writing a 1 to this register causes a DMA request as specified in the other fields of this register to take place. This field always reads as 0.
<37:20>	F_ADDR	R/W	0x0	0x0	FIFO address. Indicates the address in the data FIFO to read from.
<19:9>	COUNT	R/W	0x0	0x0	DMA request count.
<8:4>	CHANNEL	R/W	0x0	0x0	DMA channel endpoint.
<3:0>	BURST	R/W	0	0	DMA burst size.

Inbound DMA Registers
USBN_DMA0_INB_CHN(0..7)

These registers contains the starting address to use when the USB writes to L2C via channel *n*. Writing these registers sets the base address. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	—	—	Reserved.
<35:0>	ADDR	R/W	0x0	—	Base address for DMA writes to L2C.

Outbound DMA Registers
USBN_DMA0_OUTB_CHN(0..7)

These registers contains the starting address to use when the USB reads from L2C via channel *n*. Writing these registers sets the base address. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<63:36>	—	RAZ	—	—	Reserved.
<35:0>	ADDR	R/W	0x0	—	Base address for DMA reads from L2C.

21.8.2 USBC Registers

The USBC registers are 32-bit NCB registers and should be accessed with 32-bit core load/store operations. The USBC register addresses (both little-endian and big-endian) are shown in Table 21–4.

Table 21–4 USBC Registers

Register	Address (NCB, Little-Endian)	Address (NCB, Big-Endian)	CSR Type[1]	Detailed Description
USBC_GOTGCTL	0x00016F0010000000	0x00016F0010000004	NCB	See page 719
USBC_GOTGINT	0x00016F0010000004	0x00016F0010000000	NCB	See page 720
USBC_GAHBCFG	0x00016F0010000008	0x00016F001000000C	NCB	See page 721
USBC_GUSBCFG	0x00016F001000000C	0x00016F0010000008	NCB	See page 722
USBC_GRSTCTL	0x00016F0010000010	0x00016F0010000014	NCB	See page 724
USBC_GINTSTS	0x00016F0010000014	0x00016F0010000010	NCB	See page 726
USBC_GINTMSK	0x00016F0010000018	0x00016F001000001C	NCB	See page 729
USBC_GRXSTSRH	0x00016F001000001C	0x00016F0010000018	NCB	See page 730
USBC_GRXSTSPH	0x00016F0010000020	0x00016F0010000024	NCB	See page 730
USBC_GRXFSIZ	0x00016F0010000024	0x00016F0010000020	NCB	See page 732
USBC_GNPTXFSIZ	0x00016F0010000028	0x00016F001000002C	NCB	See page 733
USBC_GNPTXSTS	0x00016F001000002C	0x00016F0010000028	NCB	See page 733
USBC_GSNPSID	0x00016F0010000040	0x00016F0010000044	NCB	See page 734
USBC_GHWCFG1 USBC_GHWCFG2 USBC_GHWCFG3 USBC_GHWCFG4	0x00016F0010000044 0x00016F0010000048 0x00016F001000004C 0x00016F0010000050	0x00016F0010000040 0x00016F001000004C 0x00016F0010000048 0x00016F0010000054	NCB	See page 734
USBC_HPTXFSIZ	0x00016F0010000100	0x00016F0010000104	NCB	See page 737
USBC_DPTXFSIZ1 USBC_DPTXFSIZ2 USBC_DPTXFSIZ3 USBC_DPTXFSIZ4	0x00016F0010000104 0x00016F0010000108 0x00016F001000010C 0x00016F0010000110	0x00016F0010000100 0x00016F001000010C 0x00016F0010000108 0x00016F0010000114	NCB	See page 737
USBC_HCFG	0x00016F0010000400	0x00016F0010000404	NCB	See page 737
USBC_HFIR	0x00016F0010000404	0x00016F0010000400	NCB	See page 738
USBC0_HFNUM	0x00016F0010000408	0x00016F001000040C	NCB	See page 739
USBC_HPTXSTS	0x00016F0010000410	0x00016F0010000414	NCB	See page 739
USBC_HAINT	0x00016F0010000414	0x00016F0010000410	NCB	See page 740
USBC_HAINTMSK	0x00016F0010000418	0x00016F001000041C	NCB	See page 740
USBC_HPRT	0x00016F0010000440	0x00016F0010000444	NCB	See page 741
USBC_HCCHAR0 USBC_HCCHAR1 ... USBC_HCCHAR6 USBC_HCCHAR7	0x00016F0010000500 0x00016F0010000520 ... 0x00016F00100005C0 0x00016F00100005E0	0x00016F0010000504 0x00016F0010000524 ... 0x00016F00100005C4 0x00016F00100005E4	NCB	See page 743
USBC_HCSPLT0 USBC_HCSPLT1 ... USBC_HCSPLT6 USBC_HCSPLT7	0x00016F0010000504 0x00016F0010000524 ... 0x00016F00100005C4 0x00016F00100005E4	0x00016F0010000500 0x00016F0010000520 ... 0x00016F00100005C0 0x00016F00100005E0	NCB	See page 744
USBC_HCINT0 USBC_HCINT1 ... USBC_HCINT06 USBC_HCINT7	0x00016F0010000508 0x00016F0010000528 ... 0x00016F00100005C8 0x00016F00100005E8	0x00016F001000050C 0x00016F001000052C ... 0x00016F00100005CC 0x00016F00100005EC	NCB	See page 744

Table 21–4 USBC Registers (Continued)

Register	Address (NCB, Little-Endian)	Address (NCB, Big-Endian)	CSR Type[1]	Detailed Description
USBC_HCINTMSK0 USBC_HCINTMSK1 ... USBC_HCINTMSK6 USBC_HCINTMSK7	0x00016F001000050C 0x00016F001000052C ... 0x00016F00100005CC 0x00016F00100005EC	0x00016F0010000508 0x00016F0010000528 ... 0x00016F00100005C8 0x00016F00100005E8	NCB	See page 745
USBC_HCTSIZ0 USBC_HCTSIZ1 ... USBC_HCTSIZ06 USBC_HCTSIZ7	0x00016F0010000510 0x00016F0010000530 ... 0x00016F00100005D0 0x00016F00100005F0	0x00016F0010000514 0x00016F0010000534 ... 0x00016F00100005D4 0x00016F00100005F4	NCB	See page 745
USBC_DCFG	0x00016F0010000800	0x00016F0010000804	NCB	See page 746
USBC_DCTL	0x00016F0010000804	0x00016F0010000800	NCB	See page 747
USBC_DSTS	0x00016F0010000808	0x00016F001000080C	NCB	See page 748
USBC_DIEPMSK	0x00016F0010000810	0x00016F0010000814	NCB	See page 749
USBC_DOEPMSK	0x00016F0010000814	0x00016F0010000810	NCB	See page 749
USBC_DAINT	0x00016F0010000818	0x00016F001000081C	NCB	See page 750
USBC_DAINTMSK	0x00016F001000081C	0x00016F0010000818	NCB	See page 750
USBC_DTKNQR1 USBC_DTKNQR2 USBC_DTKNQR3 USBC_DTKNQR4	0x00016F0010000820 0x00016F0010000824 0x00016F0010000830 0x00016F0010000834	0x00016F0010000824 0x00016F0010000820 0x00016F0010000834 0x00016F0010000830	NCB	See page 750
USBC_DIEPCTL0 USBC_DIEPCTL1 USBC_DIEPCTL2 USBC_DIEPCTL3 USBC_DIEPCTL4	0x00016F0010000900 0x00016F0010000920 0x00016F0010000940 0x00016F0010000960 0x00016F0010000980	0x00016F0010000904 0x00016F0010000924 0x00016F0010000944 0x00016F0010000964 0x00016F0010000984	NCB	See page 751
USBC_DIEPINT0 USBC_DIEPINT1 USBC_DIEPINT2 USBC_DIEPINT3 USBC_DIEPINT4	0x00016F0010000908 0x00016F0010000928 0x00016F0010000948 0x00016F0010000968 0x00016F0010000988	0x00016F001000090C 0x00016F001000092C 0x00016F001000094C 0x00016F001000096C 0x00016F001000098C	NCB	See page 754
USBC_DIEPTSIZ0 USBC_DIEPTSIZ1 USBC_DIEPTSIZ2 USBC_DIEPTSIZ3 USBC_DIEPTSIZ4	0x00016F0010000910 0x00016F0010000930 0x00016F0010000950 0x00016F0010000970 0x00016F0010000990	0x00016F0010000914 0x00016F0010000934 0x00016F0010000954 0x00016F0010000974 0x00016F0010000994	NCB	See page 755
USBC_DOEPCTL0 USBC_DOEPCTL1 USBC_DOEPCTL2 USBC_DOEPCTL3 USBC_DOEPCTL4	0x00016F0010000B00 0x00016F0010000B20 0x00016F0010000B40 0x00016F0010000B60 0x00016F0010000B80	0x00016F0010000B04 0x00016F0010000B24 0x00016F0010000B44 0x00016F0010000B64 0x00016F0010000B84	NCB	See page 756
USBC_DOEPINT0 USBC_DOEPINT1 USBC_DOEPINT2 USBC_DOEPINT3 USBC_DOEPINT4	0x00016F0010000B08 0x00016F0010000B28 0x00016F0010000B48 0x00016F0010000B68 0x00016F0010000B88	0x00016F0010000B0C 0x00016F0010000B2C 0x00016F0010000B4C 0x00016F0010000B6C 0x00016F0010000B8C	NCB	See page 759
USBC_DOEPTSIZ0 USBC_DOEPTSIZ1 USBC_DOEPTSIZ2 USBC_DOEPTSIZ3 USBC_DOEPTSIZ4	0x00016F0010000B10 0x00016F0010000B30 0x00016F0010000B50 0x00016F0010000B70 0x00016F0010000B90	0x00016F0010000B14 0x00016F0010000B34 0x00016F0010000B54 0x00016F0010000B74 0x00016F0010000B94	NCB	See page 759
USBC_PCGCCTL	0x00016F0010000E00	0x00016F0010000E04	NCB	See page 760
USBC_NPTXDFIFO0	0x00016F0010001000	0x00016F0010001004	NCB	See page 761

Table 21–4 USBC Registers (Continued)

Register	Address (NCB, Little-Endian)	Address (NCB, Big-Endian)	CSR Type[1]	Detailed Description
USBC_NPTXDFIFO1	0x00016F0010002000	0x00016F0010002004	NCB	See page 761
USBC_NPTXDFIFO2	0x00016F0010003000	0x00016F0010003004	NCB	See page 761
USBC_NPTXDFIFO3	0x00016F0010004000	0x00016F0010004004	NCB	See page 761
USBC_NPTXDFIFO4	0x00016F0010005000	0x00016F0010005004	NCB	See page 761
USBC_NPTXDFIFO5	0x00016F0010006000	0x00016F0010006004	NCB	See page 761
USBC_NPTXDFIFO6	0x00016F0010007000	0x00016F0010007004	NCB	See page 761
USBC_NPTXDFIFO7	0x00016F0010008000	0x00016F0010008004	NCB	See page 761
USBC_GRXSTSRD	0x00016F001004001C	0x00016F0010040018	NCB	See page 731
USBC_GRXSTSPD	0x00016F0010040020	0x00016F0010040024	NCB	See page 732

1. NCB-type registers are accessed directly across the I/O Bus.

OTG Control and Status Register USBC_GOTGCTL

This register controls the behavior and reflects the status of the OTG function of the core. See Table 21–4 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:21>	—	RO	—	—	Reserved
<20>	—	RO	—	—	Present release of code incorrectly attaches this to the device-mode status bit. Software should disregard the value of this bit.
<19>	BSESVLD	RO	—	—	B-Session valid. Indicates the device mode transceiver status. Valid only when CN50XX USB core is configured as a USB device. 0 = B-session is not valid, 1 = B-session is valid.
<18>	ASESVLD	RO	—	—	A-Session valid. Indicates the host mode transceiver status. Valid only when CN50XX USB core is configured as a USB host. 0 = A-session is not valid, 1 = A-session is valid
<17>	DBNCTIME	RO	0	0	Long/short debounce time. In the present version of the core, this bit will only read as 0.
<16>	CONIDSTS	RO	—	—	Connector ID status. Indicates the connector ID status on a connect event. 0 = the CN50XX USB core is in A-device mode 1 = the CN50XX USB core is in B-device mode
<15:12>	—	RO	—	—	Reserved
<11>	DEVHNPEN	R/W	0	0	Device HNP enabled. Since CN50XX USB core is not HNP capable, this bit is 0.
<10>	HSTSETHNPEN	R/W	0	0	Host set HNP enable. Since CN50XX USB core is not HNP capable, this bit is 0.
<9>	HNPREQ	R/W	0	0	Host request. Since CN50XX USB core is not HNP capable, this bit is 0.
<8>	HSTNEGSCS	R/W	0	0	Host negotiation success. Since CN50XX USB core is not HNP capable, this bit is 0.
<7:2>	—	RO	—	—	Reserved
<1>	SESREQ	R/W	0	0	Session request. Since CN50XX USB core is not HNP capable, this bit is 0.
<0>	SESREQSCS	R/W	0	0	Session request success. Since CN50XX USB core is not HNP capable, this bit is 0.

OTG Interrupt Register
USBC_ GOTGINT

The application reads this register whenever there is an OTG interrupt and clears the bits in this register to clear the OTG interrupt. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:20>	—	RO	—	—	Reserved.
<19>	DBNCEDONE	R/W1C	0	0	Debounce done. In the present version of the code this bit is tied to 0.
<18>	ADEVTOUTCHG	R/W1C	0	0	A-device timeout change. Since CN50XX USB core is not HNP or SRP capable, this bit is 0.
<17>	HSTNEGDET	R/W1C	0	0	Host negotiation detected. Since CN50XX USB core is not HNP or SRP capable, this bit is 0.
<16:10>	—	RO	—	—	Reserved.
<9>	HSTNEGSUCSTSCHNG	R/W1C	0	0	Host negotiation success status change. Since CN50XX USB core is not HNP or SRP capable, this bit is 0.
<8>	SESREQSUCSTSCHNG	R/W1C	0	0	Session request success status change. Since CN50XX USB core is not HNP or SRP capable, this bit is 0.
<7:3>	—	RO	—	—	Reserved.
<2>	SESENDDET	R/W1C	0	0	Session end detected. Since CN50XX USB core is not HNP or SRP capable, this bit is 0.
<1>	—	RO	—	—	Reserved.
<0>	—	RO	—	—	Reserved. Present version of the code ties this to device-mode. Software should disregard the value of this bit.

USB Core AHB Configuration Register
USBC_ GAHBCFG

This register can be used to configure the core after power-on or a change in mode of operation. It mainly contains AHB system-related configuration parameters. The AHB is the processor interface to the CN50XX USB core. In general, software need not know about this interface except to program the values as specified. The application must program this register as part of the CN50XX USB core initialization. Do not change this register after the initial programming. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:9>	—	RO	—	—	Reserved.
<8>	PTXFEMPLVL	R/W	0	1	Periodic TxFIFO empty level. Indicates when GINTSTS[PTXFEMP] is triggered. Software should set this bit to 1. This bit is used only in slave mode. 0 = GINTSTS[PTXFEMP] indicates that the periodic TxFIFO is half empty 1 = GINTSTS[PTXFEMP] indicates that the periodic TxFIFO is completely empty
<7>	NPTXFEMPLVL	R/W	0	1	Nonperiodic TxFIFO empty level. Indicates when GINTSTS[NPTXFEMP] is triggered. Software should set this bit to 1. This bit is used only in slave mode. 0 = GINTSTS[NPTXFEMP] indicates that the nonperiodic TxFIFO is half empty 1 = GINTSTS[NPTXFEMP] indicates that the nonperiodic TxFIFO is completely empty
<6>	—	RO	—	—	Reserved.
<5>	DMAEN	R/W	0	0	DMA enable. 0 = USB core operates in slave mode 1 = USB core operates in a DMA mode
<4:1>	HBSTLEN	R/W	0x0	0x0	Burst length/type. This field has no effect and should be left as 0x0.
<0>	GLBLINTRMSK	R/W	0	0	Global interrupt mask. The application uses this bit to mask or unmask the interrupt line assertion to itself. Regardless of this bit's setting, the interrupt status registers are updated by the core. Software should set this field to 1. 0 = Mask the interrupt assertion to the application. 1 = Unmask the interrupt assertion to the application.

USB Core Configuration Register
USBC_GUSBCFG

This register can be used to configure the USB core after power-on or a changing to host mode or device mode. It contains USB and USB-PHY related configuration parameters. The application must program this register before starting any transactions on either the AHB or the USB. Do not make changes to this register after the initial programming. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:17>	—	RO	—	—	Reserved.
<16>	OTGI2CSEL	RO	0	0	UTMIFS or I^2C interface select. This bit is always 0.
<15>	PHYLPWRCLKSEL	R/W	0	0	PHY low-power clock select. Selects either 480-MHz or 48-MHz (low-power) PHY mode. In FS and LS modes, the PHY can usually operate on a 48-MHz clock to save power. Software should set this bit to 0. 0 = 480-MHz internal PLL clock 1 = 48-MHz external clock In 480-MHz mode, the UTMI interface operates at either 60- or 30-MHz, depending upon whether 8- or 16-bit data width is selected. In 48-MHz mode, the UTMI interface operates at 48-MHz in FS mode and at either 48- or 6-MHz in LS mode (depending on the PHY vendor). This bit drives the utmi_fsls_low_power core output signal, and is valid only for UTMI+ PHYs.
<14>	—	RO	—	—	Reserved.
<13:10>	USBTRDTIM	R/W	0x5	0x5	USB turnaround time. Sets the turnaround time in PHY clocks. Specifies the response time for a MAC request to the packet FIFO controller (PFC) to fetch data from the DFIFO (SPRAM). This must be programmed to 0x5.
<9>	HNPCAP	RO	0	0	HNP-capable. This bit is always 0.
<8>	SRPCAP	RO	0	0	SRP-capable. This bit is always 0.
<7>	DDRSEL	R/W	0	0	ULPI DDR select. Software should set this bit to 0.
<6>	PHYSEL	WO	0	0	USB 2.0 high-speed PHY or USB 1.1 full-speed serial. Software should set this bit to 0.
<5>	FSINTF	WO	0	0	Full-speed serial interface select. Software should set this bit to 0.
<4>	ULPI_UTMI_SEL	RO	0	0	ULPI or UTMI+ select. This bit is always 0.
<3>	PHYIF	RO	1	1	PHY Interface. This bit is always 1.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<2:0>	TOUTCAL	R/W	0x0	0x0	HS/FS timeout calibration (TOutCal). The number of PHY clocks that the application programs in this field is added to the high-speed/full-speed interpacket timeout duration in the core to account for any additional delays introduced by the PHY. This may be required, since the delay introduced by the PHY in generating the linestate condition may vary from one PHY to another. The USB standard timeout value for high-speed operation is 736 to 816 (inclusive) bit times. The USB standard timeout value for full-speed operation is 16 to 18 (inclusive) bit times. The application must program this field based on the speed of enumeration. The number of bit times added per PHY clock are: High-speed operation: ● One 30-MHz PHY clock = 16 bit times ● One 60-MHz PHY clock = 8 bit times Full-speed operation: ● One 30-MHz PHY clock = 0.4 bit times ● One 60-MHz PHY clock = 0.2 bit times ● One 48-MHz PHY clock = 0.25 bit times

USB Core Reset Register
USBC_GRSTCTL

The application uses this register to reset various hardware features inside the USB core. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	AHBIDLE	RO	1	1	AHB master idle. Indicates that the AHB master state machine is in the IDLE condition.
<30>	DMAREQ	RO	0	0	DMA request signal. Indicates that the DMA request is in progress. Used for debug purposes.
<29:11>	—	RO	—	—	Reserved.
<10:6>	TXFNUM	R/W	0x0	0x0	TxFIFO number. Indicates the FIFO number that must be flushed using TXFFLSH. This field must not be changed until the USB core clears TXFFLSH. 0x0 = nonperiodic TxFIFO flush 0x1 = periodic TxFIFO1 flush in device mode or periodic TxFIFO flush in Host mode 0x2 = periodic TxFIFO2 flush in device mode ... 0xF = periodic TxFIFO15 flush in device mode 0x10 = Flush all the periodic and nonperiodic TxFIFOs in the USB core
<5>	TXFFLSH	R/W	0	0	TxFIFO flush. This bit selectively flushes a single or all transmit FIFOs, but cannot do so if the core is in the midst of a transaction. The application must only write this bit after checking that the core is neither writing to nor reading from the TxFIFO. The application must wait until the core clears this bit before performing any operations. This bit takes eight clock cycles (of PHY_CLK or HCLK, whichever is slower) to clear.
<4>	RXFFLSH	R/W	0	0	RxFIFO flush. The application can flush the entire RxFIFO using this bit, but must first ensure that the core is not in the middle of a transaction. The application must only write to this bit after checking that the core is neither reading from nor writing to the RxFIFO. The application must wait until the bit is cleared before performing any other operations. This bit will take eight clock cycles (of PHY_CLK or HCLK, whichever is slower) to clear.
<3>	INTKNQFLSH	R/W	0	0	IN token sequence learning-queue flush. The application writes this bit to flush the IN token sequence learning queue.
<2>	FRMCNTRRST	R/W	0	0	Host frame counter reset. The application writes this bit to reset the (micro)frame number counter inside the USB core. When the (micro)frame counter is reset, the subsequent SOF sent out by the USB core has a (micro)frame number of 0.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<1>	HSFTRST	R/W	0	0	HCLK soft reset. The application uses this bit to flush the control logic in the AHB clock domain. Only AHB clock domain pipelines are reset. • FIFOs are not flushed with this bit. • All state machines in the AHB clock domain are reset to the idle state after terminating the transactions on the AHB, following the protocol. • CSR control bits used by the AHB clock domain state machines are cleared. • To clear this interrupt, status mask bits that control the interrupt status and are generated by the AHB clock domain state machine are cleared. • Because interrupt status bits are not cleared, the application can get the status of any core events that occurred after it set this bit. This is a self-clearing bit that the USB core clears after all necessary logic is reset in the core. This may take several clock cycles, depending on the USB core's current state.
<0>	CSFTRST	R/W	0	0	USB core soft reset. Resets the HCLK and PHY_CLOCK domains as follows: • Clears the interrupts and all the CSR registers except the following register bits: USBC_PCGCCTL[RSTPDWNMODULE] USBC_PCGCCTL[GATEHCLK] USBC_PCGCCTL[PWRCLMP] USBC_PCGCCTL[STOPPPHYLPWRCLKSELCLK] USBC_GUSBCFG[PHYLPWRCLKSEL] USBC_GUSBCFG[DDRSEL] USBC_GUSBCFG[PHYSEL] USBC_GUSBCFG[FSINTF] USBC_GUSBCFG[ULPI_UTMI_SEL] USBC_GUSBCFG[PHYIF] USBC_HCFG[FSLSPCLKSEL] USBC_DCFG[DEVSPD] • All module state machines except the AHB slave unit are reset to the IDLE state, and all the transmit FIFOs and the receive FIFO are flushed. • Any transactions on the AHB master are terminated as soon as possible, after gracefully completing the last data phase of an AHB transfer. Any transactions on the USB are terminated immediately. The application can write to this bit any time it wants to reset the core. This is a self-clearing bit and the core clears this bit after all the necessary logic is reset in the core, which may take several clock cycles, depending on the current state of the USB core. Once this bit is cleared, software should wait at least three PHY clock cycles before doing any access to the PHY domain (synchronization delay). Software should also should check that bit [31] of this register is 1 (AHB master is IDLE) before starting any operation. Typically software reset is used during software development and also when you dynamically change the PHY selection bits in the USB configuration registers listed above. When you change the PHY, the corresponding clock for the PHY is selected and used in the PHY domain. Once a new clock is selected, the PHY domain has to be reset for proper operation.

USB Core Interrupt Register
USBC_ GINTSTS

This register interrupts the application for system-level events in the current mode of operation (device mode or host mode). Some of the bits in this register are valid only in host mode, while others are valid in device mode only. This register also indicates the current mode of operation. In order to clear the interrupt status bits of type R/W1C, the application must write 1 into the bit. The FIFO status interrupts are read only; once software reads from or writes to the FIFO while servicing these interrupts, FIFO interrupt conditions are cleared automatically. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	WKUPINT	R/W1C	0x0	0x0	Resume/remote wakeup detected interrupt. In device mode, this interrupt is asserted when a resume is detected on the USB. In host mode, this interrupt is asserted when a remote wakeup is detected on the USB.
<30>	SESSREQINT	R/W1C	0x0	0x0	Session request/new session detected interrupt. In host mode, this interrupt is asserted when a session request is detected from the device. In device mode, this interrupt is asserted when the utmiotg_bvalid signal goes high.
<29>	DISCONNINT	R/W1C	0x0	0x0	Disconnect detected interrupt. Asserted when a device disconnect is detected.
<28>	CONIDSTSCHNG	R/W1C	0x0	0x0	Connector ID status change. The USB core sets this bit when there is a change in connector ID status.
<27>	—	RO	—	—	Reserved.
<26>	PTXFEMP	RO	0x0	0x0	Periodic TxFIFO empty. Asserted when the periodic transmit FIFO is either half or completely empty and there is space for at least one entry to be written in the periodic request queue. The half- or completely-empty status is determined by USBC_GAHBCFG[PTXFEMPLVL].
<25>	HCHINT	RO	0x0	0x0	Host channels interrupt. The USB core sets this bit to indicate that an interrupt is pending on one of the channels of the core (in host mode). The application must read USBC_HAINT to determine the exact number of the channel on which the interrupt occurred, and then read the corresponding USBC_HCINT*n* register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in USBC_HCINT*n* to clear this bit.
<24>	PRTINT	RO	0x0	0x0	Host port interrupt. The USB core sets this bit to indicate a change in port status of one of the CN50XX USB core ports in host mode. The application must read USBC_HPRT to determine the exact event that caused this interrupt. The application must clear the appropriate status bit in USBC_HPRT to clear this bit.
<23>	—	RO	—	—	Reserved.
<22>	FETSUSP	R/W1C	0x0	0x0	Data fetch suspended. This interrupt is valid only in DMA mode. This interrupt indicates that the core has stopped fetching data for IN endpoints due to the unavailability of TxFIFO space or request-queue space. This interrupt is used by the application for an endpoint mismatch algorithm.
<21>	INCOMPLP	R/W1C	0x0	0x0	Incomplete periodic transfer. In host mode, the core sets this interrupt bit when there are incomplete periodic transactions still pending that are scheduled for the current microframe. Incomplete Isochronous OUT Transfer (INCOMPISOOUT) In device mode, the core sets this interrupt to indicate that there is at least one isochronous OUT endpoint on which the transfer is not completed in the current microframe. This interrupt is asserted along with EOPF.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<20>	INCOMPISOIN	R/W1C	0x0	0x0	Incomplete isochronous IN transfer. The USB core sets this interrupt to indicate that there is at least one isochronous IN endpoint on which the transfer is not completed in the current microframe. This interrupt is asserted along with EOPF.
<19>	OEPINT	RO	0x0	0x0	OUT endpoints interrupt. The core sets this bit to indicate that an interrupt is pending on one of the OUT endpoints of the core (in device mode). The application must read USBC_DAINT to determine the exact number of the OUT endpoint on which the interrupt occurred, and then read the corresponding USBC_DOEPINT*n* register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding USBC_DOEPINT*n* to clear this bit.
<18>	IEPINT	RO	0x0	0x0	IN endpoints interrupt. The USB core sets this bit to indicate that an interrupt is pending on one of the IN endpoints of the core (in device mode). The application must read USBC_DAINT to determine the exact number of the IN endpoint on which the interrupt occurred, and then read the corresponding USBC_DIEPINT*n* resister to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding USBC_DIEPINT*n* register to clear this bit.
<17>	EPMIS	R/W1C	0x0	0x0	Endpoint mismatch interrupt. Indicates that an IN token has been received for a nonperiodic endpoint, but the data for another endpoint is present in the top of the nonperiodic transmit FIFO and the IN endpoint mismatch count programmed by the application has expired.
<16>	—	RO	—	—	Reserved.
<15>	EOPF	R/W1C	0x0	0x0	End of periodic frame interrupt. Indicates that the period specified in USBC_DCFG[PERFRINT] has been reached in the current microframe.
<14>	ISOOUTDROP	R/W1C	0x0	0x0	Isochronous OUT packet dropped interrupt. The core sets this bit when it fails to write an isochronous OUT packet into the RxFIFO because the RxFIFO doesn't have enough space to accommodate a maximum-packet-size packet for the isochronous OUT endpoint.
<13>	ENUMDONE	R/W1C	0x0	0x0	Enumeration done. The core sets this bit to indicate that speed enumeration is complete. The application must read USBC_DSTS to obtain the enumerated speed.
<12>	USBRST	R/W1C	0x0	0x0	USB reset. The USB core sets this bit to indicate that a reset is detected on the USB.
<11>	USBSUSP	R/W1C	0x0	0x0	USB suspend. The USB core sets this bit to indicate that a suspend was detected on the USB. The USB core enters the suspended state when there is no activity on the phy_line_state_i signal for an extended period of time.
<10>	ERLYSUSP	R/W1C	0x0	0x0	Early suspend. The USB core sets this bit to indicate that an Idle state has been detected on the USB for 3 ms.
<9>	I2CINT	R/W1C	0x0	0x0	I^2C interrupt. This bit is always 0.
<8>	ULPICKINT	R/W1C	0x0	0x0	ULPI carkit interrupt. This bit is always 0.
<7>	GOUTNAKEFF	RO	0x0	0x0	Global OUT NAK effective. Indicates that USBC_DCTL[SGOUTNAK], set by the application, has taken effect in the USB core. This bit can be cleared by writing to USBC_DCTL[CGOUTNAK].

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<6>	GINNAKEFF	RO	0x0	0x0	Global IN nonperiodic NAK effective. Indicates that USBC_DCTL[SGNPINNAK], set by the application, has taken effect in the USB core (i.e. the USB core has sampled the global IN NAK bit set by the application). This bit can be cleared by clearing USBC_DCTL[CGNPINNAK]. This interrupt does not necessarily mean that a NAK handshake is sent out on the USB. The STALL bit takes precedence over the NAK bit.
<5>	NPTXFEMP	RO	0x0	0x0	Nonperiodic TxFIFO empty. This interrupt is asserted when the nonperiodic TxFIFO is either half or completely empty, and there is space for at least one entry to be written to the nonperiodic transmit request queue. The half or completely empty status is determined by USBC_GAHBCFG[NPTXFEMPLVL].
<4>	RXFLVL	RO	0x0	0x0	RxFIFO non-empty. Indicates that there is at least one packet pending to be read from the RxFIFO.
<3>	SOF	R/W1C	0x0	0x0	Start of (micro)frame. In host mode, the USB core sets this bit to indicate that an SOF (FS), micr-SOF (HS), or keep-alive (LS) is transmitted on the USB. The application must write a 1 to this bit to clear the interrupt. In device mode, in the USB core sets this bit to indicate that an SOF token has been received on the USB. The application can read the device-status register to get the current (micro)frame number. This interrupt is seen only when the core is operating at either HS or FS.
<2>	OTGINT	RO	0x0	0x0	OTG interrupt. The USB core sets this bit to indicate an OTG protocol event. The application must read USBC_GOTGINT to determine the exact event that caused this interrupt. The application must clear the appropriate status bit in USBC_GOTGINT to clear this bit.
<1>	MODEMIS	R/W1C	0x0	0x0	Mode mismatch interrupt. The USB core sets this bit when the application is trying to access: ●a host mode register, when the core is operating in device mode ●a device mode register, when the core is operating in host mode The register access is completed on the AHB with an OKAY response, but is ignored by the core internally and does not affect the operation of the USB core.
<0>	CURMOD	RO	0x0	0x0	Current mode of operation. Indicates the current mode of operation. 0 = device mode, 1 = host mode

Core Interrupt Mask Register
USBC_GINTMSK

This register works with USBC_GINTSTS to interrupt the application. When an interrupt bit is masked, the interrupt associated with that bit is not generated. However, the USBC_GINTSTS bit corresponding to that interrupt is set.

For all interrupts, 0 = mask the interrupt, 1 =1 unmask interrupt. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	WKUPINTMSK	R/W	0	0	Resume/remote wakeup detected interrupt mask.
<30>	SESSREQINTMSK	R/W	0	0	Session request/new session detected interrupt mask.
<29>	DISCONNINTMSK	R/W	0	0	Disconnect detected interrupt mask.
<28>	CONIDSTSCHNGMSK	R/W	0	0	Connector ID status change mask.
<27>	—	RO	—	—	Reserved.
<26>	PTXFEMPMSK	R/W	0	0	Periodic TxFIFO empty mask.
<25>	HCHINTMSK	R/W	0	0	Host channels interrupt mask.
<24>	PRTINTMSK	R/W	0	0	Host port interrupt mask.
<23>	—	RO	—	—	Reserved.
<22>	FETSUSPMSK	R/W	0	0	Data fetch suspended mask.
<21>	INCOMPISOOUTMSK	R/W	0	0	Incomplete periodic transfer mask (host mode). Incomplete isochronous OUT transfer mask (device mode).
<20>	INCOMPISOINMSK	R/W	0	0	Incomplete isochronous IN transfer mask.
<19>	OEPINTMSK	R/W	0	0	OUT endpoints interrupt mask.
<18>	INEPINTMSK	R/W	0	0	IN endpoints interrupt mask.
<17>	EPMISMSK	R/W	0	0	Endpoint mismatch interrupt mask.
<16>	—	RO	—	—	Reserved.
<15>	EOPFMSK	R/W	0	0	End of periodic frame interrupt mask.
<14>	ISOOUTDROPMSK	R/W	0	0	Isochronous OUT packet dropped interrupt mask.
<13>	ENUMDONEMSK	R/W	0	0	Enumeration done mask.
<12>	USBRSTMSK	R/W	0	0	USB reset mask.
<11>	USBSUSPMSK	R/W	0	0	USB suspend mask.
<10>	ERLYSUSPMSK	R/W	0	0	Early suspend mask.
<9>	I2CINT	R/W	0	0	I^2C interrupt mask.
<8>	ULPICKINTMSK	R/W	0	0	ULPI carkit interrupt mask, I2C carkit interrupt mask.
<7>	GOUTNAKEFFMSK	R/W	0	0	Global OUT NAK effective mask.
<6>	GINNAKEFFMSK	R/W	0	0	Global nonperiodic IN NAK effective mask.
<5>	NPTXFEMPMSK	R/W	0	0	Nonperiodic TxFIFO empty mask.
<4>	RXFLVLMSK	R/W	0	0	Receive FIFO non-empty mask.
<3>	SOFMSK	R/W	0	0	Start of (micro)frame mask.
<2>	OTGINTMSK	R/W	0	0	OTG interrupt mask.
<1>	MODEMISMSK	R/W	0	0	Mode mismatch interrupt mask.
<0>	—	RO	—	—	Reserved.

Receive-Status Debug Read Register, Host Mode USBC_GRXSTSRH

A read to this register returns the contents of the top of the receive FIFO.

NOTE: This description is only valid when the core is in host mode. For device mode, use USBC_GRXSTSRD instead. USBC_GRXSTSRH and USBC_GRXSTSRD are physically the same register and share the same offset in the CN50XX USB core. The address difference specified in this document is for software clarity and is actually ignored by the hardware.

See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:21>	—	RO	—	—	Reserved.
<20:17>	PKTSTS	RO	0x0	0x0	Packet status. Indicates the status of the received packet. 0x0 = IN data packet received 0x3 = IN transfer completed (triggers an interrupt) 0x5 = data toggle error (triggers an interrupt) 0x7 = channel halted (triggers an interrupt) All other values = reserved
<16:15>	DPID	RO	0x0	0x0	Data PID. 0 = data0 2 = data1 1 = data2 3 = Mdata
<14:4>	BCNT	RO	0x0	0x0	Byte count. Indicates the byte count of the received IN data packet.
<3:0>	CHNUM	RO	0x0	0x0	Channel number. Indicates the channel number to which the current received packet belongs.

Receive-Status Read and Pop Register, Host Mode USBC_GRXSTSPH

A read to the receive-status read and pop register returns and additionally pops the top data entry out of the RxFIFO.

NOTE: This description is only valid when the core is in host mode. For device mode, use USBC_GRXSTSRD instead. USBC_GRXSTSPH and USBC_GRXSTSPD are physically the same register and share the same offset in the CN50XX USB core. The address difference specified in this document is for software clarity and is actually ignored by the hardware.

See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:21>	—	RO	—	—	Reserved.
<20:17>	PKTSTS	RO	0x0	0x0	Packet status. Indicates the status of the received packet. 0x0 = IN data packet received 0x3 = IN transfer completed (triggers an interrupt) 0x5 = data toggle error (triggers an interrupt) 0x7 = channel halted (triggers an interrupt) All other values = reserved
<16:15>	DPID	RO	0x0	0x0	Data PID. 0 = data0 2 = data1 1 = data2 3 = Mdata

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<14:4>	BCNT	RO	0x0	0x0	Byte count. Indicates the byte count of the received IN data packet.
<3:0>	CHNUM	RO	0x0	0x0	Channel number. Indicates the channel number to which the current received packet belongs.

Receive-Status Debug Read Register, Device Mode USBC_GRXSTSRD

A read to the receive-status debug read register returns the contents of the top of the receive FIFO.

NOTE: This description is only valid when the core is in DEVICE mode. For HOST mode, use USBC_GRXSTSRH instead. USBC_GRXSTSPH and USBC_GRXSTSPD are physically the same register and share the same offset in the CN50XX USB core. The address difference specified in this document is for software clarity and is actually ignored by the hardware.

See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:25>	—	RO	—	—	Reserved.
<24:21>	FN	RO	0x0	0x0	Frame number. Indicates the least significant four bits of the (micro)frame number in which the packet is received on the USB. This field is supported only when the isochronous OUT endpoints are supported.
<20:17>	PKTSTS	RO	0x0	0x0	Packet status. Indicates the status of the received packet. 0x1 = Global OUT NAK (triggers an interrupt) 0x2 = OUT data packet received 0x4 = SETUP transaction completed (triggers an interrupt) 0x6 = SETUP data packet received All other values = reserved
<16:15>	DPID	RO	0x0	0x0	Data PID. 0 = data0 2 = data1 1 = data2 3 = Mdata
<14:4>	BCNT	RO	0x0	0x0	Byte count. Indicates the byte count of the received data packet.
<3:0>	EPNUM	RO	0x0	0x0	Endpoint number. Indicates the endpoint number to which the current received packet belongs.

Receive-Status Debug Read Register, Device Mode USBC_ GRXSTSPD

A read to the receive-status read and pop register returns and additionally pops the top data entry out of the RxFIFO.

NOTE: This description is only valid when the core is in device mode. For host mode, use USBC_GRXSTSRH instead. USBC_GRXSTSPH and USBC_GRXSTSPD are physically the same register and share the same offset in the CN50XX USB core. The address difference specified in this document is for software clarity and is actually ignored by the hardware.

See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:25>	—	RO	—	—	Reserved.
<24:21>	FN	RO	0x0	0x0	Frame number. Indicates the least significant four bits of the (micro)frame number in which the packet is received on the USB. This field is supported only when the isochronous OUT endpoints are supported.
<20:17>	PKTSTS	RO	0x0	0x0	Packet status. Indicates the status of the received packet. 0x1 = Global OUT NAK (triggers an interrupt) 0x2 = OUT data packet received 0x4 = SETUP transaction completed (triggers an interrupt) 0x6 = SETUP data packet received All other values = reserved
<16:15>	DPID	RO	0x0	0x0	Data PID. 0 = data0 2 = data1 1 = data2 3 = Mdata
<14:4>	BCNT	RO	0x0	0x0	Byte count. Indicates the byte count of the received data packet.
<3:0>	EPNUM	RO	0x0	0x0	Endpoint number. Indicates the endpoint number to which the current received packet belongs.

Receive FIFO Size Register USBC_ GRXFSIZ

The application can program the RAM size that must be allocated to the RxFIFO. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	—	RO	—	—	Reserved.
<15:0>	RXFDEP	R/W	0x720	0x1C8	RxFIFO depth. This value is in terms of 32-bit words.

Nonperiodic Transmit FIFO Size Register USBC_GNPTXFSIZ

The application can program the RAM size and the memory start address for the nonperiodic TxFIFO. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	NPTXFDEP	R/W	0x720	0x390	Nonperiodic TxFIFO depth. This value is in terms of 32-bit words. Minimum value is 16. Maximum value is 32768.
<15:0>	NPTXFSTADDR	R/W	0x720	0x1C8	Nonperiodic transmit RAM start address. Contains the memory start address for nonperiodic transmit FIFO RAM.

Nonperiodic Transmit FIFO/Queue Status Register USBC_GNPTXSTS

The application can program the RAM size and the memory start address for the nonperiodic TxFIFO. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	—	RO	—	—	Reserved.
<30:24>	NPTXQTOP	RO	0x0	0x0	Top of the nonperiodic transmit request queue. Entry in the nonperiodic Tx request queue that is currently being processed by the MAC. Bits [30:27] = channel/endpoint number Bits [26:25]: • 0x0 = IN/OUT token • 0x1 = Zero-length transmit packet (device IN/host OUT) • 0x2 = PING/CSPLIT token • 0x3 = channel halt command Bit [24]: terminate (last entry for selected channel/ endpoint)
<23:16>	NPTXQSPCAVAIL	RO	0x0	0x0	Nonperiodic transmit request queue space available. Indicates the amount of free space available in the nonperiodic transmit request queue. This queue holds both IN and OUT requests in host mode. Device mode has only IN requests. 0x0 = nonperiodic transmit-request queue is full 0x1 = one location available 0x2 = two locations available ... 0x8 = eight locations available All others = reserved
<15:0>	NPTXFSPCAVAIL	RO	0x0	0x0	Nonperiodic TxFIFO space available. Indicates the amount of free space available in the nonperiodic TxFIFO. Values are in terms of 32-bit words. 0x0 = nonperiodic TxFIFO is full 0x1 = one location available 0x2 = two locations available ... 0x8 = eight locations available All others = reserved

Synopsys ID Register
USBC_GSNPSID

This is a read-only register that contains the release number of the core being used. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	SYNOPSYSID	RO	—	—	Release number of the core being used, in the format: 0x4F54<version>A. 0x4F54220A = pass 1.x, 0x4F54240A = pass 2.x

User Hardware Configuration Register 1
USBC_GHWCFG1

This register contains the logical endpoint directions of the CN50XX USB core. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	EPDIR	RO	0x0	0x0	Endpoint direction. Two bits per endpoint represent the direction. 00 = bidirectional (IN and OUT) endpoint 01 = IN endpoint 10 = OUT endpoint 11 = reserved Bits <31:30>: Endpoint 15 direction Bits <29:28>: Endpoint 14 direction ... Bits <3:2>: Endpoint 1 direction Bits<1:0>: Endpoint 0 direction (always BIDIR)

User Hardware Configuration Register 2
USBC_GHWCFG2

This register contains configuration options of the CN50XX USB core. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	—	RO	—	—	Reserved.
<30:26>	TKNQDEPTH	RO	0x1E	0x1E	Device-mode IN token sequence learning queue depth. Range: 0x0–0x30.
<25:24>	PTXQDEPTH	RO	0x2	0x2	Host-mode periodic request-queue depth. 00 = 2, 01 = 4, 10 = 8, 11 = reserved
<23:22>	NPTXQDEPTH	RO	0x2	0x2	Nonperiodic request-queue depth. 00 = 2, 01 = 4, 10 = 8, 11 = reserved
<21:20>	—	RO	—	—	Reserved.
<19>	DYNFIFOSIZING	RO	1	1	Dynamic FIFO sizing enabled. 0 = not enabled, 1 = enabled
<18>	PERIOSUPPORT	RO	1	1	Periodic OUT channels supported in host mode. 0 = not supported, 1 = supported
<17:14>	NUMHSTCHN1	RO	0x7	0x7	Number of host channels. Indicates the number of host channels supported by the USB core in host mode. The range of this field is 0–15: 0 specifies 1 channel, ... 15 specifies 16 channels.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<13:10>	NUMDEVEPS	RO	0x4	0x4	Number of device endpoints. Indicates the number of device endpoints supported by the USB core in device mode in addition to control endpoint 0. The range of this field is 1–15.
<9:8>	FSPHYTYPE	RO	0x0	0x0	Full-speed PHY interface type. 00 = Full-speed interface not supported 01 = Dedicated full-speed interface 10 = Full-speed pins shared with UTMI+ pins 11 = Full-speed pins shared with ULPI pins
<7:6>	HSPHYTYPE	RO	0x1	0x1	High-speed PHY interface type. 00 = High-speed interface not supported 01 = UTMI+ 10 = ULPI 11 = UTMI+ and ULPI
<5>	SINGPNT	RO	0	0	Single point. 0 = multipoint application, 1 = single-point application
<4:3>	OTGARCH	RO	0x1	0x1	Architecture. 00 = slave only 01 = external DMA 10 = internal DMA 11 = reserved
<2:0>	OTGMODE	RO	0x2	0x2	Mode of operation. 0x0 = HNP- and SRP-capable OTG (host and device) 0x1 = SRP-Capable OTG (host and device) 0x2 = Non-HNP and Non-SRP Capable OTG (host and device) 0x3 = SRP-capable Device 0x4 = non-OTG Device 0x5 = SRP-capable host 0x6 = non-OTG host 0x7 = reserved

User Hardware Configuration Register 3 USBC_GHWCFG3

This register contains configuration options of the CN50XX USB core. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	DFIFODEPTH	RO	0x720	0x720	DFIFO depth. This value is in terms of 32-bit words. Minimum value is 32. Maximum value is 32768.
<15:13>	—	RO	—	—	Reserved.
<12>	AHBPHYSYNC	RO	0	0	AHB and PHY synchronous. Indicates whether AHB and PHY clocks are synchronous to each other. 0 = not tied together, 1 = tied together
<11>	RSTTYPE	RO	1	1	Reset style for clocked always blocks in RTL. 0 = asynchronous reset is used in the USB core 1 = synchronous reset is used in the USB core
<10>	OPTFEATURE	RO	1	1	Optional features removed. Indicates whether the user ID register, GPIO interface ports, and SOF toggle and counter ports were removed for gate count optimization. 0 = not removed, 1 = removed

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<9>	VENDOR_CONTROL_INTERFACE_SUPPORT	RO	1	1	Vendor-control interface support. 0 = vendor-control interface is not available on the USB core. 1 = vendor-control interface is available.
<8>	I2C_SELECTION	RO	0	0	I^2C selection 0 = I^2C Interface is not available on the USB core. 1 = I^2C interface is available.
<7>	OTGEN	RO	1	1	OTG function enabled. The application uses this bit to indicate the CN50XX USB cores OTG capabilities. 0 = not OTG capable 1 = OTG Capable
<6:4>	PKTSIZEWIDTH	RO	0x6	0x6	Width of packet-size counters. 0x0 = 4 bits 0x1 = 5 bits ... 0x6 = 10 bits 0x7 = reserved
<3:0>	XFERSIZEWIDTH	RO	0x8	0x8	Width of transfer-size counters. 0x0 = 11 bits 0x1 = 12 bits ... 0x8 = 19 bits All others = reserved

User Hardware Configuration Register 4 USBC_GHWCFG4

This register contains configuration options of the CN50XX USB core. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:30>	—	RO	—	—	Reserved.
<29:26>	NumDevModInEnd	RO	0x2	0x2	Enable dedicated transmit FIFO for device IN endpoints.
<25>	EnDedTrFifo	RO	0	0	Enable dedicated transmit FIFO for device IN endpoints.
<24>	SESSENDFLTR	RO	0	0	session_end filter enabled. 0 = no filter 1 = filter enabled
<23>	BVALIDFLTR	RO	0	0	b_valid filter enabled. 0 = no filter 1 = filter enabled
<22>	AVALIDFLTR	RO	0	0	a_valid filter enabled 0 = no filter 1 = filter enabled
<21>	VBUSVALIDFLTR	RO	1	1	vbus_valid filter enabled. 0 = no filter 1 = filter enabled
<20>	IDDGFLTR	RO	1	1	iddig filter enabled. 0 = no filter 1 = filter enabled
<19:16>	NUMCTLEPS	RO	0x4	0x4	Number of device-mode control endpoints in addition to endpoint 0. Range: 0x1-0xF.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<15:14>	PHYDATAWIDTH	RO	0x1	0x1	UTMI+ PHY/ULPI-to-internal UTMI+ wrapper data width. When a ULPI PHY is used, an internal wrapper converts ULPI to UTMI+. 00 = 8 bits 01 = 16 bits 10 = 8/16 bits, software selectable 11 = reserved
<13:6>	—	RO	—	—	Reserved.
<5>	AHBFREQ	RO	1	1	Minimum AHB frequency less than 60 MHz. 0 = no, 1 = yes
<4>	ENABLEPWROPT	RO	0	0	Enable power optimization? (EnablePwrOpt) 0 = no, 1 = yes
<3:0>	NUMDEVPERIOEPS	RO	0x4	0x4	Number of device mode periodic IN endpoints. Range: 0–0xF.

Host Periodic Transmit FIFO Size Register USBC_HPTXFSIZ

This register holds the size and the memory start address of the periodic TxFIFO. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	PTXFSIZE	R/W	0x100	0x1C8	Host periodic TxFIFO depth. This value is in terms of 32-bit words. Minimum value is 16, maximum value is 32768.
<15:0>	PTXFSTADDR	R/W	0xE40	0x390	Host periodic TxFIFO start address.

Device Periodic Transmit FIFO*n* Size Registers USBC_DPTXFSIZ(1..4)

This register holds the memory start address of each periodic TxFIFO to be implemented in device mode. Each periodic FIFO holds the data for one periodic IN endpoint. This register is repeated for each periodic FIFO instantiated. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	DPTXFSIZE	RO	0x768	0x768	Device periodic TxFIFO size. This value is in terms of 32-bit words. Minimum value is 4, maximum value is 768.
<15:0>	PTXFSTADDR	R/W	0xE40	0x390	Device periodic TxFIFO RAM start address. Holds the start address in the RAM for this periodic FIFO.

Host Configuration Register USBC_HCFG

This register configures the core after power-on. Do not make changes to this register after initializing the host. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:3>	—	RO	—	—	Reserved.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<2>	FSLSSUPP	R/W	0	0	FS- and LS-only support. The application uses this bit to control the USB core's enumeration speed. Using this bit, the application can make the USB core enumerate as a FS host, even if the connected device supports HS traffic. Do not make changes to this field after initial programming. 0 = HS/FS/LS, based on the maximum speed supported by the connected device 1 = FS/LS-only, even if the connected device can support HS.
<1:0>	FSLSPCLKSEL	R/W	0	0	FS/LS PHY clock select. When the USB core is in FS host mode: 00 = PHY clock is running at 30/60 MHz 01 = PHY clock is running at 48 MHz 1x = reserved When the core is in LS host mode: 00 = PHY clock is running at 30/60 MHz. When the UTMI+/ULPI PHY low-power mode is not selected, use 30/60 MHz. 01 = PHY clock is running at 48 MHz. When the UTMI+ PHY low-power mode is selected, use 48 MHz if the PHY supplies a 48-MHz clock during LS mode. 10 = PHY clock is running at 6 MHz. In USB 1.1 FS mode, use 6 MHz when the UTMI+ PHY low-power mode is selected and the PHY supplies a 6-MHz clock during LS mode. If you select a 6-MHz clock during LS mode, you must do a soft reset. 11 = reserved

Host Frame Interval Register
USBC_HFIR

This register contains the interrupt summary bits of the USBN. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	—	RO	—	—	Reserved.
<15:0>	FRINT	R/W	0x0B8F	0x0EA6	Frame interval. The value that the application programs to this field specifies the interval between two consecutive SOFs (FS) or microSOFs (HS) or Keep-Alive tokens (HS). This field contains the number of PHY clocks that constitute the required frame interval. The default value set in this field for a FS operation when the PHY clock frequency is 60 MHz. The application can write a value to this register only after USBC_HPRT[PRTENAPORT)] has been set. If no value is programmed, the USB core calculates the value based on the PHY clock specified in USBC_HCFG[FSLSPCLKSEL]. Do not change the value of this field after the initial configuration. 125 µs (PHY clock frequency for HS) 1 ms (PHY clock frequency for FS/LS)

Host Frame Number/Frame Time Remaining Register USBC_HFNUM

This register indicates the current frame number and indicates the time remaining (in terms of the number of PHY clocks) in the current frame or microframe. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	FRREM	RO	0x0	0x0	Frame time remaining. Indicates the amount of time remaining in the current microframe (HS) or frame (FS/LS), in terms of PHY clock cycles. This field decrements on each PHY clock. When it reaches zero, this field is reloaded with the value in USBC_HFIR and a new SOF is transmitted on the USB.
<15:0>	FRNUM	RO	0x3FFF	0x0	Frame number. This field increments when a new SOF is transmitted on the USB, and is reset to 0x0 when it reaches 0x3FFF.

Host Periodic Transmit FIFO/Queue Status Register USBC_HPTXSTS

This read-only register contains the free space information for the periodic TxFIFO and the periodic transmit-request queue. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:24>	PTXQTOP	RO	0x0	0x0	Top of the periodic transmit-request queue. This indicates the entry in the periodic Tx request queue that is currently being processes by the MAC. This register is used for debugging. Bit [31]: odd/even (micro)frame • 0 = send in even (micro)frame • 1 = send in odd (micro)frame Bits [30:27]: channel/endpoint number Bits [26:25]: type • 00 = IN/OUT • 01 = Zero-length packet • 10 = CSPLIT • 11 = Disable channel command Bit [24]: terminate (last entry for the selected channel/endpoint)
<23:16>	PTXQSPCAVAIL	RO	0x0	0x0	Periodic transmit-request queue space available. Indicates the number of free locations available to be written in the periodic transmit-request queue. This queue holds both IN and OUT requests. 0x0 = periodic transmit-request queue is full 0x1 = 1 location available 0x2 = 2 locations available 0xN = N locations available (0..0x8) All others = reserved
<15:0>	PTXFSPCAVAIL	RO	0x0	0x0	Periodic transmit-data FIFO space available. Indicates the number of free locations available to be written to in the periodic TxFIFO. Values are in terms of 32-bit words. 0x0 = Periodic TxFIFO is full 0x1 = 1 word available 0x2 = 2 words available 0xN = N words available (0..0x8000) All others = reserved

Host All Channels Interrupt Register USBC_HAINT

When a significant event occurs on a channel, this register interrupts the application using USBC_ GINTSTS[HCHINT]. There is one interrupt bit per channel, up to a maximum of 16 bits. Bits in this register are set and cleared when the application sets and clears bits in the corresponding USBC_HCINT*n* register. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	—	RO	—	—	Reserved.
<15:0>	HAINT	RO	0x0	0x0	Channel interrupts. One bit per channel: bit<0> for channel 0, ..., bit<15> for channel 15

Host All Channels Interrupt Mask Register USBC_HAINTMSK

This register works with USBC_HAINT to interrupt the application when an event occurs on a channel. There is one interrupt mask bit per channel, up to a maximum of 16 bits. 0 = mask the interrupt, 1 = unmask the interrupt. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	—	RO	—	—	Reserved.
<15:0>	HAINTMSK	R/W	0x0	0x0	Channel interrupt masks. One bit per channel: bit<0> for channel 0, ..., bit<15> for channel 15

Host Port Control and Status Register USBC_HPRT

This register is available in both host and device modes. Currently, the OTG host supports only one port. A single register holds USB port-related information such as USB reset, enable, suspend, resume, connect status, and test mode for each port. The R/W1C bits in this register can trigger an interrupt to the application through the USBC_GINTSTS[PRTINT] bit. On a port interrupt, the application must read this register and clear the bit that caused the interrupt. For the R/W1C bits, the application must write a 1 to the bit to clear the interrupt. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:19>	—	RO	—	—	Reserved.
<18:17>	PRTSPD	RO	0x0	0x0	Port speed. Indicates the speed of the device attached to this port. 00 = high speed, 01 = full speed, 10 = low speed, 11 = reserved
<16:13>	PRTTSTCTL	R/W	0x0	0x0	Port test control. The application writes a nonzero value to this field to put the port into a test mode, and the corresponding pattern is signaled on the port. 0x0 = test mode disabled 0x1 = Test_J mode 0x2 = Test_K mode 0x3 = Test_SE0_NAK mode 0x4 = Test_Packet mode 0x5 = Test_Force_Enable All others = reserved To use the PRTTSTCTL test modes, PRTSPD must be zero (i.e. the interface must be in high-speed mode).
<12>	PRTPWR	R/W	0x0	0x0	Port power. The application uses this field to control power to this port, and the USB core clears this bit on an overcurrent condition. 0 = power off 1 = power on
<11:10>	PRTLNSTS	RO	0x0	0x0	Port line status. Indicates the current logic level of USB data lines. Bit [10]: logic level of D– Bit [11]: logic level of D+
<9>	—	RO	—	—	Reserved.
<8>	PRTRST	R/W	0	0	Port reset. When the application sets this bit, a reset sequence is started on this port. The application must time the reset period and clear this bit after the reset sequence is complete. 0 = port not in reset 1 = port in reset The application must leave this bit set for at least the minimum duration mentioned below to start a reset on the port. The application can leave it set for another 10 ms in addition to the required minimum duration, before clearing the bit, even though there is no maximum limit set by the USB standard. • high speed: 50 ms • full speed/low speed: 10 ms

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<7>	PRTSUSP	R/W	0	0	Port suspend. The application sets this bit to put this port in suspend mode. The USB core only stops sending SOFs when this is set. To stop the PHY clock, the application must set the Port Clock Stop bit, which will assert the suspend input pin of the PHY. The read value of this bit reflects the current suspend status of the port. This bit is cleared by the core after a remote wakeup signal is detected or the application sets PRTRST or PRTRES bit in this register or the USBC_GINTSTS[WKUPINT, DISCONNINT] bits, respectively). 0 = port is not in suspend mode 1 = port is in suspend mode
<6>	PRTRES	R/W	0	0	Port resume. The application sets this bit to drive resume signaling on the port. The USB core continues to drive the resume signal until the application clears this bit. If the core detects a USB remote-wakeup sequence, as indicated by USBC_GINTSTS[WKUPINT], the core starts driving resume signaling without application intervention and clears this bit when it detects a disconnect condition. The read value of this bit indicates whether the core is currently driving resume signaling. 0 = resume is not driven 1 = resume is driven
<5>	PRTOVRCURRCHNG	R/W1C	0	0	Port overcurrent change. The USB core sets this bit when the status of PRTOVRCURACT changes.
<4>	PRTOVRCURRACT	RO	0	0	Port overcurrent active. Indicates the overcurrent condition of the port. 0 = no overcurrent condition 1 = overcurrent condition
<3>	PRTENCHNG	R/W1C	0	0	Port enable/disable change. The USB core sets this bit when the status of PRTENA changes.
<2>	PRTENA	R/W1C	0	0	Port enable. A port is enabled only by the core after a reset sequence, and is disabled by an overcurrent condition, a disconnect condition, or by the application clearing this bit. The application cannot set this bit by a register write. It can only clear it to disable the port. This bit does not trigger any interrupt to the application. 0 = port disabled 1 = port enabled
<1>	PRTCONNDET	R/W1C	0	0	Port connect detected. The USB core sets this bit when a device connection is detected to trigger an interrupt to the application using USBC_GINTSTS[PRTINT].
<0>	PRTCONNSTS	RO	0	0	Port connect status. 0 = no device is attached to the port. 1 = a device is attached to the port.

Host Channel Characteristics Registers USBC_HCCHAR(0..7)

This register contains characteristic information for the corresponding channel. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	CHENA	R/W	0	0	Channel enable. This field is set by the application and cleared by the OTG host: 0 = channel is disabled, 1 = channel is enabled.
<30>	CHDIS	R/W	0	0	Channel disable. The application sets this bit to stop transmitting/ receiving data on a channel, even before the transfer for that channel is complete. The application must wait for the channel-disabled interrupt before treating the channel as disabled.
<29>	ODDFRM	R/W	0	0	Odd frame. This field is set or reset by the application to indicate that the OTG host must perform a transfer in an odd (micro)frame. This field is applicable for only periodic (isochronous and interrupt) transactions. 0 = even (micro)frame 1 = odd (micro)frame
<28:22>	DEVADDR	R/W	0x0	0x0	Device address. This field selects the specific device serving as the data source or sink.
<21:20>	EC	R/W	0x0	0x0	Multicount/error count. When USBC_HCSPLT*n*[SPLTENA] is reset to 0, this field indicates to the host the number of transactions that should be executed per microframe for this endpoint. 00 = reserved. This field yields undefined results. 01 = 1 transaction 10 = 2 transactions to be issued for this endpoint per microframe 11 = 3 transactions to be issued for this endpoint per microframe When USBC_HCSPLT*n*[SPLTENA] is set to 1, this field indicates the number of immediate retries to be performed for a periodic split transactions on transaction errors. This field must be set to at least 0x1.
<19:18>	EPTYPE	R/W	0x0	0x0	Endpoint type. Indicates the transfer type selected. 00 = control 01 = isochronous 10 = bulk 11 = interrupt
<17>	LSPDDEV	R/W	0	0	Low-speed device. This field is set by the application to indicate that this channel is communicating to a low-speed device.
<16>	—	RO	—	—	Reserved.
<15>	EPDIR	R/W	0	0	Endpoint direction. Indicates whether the transaction is IN or OUT. 0 = OUT, 1 = IN
<14:11>	EPNUM	R/W	0x0	0x0	Endpoint number. Indicates the endpoint number on the device serving as the data source or sink.
<10:0>	MPS	R/W	0x0	0x0	Maximum packet size. Indicates the maximum packet size of the associated endpoint.

Host Channel Split Control Registers
USBC_HCSPLT(0..7)

This register contains split information for the corresponding channel. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	SPLTENA	R/W	0	0	Split enable. The application sets this field to indicate that this channel is enabled to perform split transactions.
<30:17>	—	RO	—	—	Reserved.
<16>	COMPSPLT	R/W	0	0	Perform complete split. The application sets this field to request the OTG host to perform a complete split transaction.
<15:14>	XACTPOS	R/W	0x0	0x0	Transaction position. This field is used to determine whether to send all, first, middle, or last payloads with each OUT transaction. 11 = all This is the entire data payload of this transaction (which is less than or equal to 188 bytes). 10 = begin This is the first data payload of this transaction (which is larger than 188 bytes). 00 = middle This is the middle payload of this transaction (which is larger than 188 bytes). 01 = end This is the last payload of this transaction (which is larger than 188 bytes).
<13:7>	HUBADDR	R/W	0x0	0x0	Hub address. This field holds the device address of the transaction translator's hub.
<6:0>	PRTADDR	R/W	0x0	0x0	Port address. This field is the port number of the recipient transaction translator.

Host Channel Interrupt Registers
USBC_HCINT(0..7)

This register indicates the status of a channel with respect to USB- and AHB-related events. The application must read this register when USBC_GINTSTS[HCHINT] is set. Before the application can read this register, it must first read USBC_HAINT to get the exact channel number for USBC_HINT*n*. The application must clear the appropriate bit in this register to clear the corresponding bits in the USBC_HAINT and USBC_GINTSTS registers. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:11>	—	RO	—	—	Reserved.
<10>	DATATGLERR	R/W1C	0	0	Data toggle error.
<9>	FRMOVRUN	R/W1C	0	0	Frame overrun.
<8>	BBLERR	R/W1C	0	0	Babble error.
<7>	XACTERR	R/W1C	0	0	Transaction error.
<6>	NYET	R/W1C	0	0	NYET response-received interrupt.
<5>	ACK	R/W1C	0	0	ACK response-received interrupt.
<4>	NAK	R/W1C	0	0	NAK response-received interrupt.
<3>	STALL	R/W1C	0	0	STALL response-received interrupt.
<2>	AHBERR	R/W1C	0	0	This bit is always 0.
<1>	CHHLTD	R/W1C	0	0	Channel halted. Indicates the transfer completed abnormally either because of any USB transaction error or in response to disable request by the application.
<0>	XFERCOMPL	R/W1C	0	0	Transfer completed. The transfer completed normally without any errors.

Host Channel Interrupt Mask Registers USBC_HCINTMSK(0..7)

This register reflects the mask for each channel status described in USBC_HCINT(0..7): 0 = mask the interrupt, 1 = unmask the interrupt. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:11>	—	RO	—	—	Reserved.
<10>	DATATGLERRMSK	R/W	0	0	Data-toggle-error mask.
<9>	FRMOVRUNMSK	R/W	0	0	Frame-overrun mask.
<8>	BBLERRMSK	R/W	0	0	Babble-error mask.
<7>	XACTERRMSK	R/W	0	0	Transaction-error mask.
<6>	NYETMSK	R/W	0	0	NYET response-received interrupt mask.
<5>	ACKMSK	R/W	0	0	ACK response-received interrupt mask.
<4>	NAKMSK	R/W	0	0	NAK response-received interrupt mask.
<3>	STALLMSK	R/W	0	0	STALL response-received interrupt mask.
<2>	AHBERRMSK	R/W	0	0	AHB-error mask.
<1>	CHHLTDMSK	R/W	0	0	Channel-halted mask.
<0>	XFERCOMPLMSK	R/W	0	0	Transfer-completed mask.

Host Channel Transfer Size Registers USBC_HCTSIZ(0..7)

This register contains the transfer size and packet count, as well as the PING enable bit. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	DOPNG	R/W	0	0	Do PING. Setting this field to 1 directs the host to do PING protocol.
<30:29>	PID	R/W	0x0	0x0	PID. The application programs this field with the type of PID to use for the initial transaction. The host maintains this field for the rest of the transfer. 00 = data0 01 = data2 10 = data1 11 = Mdata (non-control)/SETUP (control)
<28:19>	PKTCNT	R/W	0x0	0x0	Packet count. This field is programmed by the application with the expected number of packets to be transmitted (OUT) or received (IN). The host decrements this count on every successful transmission or reception of an OUT/IN packet. Once this count reaches zero, the application is interrupted to indicate normal completion.
<18:0>	XFERSIZE	R/W	0x0	0x0	Transfer size. For an OUT, this field is the number of data bytes the host sends during the transfer For an IN, this field is the buffer size that the application has reserved for the transfer. The application is expected to program this field as an integer multiple of the maximum packet size for IN transactions (periodic and nonperiodic).

Device Configuration Register
USBC_DCFG

This register configures the USB core in device mode after power-on or after certain control commands or enumeration. Do not make changes to this register after initial programming. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:23>	—	RO	—	—	Reserved.
<22:18>	EPMISCNT	R/W	0x8	0x0	IN endpoint-mismatch count. The application programs this field with a count that determines when the core generates an Endpoint Mismatch interrupt (USBC_GINTSTS[EPMIS]). The core loads this value into an internal counter and decrements it. The counter is reloaded whenever there is a match or when the counter expires. The width of this counter depends on the depth of the token queue.
<17:13>	—	RO	—	—	Reserved.
<12:11>	PERFRINT	R/W	0x0	0x0	Periodic frame interval. Indicates the time within a (micro)frame at which the application must be notified using the end-of-periodic-frame interrupt. This can be used to determine if all the isochronous traffic for that (micro)frame is complete. 00 = 80% of the (micro)frame interval 01 = 85% 10 = 90% 11 = 95%
<10:4>	DEVADDR	R/W	0x0	0x0	Device address. The application must program this field after every SetAddress control command.
<3>	—	RO	—	—	Reserved.
<2>	NZSTSOUTHSHK	R/W	0x0	0x0	Nonzero-length status OUT handshake. The application can use this field to select the handshake the core sends on receiving a nonzero-length data packet during the OUT transaction of a control transfer's status stage. 1 = Send a STALL handshake on a nonzero-length status OUT transaction and do not send the received OUT packet to the application. 0 = Send the received OUT packet to the application (zero-length or nonzero-length) and send a handshake based on the NAK and STALL bits for the endpoint in the device-endpoint control register.
<1:0>	DEVSPD	R/W	0x0	0x0	Device speed. Indicates the speed at which the application requires the USB core to enumerate, or the maximum speed the application can support. However, the actual bus speed is determined only after the chirp sequence is completed, and is based on the speed of the USB host to which the core is connected. 00 = high speed (USB 2.0 PHY clock is 30 MHz or 60 MHz) 01 = full speed (USB 2.0 PHY clock is 30 MHz or 60 MHz) 10 = low speed (USB 1.1 transceiver clock is 6 MHz). If you select 6-MHz LS mode, you must do a soft reset. 11 = full speed (USB 1.1 transceiver clock is 48 MHz)

Device Control Register
USBC_DCTL

This register contains the control bits for the operation of the USB core. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:12>	—	RO	—	—	Reserved.
<11>	PWRONPRGDONE	R/W	0	0	Power-on programming done. The application uses this bit to indicate that register programming is completed after a wake-up from power-down mode.
<10>	CGOUTNAK	WO	0	0	Clear global OUT NAK. A write to this field clears the global OUT NAK.
<9>	SGOUTNAK	WO	0	0	Set global OUT NAK. A write to this field sets the global OUT NAK. The application uses this bit to send a NAK handshake on all OUT endpoints. The application should set the this bit only after making sure that USBC_GINTSTS[GOUTNAKEFF] is cleared.
<8>	CGNPINNAK	WO	0	0	Clear global nonperiodic IN NAK. A write to this field clears the global nonperiodic IN NAK.
<7>	SGNPINNAK	WO	0	0	Set global nonperiodic IN NAK. A write to this field sets the global nonperiodic IN NAK. The application uses this bit to send a NAK handshake on all nonperiodic IN endpoints. The USB core can also set this bit when a timeout condition is detected on a nonperiodic endpoint. The application should set this bit only after making sure that USBC_GINTSTS[GINNAKEFF] is cleared.
<6:4>	TSTCTL	R/W	0x0	0x0	Test control. 000 = test mode disabled 001 = test_J mode 010 = test_K mode 011 = test_SE0_NAK mode 100 = test_Packet mode 101 = test_Force_Enable All others = reserved
<3>	GOUTNAKSTS	RO	0	0	Global OUT NAK status. 0 = A handshake is sent based on the FIFO Status and the NAK and STALL bit settings. 1 = No data is written to the RxFIFO, irrespective of space availability. Sends a NAK handshake on all packets, except on SETUP transactions. All isochronous OUT packets are dropped.
<2>	GNPINNAKSTS	RO	0	0	Global nonperiodic IN NAK status. 0 = A handshake is sent based on the data availability in the transmit FIFO. 1 = A NAK handshake is sent out on all non-periodic IN endpoints, irrespective of the data availability in the transmit FIFO.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<1>	SFTDISCON	R/W	0	0	Soft disconnect. The application uses this bit to signal the CN50XX USB core to do a soft disconnect. As long as this bit is set, the host will not see that the device is connected, and the device will not receive signals on the USB. The core stays in the disconnected state until the application clears this bit. The minimum duration for which the core must keep this bit set is specified in Minimum Duration for Soft Disconnect. 0 = Normal operation. When this bit is cleared after a soft disconnect, the core drives the phy_opmode_o signal on the UTMI+ to 0x0, which generates a device connect event to the USB host. When the device is reconnected, the USB host restarts device enumeration. 1 = The USB core drives the phy_opmode_o signal on the UTMI+ to 0x1, which generates a device disconnect event to the USB host.
<0>	RMTWKUPSIG	R/W	0	0	Remote wakeup signaling. When the application sets this bit, the USB core initiates remote signaling to wake up the USB host.The application must set this bit to get the USB core out of suspended state and must clear this bit after the USB core comes out of suspended state.

Device Status Register USBC_DSTS

This register indicates the status of the USB core with respect to USB-related events. It must be read on interrupts from USBC_DAINT. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:22>	—	RO	—	—	Reserved.
<21:8>	SOFFN	RO	0x0	0x0	Frame or microframe number of the received SOF. When the USB core is operating at high speed, this field contains a microframe number. When the USB core is operating at full or low speed, this field contains a frame number.
<7:4>	—	RO	—	—	Reserved.
<3>	ERRTICERR	RO	0x0	0x0	Erratic error. The USB core sets this bit to report any erratic errors (phy_rxvalid_i/phy_rxvldh_i or phy_rxactive_i is asserted for at least 2 ms, due to PHY error) seen on the UTMI+. Due to erratic errors, the CN50XX USB core goes into suspended state and an interrupt is generated to USBC_GINTSTS[ERLYSUSP]. If the early suspend is asserted due to an erratic error, the application can only perform a soft disconnect recover.
<2:1>	ENUMSPD	RO	0x0	0x0	Enumerated speed. Indicates the speed at which the CN50XX USB core has come up after speed detection through a chirp sequence. 00 = high speed (PHY clock is running at 30 or 60 MHz) 01 = full speed (PHY clock is running at 30 or 60 MHz) 10 = low speed (PHY clock is running at 6 MHz) 11 = full speed (PHY clock is running at 48 MHz) Low speed is not supported for devices using a UTMI+ PHY.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<0>	SUSPSTS	RO	0x0	0x0	Suspend status. In device mode, this bit is set as long as a suspend condition is detected on the USB. The USB core enters the suspended state when there is no activity on the phy_line_state_i signal for an extended period of time. The USB core comes out of the suspend: • When there is any activity on the phy_line_state_i signal • When the application writes to USBC_DCTL[RMTWKUPSIG].

Device IN Endpoint Common Interrupt Mask Register USBC_DIEPMSK

This register works with each of the USBC_DIEPINT*n* registers for all endpoints to generate an interrupt per IN endpoint. The IN endpoint interrupt for a specific status in the USBC_DIEPINT*n* register can be masked by writing to the corresponding bit in this register. Status bits are masked by default: 0 = mask the interrupt, 1 = unmask the interrupt. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:7>	—	RO	—	—	Reserved.
<6>	INEPNAKEFFMSK	R/W	0	0	IN endpoint NAK effective mask.
<5>	INTKNEPMISMSK	R/W	0	0	IN token received with EP mismatch mask.
<4>	INTKNTXFEMPMSK	R/W	0	0	IN token received when TxFIFO empty mask.
<3>	TIMEOUTMSK	R/W	0	0	Timeout condition mask (non-isochronous endpoints).
<2>	AHBERRMSK	R/W	0	0	AHB error mask.
<1>	EPDISBLDMSK	R/W	0	0	Endpoint disabled interrupt mask.
<0>	XFERCOMPLMSK	R/W	0	0	Transfer completed interrupt mask.

Device OUT Endpoint Common Interrupt Mask Register USBC_DOEPMSK

This register contains the interrupt summary bits of the USBN. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:5>	—	RO	—	—	Reserved.
<4>	OUTTKNEPDISMSK	R/W	0	0	OUT token received when endpoint disabled mask. Applies to control OUT endpoints only.
<3>	SETUPMSK	R/W	0	0	SETUP phase done mask. Applies to control endpoints only.
<2>	AHBERRMSK	R/W	0	0	AHB error.
<1>	EPDISBLDMSK	R/W	0	0	Endpoint disabled interrupt mask.
<0>	XFERCOMPLMSK	R/W	0	0	Transfer completed interrupt mask.

Device All Endpoints Interrupt Register
USBC_DAINT

When a significant event occurs on an endpoint, this register interrupts the application using USBC_GINTSTS[OEPINT or IEPINT] respectively. There is one interrupt bit per endpoint, up to a maximum of 16 bits for OUT endpoints and 16 bits for IN endpoints. For a bidirectional endpoint, the corresponding IN and OUT interrupt bits are used. Bits in this register are set and cleared when the application sets and clears bits in the corresponding USBC_DIEPINT*n*/USBC_DOEPINT*n* register. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	OUTEPINT	RO	0x0	0x0	OUT endpoint interrupt bits. One bit per OUT endpoint. Bit 16 is for OUT endpoint 0; bit 31 is for OUT endpoint 15.
<15:0>	INEPINT	RO	0x0	0x0	IN endpoint interrupt bits. One bit per IN endpoint. Bit 0 is for IN endpoint 0; bit 15 is for IN endpoint 15.

Device All Endpoints Interrupt Mask Register
USBC_DAINTMSK

This register contains the. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:16>	OUTEPMSK	R/W	0x0	0x0	OUT EP interrupt mask bits. One per OUT endpoint. Bit 16 is for OUT EP 0; bit 31 is for OUT EP 15.
<15:0>	INEPMSK	R/W	0x0	0x0	IN EP interrupt mask bits. One bit per IN endpoint. Bit 0 is for IN EP 0; bit 15 is for IN EP 15.

Device IN Token Sequence Learning Queue Read Register 1
USBC_DTKNQR1

The depth of the IN token-sequence learning queue is specified for device-mode IN token-sequence learning queue depth. The queue is four bits wide to store the endpoint number. A read from this register returns the first five endpoint entries of the IN token-sequence learning queue. When the queue is full, the new token is pushed into the queue and oldest token is discarded. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:8>	EPTKN	RO	0x0	0x0	Endpoint token. Four bits per token represent the endpoint number of the token: Bits [31:28]: endpoint number of token 5 Bits [27:24]: endpoint number of token 4 ... Bits [15:12]: endpoint number of token 1 Bits [11:8]: endpoint number of token 0
<7>	WRAPBIT	RO	0x0	0x0	Wrap bit. This bit is set when the write pointer wraps. It is cleared when the learning queue is cleared.
<6:5>	—	RO	—	—	Reserved.
<4:0>	INTKNWPTR	RO	0x0	0x0	IN token queue write pointer.

Device IN Token Sequence Learning Queue Read Registers USBC_DTKNQR2/3/4

A read from one of these registers returns the next eight endpoint entries of the learning queue.

- USBC_DTKNQR2 handles tokens 6–13.
- USBC_DTKNQR3 handles tokens 14–21.
- USBC_DTKNQR4 handles tokens 22–29.

See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	EPTKN	RO	0x0	0x0	Endpoint token. Four bits per token represent the endpoint number of the token: Bits [31:28]: endpoint number of token 13/21/29 Bits [27:24]: endpoint number of token 12/20/28 ... Bits [7:4]: endpoint number of token 7/15/23 Bits [3:0]: endpoint number of token 6/14/22

Device Control IN Endpoint 0 Control Register USBC_DIEPCTL0

The application uses this register to control the behavior of logical IN endpoint 0. Note that nonzero control IN endpoints use USBC_DIEPCTL(1..4). See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	EPENA	R/W	0	0	Endpoint enable. Indicates that data is ready to be transmitted on the endpoint. The USB core clears this bit before setting any of the following interrupts on this endpoint: •endpoint disabled •transfer completed
<30>	EPDIS	R/W	0	0	Endpoint disable. The application sets this bit to stop transmitting data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the endpoint-disabled interrupt before treating the endpoint as disabled. The USB core clears this bit before setting the endpoint-disabled interrupt. The application should set this bit only if EPENA is already set for this endpoint.
<29:28>	—	RO	—	—	Reserved.
<27>	SNAK	WO	0	0	Set NAK. A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The USB core can also set this bit for an endpoint after a SETUP packet is received on that endpoint.
<26>	CNAK	WO	0	0	Clear NAK. A write to this bit clears the NAK bit for the endpoint.
<25:22>	TXFNUM	RO	0x0	0x0	TxFIFO number. This value is always set to 0, indicating that control IN endpoint 0 data is always written in the nonperiodic transmit FIFO.
<21>	STALL	R/W	0	0	STALL handshake. The application can only set this bit, and the USB core clears it, when a SETUP token is received for this endpoint. If a NAK bit, global nonperiodic IN NAK, or global OUT NAK is set along with this bit, the STALL bit takes priority.
<20>	—	RO	—	—	Reserved.
<19:18>	EPTYPE	RO	0x0	0x0	Endpoint type. Hardcoded to 00 for control.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<17>	NAKSTS	RO	0	0	NAK status. Indicates the following: 0 = The USB core is transmitting non-NAK handshakes based on the FIFO status 1 = The USB core is transmitting NAK handshakes on this endpoint. When this bit is set, either by the application or core, the USB core stops transmitting data, even if there is data available in the TxFIFO. Regardless of this bit's setting, the USB core always responds to SETUP data packets with an ACK handshake.
<16>	—	RO	—	—	Reserved.
<15>	USBACTEP	RO	1	0	USB active endpoint. This bit is always set to 1, indicating that control endpoint 0 is always active in all configurations and interfaces.
<14:11>	NEXTEP	R/W	0x0	0x0	Next endpoint. Applies to nonperiodic IN endpoints only. Indicates the endpoint number to be fetched after the data for the current endpoint is fetched. The core can access this field, even when EPENA is not set. This field is not valid in slave mode.
<10:2>	—	RO	—	—	Reserved.
<1:0>	MPS	R/W	0x0	0x0	Maximum packet size. Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical endpoint. 00 = 64 bytes, 01 = 32 bytes, 10 = 16 bytes, 11 = 8 bytes

Device IN Endpoint Control Registers USBC_DIEPCTL(1..4)

The application uses these registers to control the behavior of each logical nonzero IN endpoint. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	EPENA	R/W	0	0	Endpoint enable. Indicates that data is ready to be transmitted on the endpoint. The USB core clears this bit before setting any of the following interrupts on this endpoint: ●endpoint disabled ●transfer completed
<30>	EPDIS	R/W	0	0	Endpoint disable. The application sets this bit to stop transmitting data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the endpoint-disabled interrupt before treating the endpoint as disabled. The USB core clears this bit before setting the endpoint-disabled interrupt. The application should set this bit only if EPENA is already set for this endpoint.
<29>	SETD1PID	WO	0	0	For Interrupt/BULK endpoints: set data1 PID. Writing to this field sets DPID to data1. For Isochronous endpoints: set odd (micro)frame (SETODDFR). Writing to this field sets EO_FRNUM to odd (micro)frame.
<28>	SETD0PID	WO	0	0	For Interrupt/BULK endpoints: set data0 PID. Writing to this field sets DPID to data0. For Isochronous endpoints: Set even (micro)frame (SETEVENFR). Writing to this field sets EO_FRNUM to even (micro)frame.
<27>	SNAK	WO	0	0	Set NAK. A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The USB core can also set this bit for an endpoint after a SETUP packet is received on the endpoint.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<26>	CNAK	WO	0	0	Clear NAK. A write to this bit clears the NAK bit for the endpoint.
<25:22>	TXFNUM	R/W	0x0	0x0	TxFIFO number. Nonperiodic endpoints must set this bit to 0x0. Periodic endpoints must map this to the corresponding periodic TxFIFO number. 0x0 = nonperiodic TxFIFO All others = specified periodic TxFIFO number
<21>	STALL	R/W	0	0	STALL handshake. For noncontrol, nonisochronous endpoints: The application sets this bit to stall all tokens from the USB host to this endpoint. If a NAK bit, global nonperiodic IN NAK, or global OUT NAK is set along with this bit, the STALL bit takes priority. Only the application can clear this bit, never the USB core. For control endpoints: The application can only set this bit, and the USB core clears it, when a SETUP token is received for this endpoint. If a NAK bit, global nonperiodic IN NAK, or global OUT NAK is set along with this bit, the STALL bit takes priority. Regardless of this bit's setting, the USB core always responds to SETUP data packets with an ACK handshake.
<20>	—	RO	—	—	Reserved.
<19:18>	EPTYPE	R/W	0x0	0x0	Endpoint type. This is the transfer type supported by this logical endpoint. 00 = control 01 = isochronous 10 = bulk 11 = interrupt
<17>	NAKSTS	RO	0	0	NAK status. Indicates the following: 0 = USB core is transmitting non-NAK handshakes based on the FIFO status 1 = USB core is transmitting NAK handshakes on this endpoint. When either the application or the USB core sets this bit: ●For nonisochronous IN endpoints: the USB core stops transmitting any data on an IN endpoint, even if data is available in the TxFIFO. ●For isochronous IN endpoints: the USB core sends out a zero-length data packet, even if data is available in the TxFIFO. Regardless of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.
<16>	DPID	RO	0	0	For interrupt/bulk IN and OUT endpoints: Endpoint data PID (DPID) Contains the PID of the packet to be received or transmitted on this endpoint. The application should program the PID of the first packet to be received or transmitted on this endpoint, after the endpoint is activated. Applications use the SETD1PID and SETD0PID fields of this register to program either data0 or data1 PID. 0 = data0 1 = data1 For isochronous IN and OUT endpoints: even/odd (micro)frame (EO_FRNUM) Indicates the (micro)frame number in which the core transmits/receives isochronous data for this endpoint. The application should program the even/odd (micro) frame number in which it intends to transmit/receive isochronous data for this endpoint using the SETEVNFR and SETODDFR fields in this register. 0 = even (micro)frame 1 = odd (micro)frame

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<15>	USBACTEP	R/W	1	0	USB active endpoint. Indicates whether this endpoint is active in the current configuration and interface. The USB core clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit.
<14:11>	NEXTEP	R/W	0	0	Next endpoint. Applies to nonperiodic IN endpoints only. Indicates the endpoint number to be fetched after the data for the current endpoint is fetched. The USB core can access this field, even when EPENA is not set. This field is not valid in slave mode.
<10:0>	MPS	R/W	0x0	0x0	Maximum packet size. Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.

Device IN Endpoint Interrupt Registers
USBC_DIEPINT(0..4)

This register indicates the status of an endpoint with respect to USB- and AHB-related events. The application must read this register when the USBC_GINTSTS[OEPINT or IEPINT] bits are set. Before the application can read this register, it must first read the USBC_DAINT register to get the exact endpoint number for USBC_DIEPINT(0..4). The application must clear the appropriate bit in this register to clear the corresponding bits in the USBC_DAINT and USBC_GINTSTS registers. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:7>	—	RO	—	—	Reserved.
<6>	INEPNAKEFF	RO	0	0	IN endpoint NAK effective. Applies to periodic IN endpoints only. Indicates that the IN endpoint NAK bit set by the application has taken effect in the USB core. This bit can be cleared when the application clears the IN endpoint NAK by writing to USBC_DIEPCTL*n*[CNAK]. This interrupt indicates that the USB core has sampled the NAK bit set (either by the application or by the USB core). This interrupt does not necessarily mean that a NAK handshake is sent on the USB. A STALL bit takes priority over a NAK bit.
<5>	INTKNEPMIS	R/W	0	0	IN token received with EP mismatch. Applies to nonperiodic IN endpoints only. Indicates that the data in the top of the nonperiodic TxFIFO belongs to an endpoint other than the one for which the IN token was received. This interrupt is asserted on the endpoint for which the IN token was received.
<4>	INTKNTXFEMP	R/W1C	0	0	IN token received when TxFIFO is empty. Applies only to non-periodic IN endpoints. Indicates that an IN token was received when the associated TxFIFO (periodic/nonperiodic) was empty. This interrupt is asserted on the endpoint for which the IN token was received.
<3>	TIMEOUT	R/W1C	0	0	Timeout condition. Applies to nonisochronous IN endpoints only. Indicates that the core has detected a timeout condition on the USB for the last IN token on this endpoint.
<2>	AHBERR	R/W1C	0	0	AHB error. This is generated only in internal-DMA mode when there is an AHB error during an AHB read/write operation. The application can read the corresponding endpoint DMA address register to get the error address.
<1>	EPDISBLD	R/W1C	0	0	Endpoint disabled interrupt. This bit indicates that the endpoint is disabled per the application's request.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<0>	XFERCOMPL	R/W1C	0	0	Transfer completed interrupt. Indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.

Device IN Endpoint 0 Transfer-Size Register USBC_DIEPTSIZ0

This register contains the. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:20>	—	RO	—	—	Reserved.
<19>	PKTCNT	R/W	0	0	Packet count. Indicates the total number of USB packets that constitute the transfer-size amount of data for endpoint 0. This field is decremented every time a packet (maximum-size or short packet) is read from the TxFIFO.
<18:7>	—	RO	—	—	Reserved.
<6:0>	XFERSIZE	R/W	0x0	0x0	Transfer size. Indicates the transfer size in bytes for endpoint 0. The USB core interrupts the application only after it has exhausted the transfer-size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. The USB core decrements this field every time a packet from the external memory is written to the TxFIFO.

Device IN Endpoint Transfer-Size Registers USBC_DIEPTSIZ(1..4)

The application must modify this register before enabling the endpoint. Once endpoint 0 is enabled (using either USBC_DIEPCTL0[EPENA] or USBC_DOEPCTL0[EPENA]), the USB core modifies this register. The application can only read this register once the USB core has cleared the EPENA bit. This register is used only for endpoints other than endpoint 0. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	—	RO	—	—	Reserved.
<30:29>	MC	R/W	0x0	0x0	Multicount. Applies to IN endpoints only. For periodic IN endpoints, this field indicates the number of packets that must be transmitted per microframe on the USB. The USB core uses this field to calculate the data PID for isochronous IN endpoints. 01 = 1 packet 10 = 2 packets 11 = 3 packets For non-periodic IN endpoints, this field is valid only in internal-DMA mode. It specifies the number of packets the USB core should fetch for an IN endpoint before it switches to the endpoint pointed to by USBC_DIEPCTL*n*[NEXTEP].
<28:19>	PKTCNT	R/W	0x0	0x0	Packet count. Indicates the total number of USB packets that constitute the transfer-size amount of data for this endpoint. IN endpoints: This field is decremented every time a packet (maximum-size or short packet) is read from the TxFIFO.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<18:0>	XFERSIZE	R/W	0x0	0x0	Transfer size. This field contains the transfer size in bytes for the current endpoint. The USB core only interrupts the application after it has exhausted the transfer-size amount of data. The transfer size can be set to the maximum-packet size of the endpoint, to be interrupted at the end of each packet. IN endpoints: The USB core decrements this field every time a packet from the external memory is written to the TxFIFO.

Device Control OUT Endpoint 0 Control Register USBC_DOEPCTL0

The application uses this register to control the behavior of logical IN endpoint 0. Note that nonzero control IN endpoints use USBC_DIEPCTL(1..4). See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	EPENA	R/W	0	0	Endpoint enable. Indicates that the application has allocated the memory to start receiving data from the USB. The USB core clears this bit before setting any of the following interrupts on this endpoint: ●SETUP phase done ●endpoint disabled ●transfer completed
<30>	EPDIS	R/W	0	0	Endpoint disable. The application cannot disable control OUT endpoint 0.
<29:28>	—	RO	—	—	Reserved.
<27>	SNAK	WO	0	0	Set NAK. A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The USB core can also set bit on a transfer-completed interrupt, or after a SETUP is received on the endpoint.
<26>	CNAK	WO	0	0	Clear NAK. A write to this bit clears the NAK bit for the endpoint.
<25:22>	—	RO	—	—	Reserved.
<21>	STALL	R/W	0	0	STALL handshake. The application can only set this bit, and the USB core clears it, when a SETUP token is received for this endpoint. If a NAK bit or global OUT NAK is set along with this bit, the STALL bit takes priority. Regardless of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.
<20>	SNP	R/W	0	0	Snoop mode. This bit configures the endpoint to snoop mode. In snoop mode, the USB core does not check the correctness of OUT packets before transferring them to application memory.
<19:18>	EPTYPE	RO	0x0	0x0	Endpoint type. Hardcoded to 00 for control.
<17>	NAKSTS	RO	0	0	NAK status. Indicates the following: 0 = The USB core is transmitting non-NAK handshakes based on the FIFO status 1 = The USB core is transmitting NAK handshakes on this endpoint. When either the application or the core sets this bit, the USB core stops receiving data, even if there is space in the RxFIFO to accommodate the incoming packet. Regardless of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.
<16>	—	RO	—	—	Reserved.
<15>	USBACTEP	RO	1	0	USB active endpoint. This bit is always set to 1, indicating that control endpoint 0 is always active in all configurations and interfaces.
<14:2>	—	RO	—	—	Reserved.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<1:0>	MPS	R/W	0x0	0x0	Maximum packet size. The maximum packet size for control OUT endpoint 0 is the same as what is programmed in control IN endpoint 0. 00 = 64 bytes, 01 = 32 bytes, 10 = 16 bytes, 11 = 8 bytes

Device OUT Endpoint Control Registers USBC_DOEPCTL(1..4)

The application uses these registers to control the behavior of each logical nonzero IN endpoint. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	EPENA	R/W	0	0	Endpoint enable. Indicates that data is ready to be transmitted on the endpoint. The USB core clears this bit before setting any of the following interrupts on this endpoint: ●SETUP phase done ●endpoint disabled ●transfer completed For control OUT endpoints in DMA mode, this bit must be set to be able to transfer SETUP data packets in memory.
<30>	EPDIS	R/W	0	0	Endpoint disable. The application sets this bit to stop transmitting data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the endpoint-disabled interrupt before treating the endpoint as disabled. The USB core clears this bit before setting the endpoint-disabled interrupt. The application should set this bit only if EPENA is already set for this endpoint.
<29>	SETD1PID	WO	0	0	For Interrupt/BULK endpoints: set data1 PID. Writing to this field sets DPID to data1. For Isochronous endpoints: set odd (micro)frame (SETODDFR). Writing to this field sets EO_FRNUM to odd (micro)frame.
<28>	SETD0PID	WO	0	0	For Interrupt/BULK endpoints: set data0 PID. Writing to this field sets DPID to data0. For Isochronous endpoints: Set even (micro)frame (SETEVENFR). Writing to this field sets EO_FRNUM to even (micro)frame.
<27>	SNAK	WO	0	0	Set NAK. A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The USB core can also set this bit for an endpoint after a SETUP packet is received on the endpoint.
<26>	CNAK	WO	0	0	Clear NAK. A write to this bit clears the NAK bit for the endpoint.
<25:22>	—	RO	—	—	Reserved.
<21>	STALL	R/W	0	0	STALL handshake. For noncontrol, nonisochronous endpoints: The application sets this bit to stall all tokens from the USB host to this endpoint. If a NAK bit, global nonperiodic IN NAK, or global OUT NAK is set along with this bit, the STALL bit takes priority. Only the application can clear this bit, never the USB core. For control endpoints: The application can only set this bit, and the USB core clears it, when a SETUP token is received for this endpoint. If a NAK bit, global nonperiodic IN NAK, or global OUT NAK is set along with this bit, the STALL bit takes priority. Regardless of this bit's setting, the USB core always responds to SETUP data packets with an ACK handshake.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<20>	SNP	R/W	0	0	Snoop mode. This bit configures the endpoint to snoop mode. In snoop mode, the USB core does not check the correctness of OUT packets before transferring them to application memory.
<19:18>	EPTYPE	R/W	0x0	0x0	Endpoint type. This is the transfer type supported by this logical endpoint. 00 = control 01 = isochronous 10 = bulk 11 = interrupt
<17>	NAKSTS	RO	0	0	NAK status. Indicates the following: 0 = USB core is transmitting non-NAK handshakes based on the FIFO status 1 = USB core is transmitting NAK handshakes on this endpoint. When either the application or the USB core sets this bit: ●The core stops receiving any data on an OUT endpoint, even if there is space in the RxFIFO to accommodate the incoming packet.
<16>	DPID	RO	0	0	For interrupt/bulk IN and OUT endpoints: endpoint data PID (DPID) Contains the PID of the packet to be received or transmitted on this endpoint. The application should program the PID of the first packet to be received or transmitted on this endpoint, after the endpoint is activated. Applications use the SETD1PID and SETD0PID fields of this register to program either data0 or data1 PID. 0 = data0 1 = data1 For isochronous IN and OUT endpoints: even/odd (micro)frame (EO_FRNUM) Indicates the (micro)frame number in which the core transmits/receives isochronous data for this endpoint. The application should program the even/odd (micro) frame number in which it intends to transmit/receive isochronous data for this endpoint using the SETEVNFR and SETODDFR fields in this register. 0 = even (micro)frame 1 = odd (micro)frame
<15>	USBACTEP	R/W	1	0	USB active endpoint. Indicates whether this endpoint is active in the current configuration and interface. The USB core clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit.
<14:11>	—	RO	—	—	Reserved.
<10:0>	MPS	R/W	0x0	0x0	Maximum packet size. Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.

Device OUT Endpoint Interrupt Registers USBC_DOEPINT(0..4)

This register contains the. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:5>	—	RO	—	—	Reserved.
<4>	OUTTKNEPDIS	R/W1C	0	0	OUT token received when endpoint disabled. Applies only to control OUT endpoints. Indicates that an OUT token was received when the endpoint was not yet enabled. This interrupt is asserted on the endpoint for which the OUT token was received.
<3>	SETUP	R/W1C	0	0	SETUP phase done. Applies to control OUT endpoints only. Indicates that the SETUP phase for the control endpoint is complete and no more back-to-back SETUP packets were received for the current control transfer. On this interrupt, the application can decode the received SETUP data packet.
<2>	AHBERR	R/W1C	0	0	AHB error. This is generated only in internal-DMA mode when there is an AHB error during an AHB read/write operation. The application can read the corresponding endpoint DMA address register to get the error address.
<1>	EPDISBLD	R/W1C	0	0	Endpoint disabled interrupt. This bit indicates that the endpoint is disabled per the application's request.
<0>	XFERCOMPL	R/W1C	0	0	Transfer completed interrupt. Indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.

Device OUT Endpoint 0 Transfer-Size Register USBC_DOEPTSIZ0

This register contains the. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	—	RO	—	—	Reserved.
<30:29>	SUPCNT	R/W	0x0	0x0	SETUP packet count. This field specifies the number of back-to-back SETUP data packets the endpoint can receive. 01 = 1 packet 10 = 2 packets 11 = 3 packets
<28:20>	—	RO	—	—	Reserved.
<19>	PKTCNT	R/W	0	0	Packet count. Indicates the total number of USB packets that constitute the transfer-size amount of data for endpoint 0. This field is decremented every time a packet (maximum-size or short packet) is read from the TxFIFO.
<18:7>	—	RO	—	—	Reserved.
<6:0>	XFERSIZE	R/W	0x0	0x0	Transfer size. Indicates the transfer size in bytes for endpoint 0. The USB core interrupts the application only after it has exhausted the transfer-size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. The USB core decrements this field every time a packet from the external memory is written to the TxFIFO.

Device OUT Endpoint Transfer-Size Registers USBC_DOEPTSIZ(1..4)

The application must modify this register before enabling the endpoint. Once endpoint 0 is enabled (using either USBC_DIEPCTL0[EPENA] or USBC_DOEPCTL0[EPENA]), the USB core modifies this register. The application can only read this register once the USB core has cleared the EPENA bit. This register is used only for endpoints other than endpoint 0. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31>	—	RO	—	—	Reserved.
<30:29>	MC	R/W	0x0	0x0	Multicount. Received data PID (RxDPID). Applies to isochronous OUT endpoints only. This is the data PID received in the last packet for this endpoint. 00 = data0 01 = data1 10 = data2 11 = Mdata SETUP packet count. Applies to control OUT endpoints only. This field specifies the number of back-to-back SETUP data packets the endpoint can receive. 01 = 1 packet 10 = 2 packets 11 = 3 packets
<28:19>	PKTCNT	R/W	0x0	0x0	Packet count. Indicates the total number of USB packets that constitute the transfer-size amount of data for this endpoint. OUT endpoints: This field is decremented every time a packet (maximum-size or short packet) is written to the RxFIFO.
<18:0>	XFERSIZE	R/W	0x0	0x0	Transfer size. This field contains the transfer size in bytes for the current endpoint. The USB core only interrupts the application after it has exhausted the transfer-size amount of data. The transfer size can be set to the maximum-packet size of the endpoint, to be interrupted at the end of each packet. OUT endpoints: The USB core decrements this field every time a packet is read from the RxFIFO and written to the external memory.

Power and Clock-Gating Control Register USBC_PCGCCTL

The application can use this register to control the USB core's power-down and clock gating features. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:5>	—	RO	—	—	Reserved.
<4>	PHYSUSPENDED	RO	0	0	PHY suspended. Indicates that the PHY has been suspended. After the application sets STOPPCLK (bit <0>), this bit is updated once the PHY is suspended. Since the UTMI+ PHY suspend is controlled through a port, the UTMI+ PHY is suspended immediately after STOPPCLK is set. However, the ULPI PHY takes a few clocks to suspend, because the suspend information is conveyed through the ULPI protocol to the ULPI PHY.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<3>	RSTPDWNMODULE	R/W	0	0	Reset power-down modules. This bit is valid only in partial-power-down mode. The application sets this bit when the power is turned off. The application clears this bit after the power is turned on and the PHY clock is up.
<2>	PWRCLMP	R/W	0	0	Power clamp. This bit is only valid in partial-power-down mode. The application sets this bit before the power is turned off to clamp the signals between the power-on modules and the power-off modules. The application clears the bit to disable the clamping before the power is turned on.
<1>	GATEHCLK	R/W	0	0	Gate HCLK. The application sets this bit to gate HCLK to modules other than the AHB slave and master and wakeup logic when the USB is suspended or the session is not valid. The application clears this bit when the USB is resumed or a new session starts.
<0>	STOPPCLK	R/W	0	0	Stop PCLK. The application sets this bit to stop the PHY clock (phy_clk) when the USB is suspended, the session is not valid, or the device is disconnected. The application clears this bit when the USB is resumed or a new session starts.

NPTX Data FIFO Registers USBC_NPTXDFIFO(0..7)

A slave-mode application uses one of these registers to access the Tx FIFO for channel *n*. See Table 21–3 for the address.

Bit Pos	Field Name	Field Type	Reset Value	Typical Value	Field Description
<31:0>	DATA	R/W	0x0	0x0	Reserved.

Chapter 22

Electrical Specifications

This chapter contains the following subjects:

Overview

This chapter provides absolute maximum ratings, recommended operating conditions, power consumption, and DC characteristics for all I/O types present on the CN50XX.

22.1 Absolute Maximum Ratings

If absolute maximum ratings are exceeded, the device may fail permanently. Device operation at or above these limits is not recommended. Exposure to absolute maximum ratings for extended periods of time may also diminish device function.

Table 22–1 Voltage Absolute Maximum Ratings

Parameter	Description	Min	Max	Unit
VDD	Core supply voltage	–0.3	1.32	V
VDD18_DDR	DDR2 SSTL-2 (1.8V) supply voltage	–0.3	2.1	V
VDD33	PCI3 supply voltage	–0.3	3.97	V
VDD25_RGM	RGMII/GMII/MII (2.5V) supply voltage	–0.3	2.8	V
USB_VDD33	USB (3.3V) supply voltage	–0.3	3.97	V
V_{IN}-3.3	PCI3 (3.3V) input voltage	–0.3	3.97	V
V_{IN}-SSTL18	SSTL18 input voltage	–0.3	2.1	V
DDR_VREF	DDR2 interface reference voltage	–0.3	2.1	V
DDR_PLL_VDD33	DDR2 PLL supply voltage	–0.3	3.6	V
PLL_VDD33	Core PLL supply voltage	–0.3	3.6	V
PCI_DLL_VDD33	PCI DLL supply voltage	–0.3	3.6	V

22.1.1 Absolute Maximum Storage Temperatures

Table 22–2 shows the CN50XX maximum storage temperature range.

Table 22–2 Absolute Maximum Storage Temperatures

Parameter	Description	Min	Max	Unit
$T_{STORAGE}$	Storage temperature	–40	120	°C

22.2 Recommended Operating Conditions

Table 22–3 shows recommended commercial-grade operating conditions, defining the upper and lower limits between which the device is tested to function.

Table 22–3 Recommended Operating Temperatures (Commercial Grade)

	Junction Temp. (T_J)		Case Temp. (T_C)		
Frequency	Min	Max	Min	Max	Unit
300/350/400 MHz	0	110	0	90	°C
500 MHz	0	110	0	85	°C
600 MHz	0	105	0	75	°C
700 MHz	0	105	0	70	°C

NOTE: The CN50XX chips are tested to the specified case temperatures (T_C). The junction temperature (T_J) is provided for guidance purposes. When you design the system thermal solution, the maximum case temperature must not be exceeded.

22.2.1 Supply Voltages for the Chip Core Voltage and External Interfaces

Table 22–4 shows CN50XX's recommended supply voltages for the chip core voltage and external interfaces.

Table 22–4 Recommended Operating Supply Voltages

Parameter	Description	Min	Typical	Max	Unit
VDD	Core supply voltage for 300/350/400/500 MHz	1.0	1.05	1.1	V
VDD	Core supply voltage for 600 MHz	1.05	1.1	1.15	V
VDD	Core supply voltage for 700 MHz	1.15	1.2	1.25	V
VDD18_DDR	DDR2 SSTL-2 (1.8V) supply voltage	1.7	1.8	1.9	V
VDD33	PCI3 supply voltage	3.14	3.3	3.46	V
VDD25_RGM	RGMII (2.5V) supply voltage	2.375	2.5	2.625	V
USB_VDD33	USB supply voltage	3.07	3.3	3.53	V

22.2.2 Supply Voltages for the On-Chip PLLs and DLLs

Table 22–5 shows CN50XX's supply voltages for the on-chip PLLs and DLLs.

Table 22–5 Power Supplies

Supply	Description	Min	Typical	Max	Unit
DDR_PLL_VDD33	DDR2 PLL supply voltage	3.14	3.3	3.46	V
PLL_VDD33	Core PLL supply voltage	3.14	3.3	3.46	V
PCI_DLL_VDD33	PCI DLL supply voltage	3.14	3.3	3.46	V

22.2.3 Reference Voltages

Table 22–6 shows CN50XX's reference voltage supplies.

Table 22–6 VREF Parameters

Parameter	Description	Min	Typical	Max	Unit
DDR_VREF	DDR2 interface reference voltage	0.85	0.9	0.95	V

22.3 Power Sequencing

The CN50XX has a core voltage, multiple interface supply voltages and several PLL and DLL voltages. These supply voltages must be brought online or offline in a specific sequence to ensure correct device functionality, stability, and reliability.

22.3.1 Power Up

The power-up sequence is dependent on the value of PCI_HOST_MODE. The sequences for when PCI host mode is enabled and disabled are explained in the following subsections.

22.3.1.1 Power Sequencing with PCI_HOST_MODE = 1

The power-up sequence in host mode is specified in the following steps.

1. Enable all PLL reference clocks before or at the same time that VDD33 is powered. PLL_DCOK must be deasserted low, and CHIP_RESET_L must be asserted low during powerup.
2. Once VDD33 is enabled, all PLL/DLL voltages and other interface voltages based on 3.3V can be enabled in any order. This may be done before, coincident with, or after VDD is powered up.
3. Power up VDD once VDD33 is stable.
4. Enable VDD18_DDR after VDD is stable.
5. Power up all other interface voltages in any order.
6. Assert signal PLL_DCOK at least 3 ms after the power supplies are stable. The CN50XX samples PLL_MUL on the rising edge of PLL_DCOK to determine the internal core-clock and DDR-memory-clock multiplier ratios respectively.
7. Deassert signal CHIP_RESET_L high at least 1 ms after PLL_DCOK is asserted.

Figure 22–1 shows the power-up sequence with the PCI in host mode.

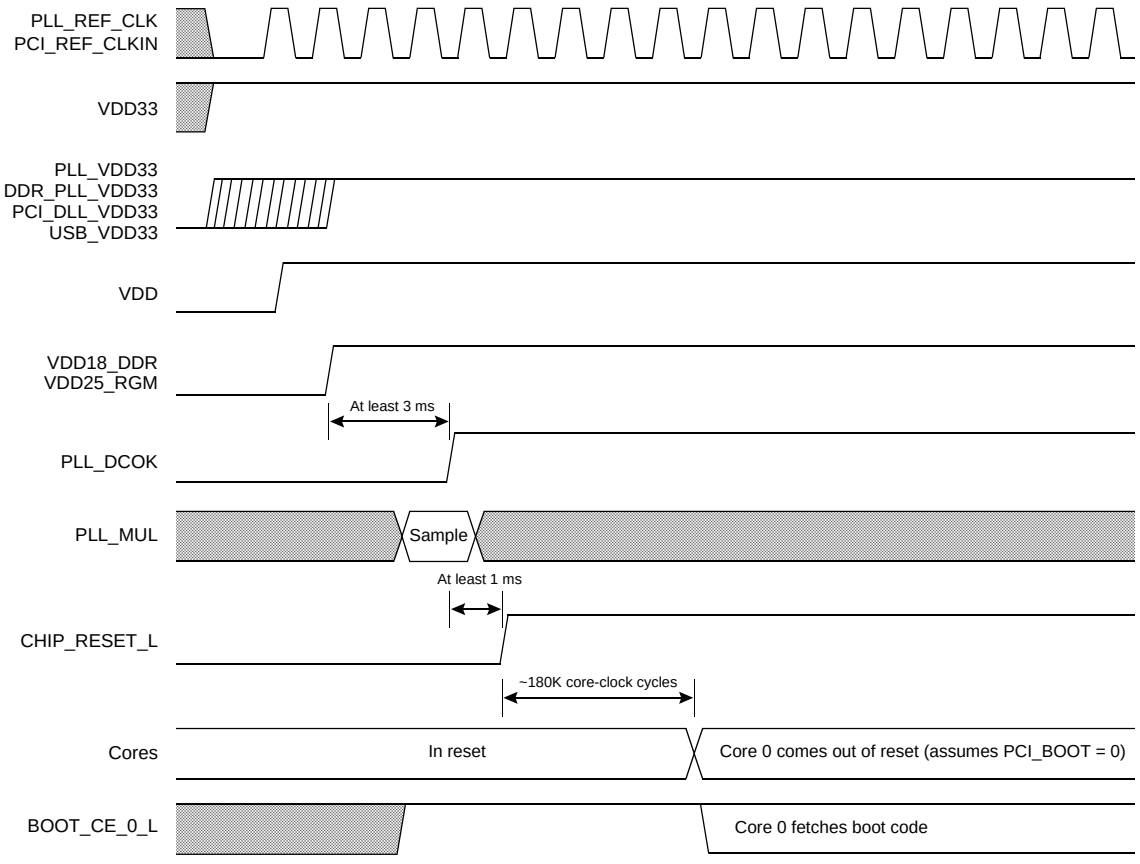


Figure 22–1 Power Sequencing with PCI_HOST_MODE=1

22.3.1.2 Power Sequencing with PCI_HOST_MODE = 0

The power-up sequence in non-host mode is specified in the following steps.

1. Enable all PLL reference clocks before or at the same time that VDD33 is powered. PLL_DCOK must be deasserted low, and PCI_RST_L must be asserted low during powerup.
2. Once VDD33 is enabled, all PLL/DLL voltages and other interface voltages based on 3.3V can be enabled in any order. This may be done before, coincident with, or after VDD is powered up.
3. Power up VDD once VDD33 is stable.
4. Enable VDD18_DDR after VDD is stable.
5. Power up all other interface voltages in any order.
6. Assert signal PLL_DCOK at least 3 ms after the power supplies are stable. The CN50XX samples PLL_MUL on the rising edge of PLL_DCOK to determine the internal core-clock and DDR-memory-clock multiplier ratios respectively.
7. Deassert signal PCI_RST_L high at least 1 ms after PLL_DCOK is asserted.

Figure 22–2 shows the power-up sequence in non-host mode.

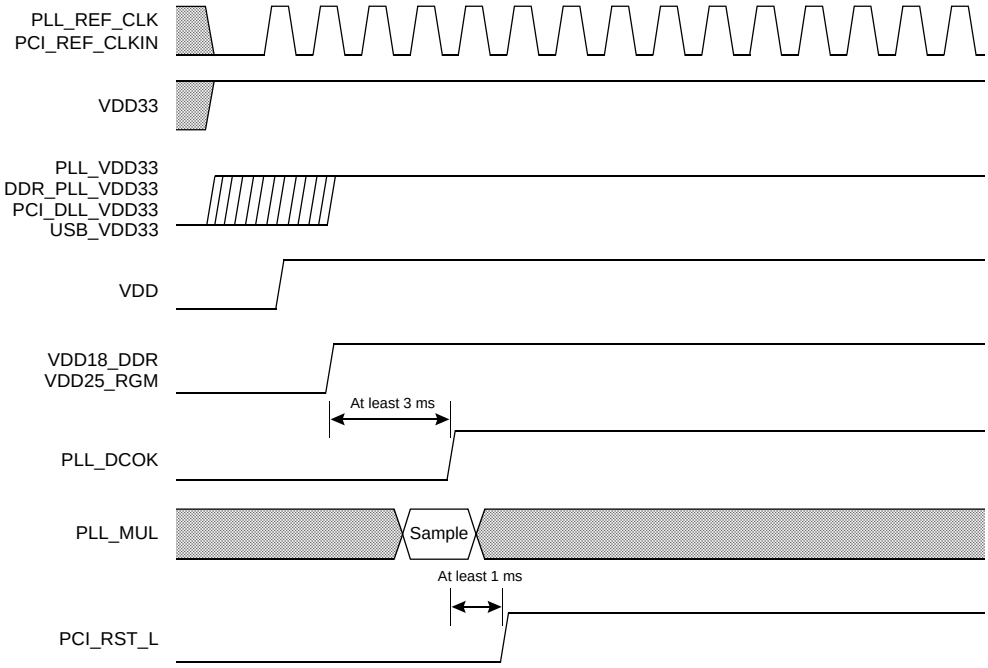


Figure 22–2 Power Sequencing with PCI_HOST_MODE=0

Refer to Section 9.11.2, “PCI Reset Sequence in Non-Host Mode”, on page 399 for more detail about the PCI reset sequence in non-host mode.

22.3.2 Power Down

Power supplies can be removed in any order without any restrictions.

NOTE: When powering down, the reference clock must not be stopped until PLL_DCOK has been deasserted.

22.4 Power Consumption

Table 22–7 lists CN50XX's power consumption.

Table 22–7 CN50XX Core Power Supply Specification

Frequency/ Voltage	Parameter	Description	1 Core	2 Cores	Units
300 MHz @ 1.05V	ICC	ICC for VDD supply	1.8	2.0	A
	ICC_{Rst}	ICC for VDD supply (reset)	1.8	1.8	A
350 MHz @ 1.05V	ICC	ICC for VDD supply	2.0	2.2	A
	ICC_{Rst}	ICC for VDD supply (reset)	2.0	2.0	A
400 MHz @ 1.05V	ICC	ICC for VDD supply	2.2	2.4	A
	ICC_{Rst}	ICC for VDD supply (reset)	2.2	2.2	A
500 MHz @ 1.05V	ICC	ICC for VDD supply	2.4	2.8	A
	ICC_{Rst}	ICC for VDD supply (reset)	2.4	2.4	A
600 MHz @ 1.1V	ICC	ICC for VDD supply	3.2	3.7	A
	ICC_{Rst}	ICC for VDD supply (reset)	3.2	3.2	A
700 MHz @ 1.2V	ICC	ICC for VDD supply	3.9	4.4	A
	ICC_{Rst}	ICC for VDD supply (reset)	3.9	3.9	A

Table 22–8 shows CN50XX's DDR2 memory interface supply current.

Table 22–8 DDR2 Memory Interface Supply Current

Parameter	Description	DDR Data Rate (MHz)			Units
		400	533	667	
I_{DDR18} (@1.8V)	ICC for VDD18 supply for 36-bit wide memory	234	311	390	mA
I_{DDR18} (@1.8V)	ICC for VDD18 supply for 18-bit wide memory	117	155	195	mA

Table 22–9 shows CN50XX's USB 2.0 interface supply current.

Table 22–9 USB 2.0 Interface (3.3V) Supply Current For Different Speed Modes (HS/FS/LS)

Parameter	Description	Supply Current	Units
$I_{HSTRANS}$ (@3.3V)	HS transmit: represents the average current drawn from USB_VDD33 supply, during packet transmission while driving a 10-pF load	4.0	mA
$I_{FSTRANS}$ (@3.3V)	FS transmit: represents the average current drawn from USB_VDD33 supply, during packet transmission while driving a 50-pF load	16.0	mA
$I_{LSTRANS}$ (@3.3V)	LS transmit: represents the average current drawn from USB_VDD33 supply, during packet transmission while driving a 600-pF load	16.0	mA

Table 22–10 shows CN50XX's 3.3-V supply current for the PCI interface and the other miscellaneous I/O interfaces.

Table 22–10 PCI3/Miscellaneous Supply Current

Parameter	Description	Misc. I/O Interfaces	PCI FREQ (MHz)		Units
			33	66	
I_{PCI33} (@3.3V)	ICC for VDD33 supply at 32-bits	40	30	50	mA

Table 22–11 shows CN50XX's RGMII/GMII/MII supply current per port.

Table 22–11 RGMII/GMII/MII Supply Current Per Port

Parameter	Description	RGMII Supply Current	Units
$I_{RGMII25}$ (@2.5V) [1]	ICC for VDD_INT1 supply	110	mA

1. For possible combinations of ports, refer to Table 13–1.

Table 22–12 shows CN50XX's on-chip PLL/DLL supply current.

Table 22–12 On-Chip PLL/DLLs Supply Current

Supply	Description	Min	Typical	Max	Unit
DDR_PLL_VDD33[1]	DDR2 PLL supply voltage	32	36	40	mA
PLL_VDD33[1]	Core PLL supply voltage	0.1	0.5	1.0	mA
PCI_DLL_VDD33	PCI DLL supply voltage	16	18	20	mA

1. The DDR PLL supply feeds the core PLL as well as the DDR PLL. PLL_VDD33 is not being used on the chip. It is specified only for 30XX-compatibility purposes.

22.5 DC Electrical Characteristics

The following tables describe the DC electrical characteristics for different types and combinations of input receivers and output drivers on the CN50XX chip.

22.5.1 2.5V CMOS Point-to-Point I/O for the RGMII/GMII/MII Interface

Table 22–13 lists CN50XX's 2.5V CMOS point-to-point I/O for the RGMII/GMII/MII interface electrical characteristics.

Table 22–13 2.5V CMOS Point-to-Point I/O for RGMII/GMII/MII Interface

Symbol	Parameter	Min	Nom	Max	Unit
VDD_INT	RGMII interface supply voltage	2.375	2.5	2.625	V
VIH (DC)	DC input logic high	1.7	—	3.6[1]	V
VIL (DC)	DC input logic low	–0.3	—	0.7	V
VOL (DC)	DC output logic low	0	—	—	V
VOH (DC)	DC output logic high	—	—	VDD25_RGM	V
ILEAK (DC)	DC input leakage current	—	—	1	µA
Rout_up	Output impedance (pull-up)	0.9 × Rext_up_RGM[2]	Rext_up_RGM	1.1 × Rext_up_RGM	Ω
Rout_dn	Output impedance (pull-down)	0.9 × Rext_up_RGM[3]	Rext_up_RGM	1.1 × Rext_up_RGM	Ω
CPIN	RGMII interface pin capacitance	—	—	2.5	pF

1. The RGMII/GMII/MII receivers are 3.3V tolerant.
2. Rext_up_RGM is an external compensation resistor (50 Ω).
3. Rext_dn_RGM is an external compensation resistor (50 Ω)

22.5.2 SSTL18 Bidirectional I/O for the DDR2 Memory Interface

Table 22–14 lists CN50XX's SSTL18 bidirectional I/O for the DDR2 memory interface electrical characteristics.

Table 22–14 SSTL18 Bidir I/O for DDR2 Memory Interface

Symbol	Parameter	Min	Nom	Max	Unit
VDD18_DDR	DDR interface supply voltage	1.7	1.8	1.9	V
DDR_VREF	DDR interface input reference voltage	0.85	0.9	0.95	V
VIH (DC)	DC input logic high	DDR_VREF+0.125	—	VDD18_DDR+0.3	V
VIL (DC)	DC input logic low	–0.3	—	DDR_VREF–0.125	V
VIH (AC)	AC input logic high	DDR_VREF+0.25	—	—	V
VIL (AC)	AC input logic low	—	—	DDR_VREF–0.25	V
VOH (DC)[1]	DC output logic high	VDD18_DDR–0.5	—	VDD18_DDR–0.2	V
VOL (DC)[2]	DC output logic low	0.2	—	0.5	V
IOH (DC)[3]	DC output pull-up current	10	—	14	mA
IOL (DC)[4]	DC output pull-down current	10	—	14	mA
ILEAK (DC)	DC input leakage current	—	—	1	µA
Rout_up	Output impedance (pull-up)	0.9 × Rext_up_ddr[5]	Rext_up_ddr	1.1 × Rext_up_ddr	Ω
Rout_dn	Output impedance (pull-down)	0.9 × Rext_up_ddr[6]	Rext_up_ddr	1.1 × Rext_up_ddr	Ω
CPIN	DDR interface pin capacitance	—	—	2.5	pF
Differential Signals					
VIN (DC)[7]	DC input signal voltage	–0.3	—	VDD18_DDR + 0.3	V
VID (DC)	DC differential input voltage	0.25	—	VDD18_DDR + 0.6	V
VID (AC)	AC differential input voltage	0.5	—	VDD18_DDR + 0.6	V
VIX (AC)[8]	AC differential input crosspoint voltage	0.5 × VDD18_DDR – 0.175	—	0.5 × VDD18_DDR + 0.175	V
VOX (AC)[9]	AC differential output crosspoint voltage	0.5 × VDD18_DDR – 0.125	—	0.5 × VDD18_DDR + 0.125	V

1. VOH is specified with a 50Ω load to a termination voltage equal to 0.5 × VDD_DDR.
2. VOL is specified with a 50Ω load to a termination voltage equal to 0.5 × VDD_DDR.
3. IOH is specified with a 50Ω load to a termination voltage equal to 0.5 × VDD_DDR.
4. IOL is specified with a 50Ω load to a termination voltage equal to 0.5 × VDD_DDR.
5. Rext_up_ddr is an external compensation resistor (18Ω – 57Ω).
6. Rext_dn_ddr is an external compensation resistor (18Ω – 57Ω).
7. Allowable DC excursion for each differential input.
8. VIX (AC) indicates the voltage at which differential inputs must cross.
9. VOX (AC) indicates the voltage at which differential outputs must cross.

22.5.3 3.3V CMOS Bidirectional and Point-to-Point I/O for the PCI/Miscellaneous Interfaces

Table 22–15 lists CN50XX's 3.3V CMOS bidirectional and point-to-point I/O for the PCI and miscellaneous interfaces electrical characteristics.

Table 22–15 3.3V CMOS Point-to-Point I/O for PCI/Miscellaneous Interfaces

Symbol	Parameter	Min	Nom	Max	Unit
VDD33	Interface supply voltage	3.14	3.3	3.46	V
VIH (DC)	DC input logic high	0.6 × VDD33	—	VDD33 + 0.5	V
VIL (DC)	DC input logic low	–0.5	—	0.15 × VDD33	V
VOH (DC)	DC output logic high	0.9 × VDD33	—	—	V
VOL (DC)	DC output logic low	—	—	0.1 × VDD33	V
ILEAK (DC)	DC input leakage current	—	—	1	µA
Rout[1]	Driver output impedance (DC)	20	—	30	Ω

1.The source and sink currents for a driver can be computed for a specified load, and the driver supply voltage.

22.5.4 GMII/RGMII Reference-Clock Differential Input

Table 22–16 lists CN50XX's GMII/RGMII reference-clock differential input electrical characteristics.

Table 22–16 GMII/RGMII Reference-Clock Differential Input

Symbol	Parameter	Min	Nom	Max	Unit
VI	Input voltage range	–0.3	—	VDD25_RGM + 0.3	V
VID	Input differential voltage	400	—	—	mV
VCM	Input common mode voltage range	0.5	—	2	V
IIL	Receiver input leakage current	—	—	2	µA

Chapter 23

AC Characteristics

This chapter contains the following subjects:

23.1 Input Clocks

23.1.1 Reference-Clock Input

Table 23–1 lists CN50XX's reference-clock input electrical characteristics.

Table 23–1 Reference-Clock Input

Clock Name	Frequency	Frequency Tolerance	Duty Cycle	Period Jitter (P-P)[1]	Edge Rate[2]	Levels	Termination
PLL_REF_CLK	50 MHz	±150 ppm	40/60	150 ps	2 ns	0.0–3.3V	N/A
GMI_REF_CLK_P/N	125 MHz	±150 ppm	45/55	100 ps	1 ns	Refer to Table 22–16	N/A
PCI_PCLK	33/66 MHz	±150 ppm	40/60	200 ps	2 ns	0.0–3.3V	N/A

1. Cycle to cycle, peak to peak.
2. Edge rate is 10%–90% (rise) and 90%–10% (fall).

23.2 PCI Interface

23.2.1 PCI I/O Signal Timing

Table 23–2 lists CN50XX's PCI I/O timing.

Table 23–2 PCI I/O Timing

Parame-ter	Description	PCI 66 (Ref)		PCI 33 (Ref)		Units	Notes
		Min	Max	Min	Max		
T_{VAL}	PCI_PCLK to signal valid delay - bused signals	2	6	2	11	ns	1, 2
$T_{VAL(PTP)}$	PCI_PCLK to signal valid delay - point-to-point signals	2	6	2	12	ns	1, 2
T_{ON}	Float to active delay	2		2		ns	1, 6
TOFF	Active to float delay		14		28	ns	1, 6
T_{SU}	Input setup time to PCI_PCLK - bused signals	3		7		ns	2, 3, 7
$T_{SU(PTP)}$	Input Set up Time to PCI_PCLK - point-to-point signals	5		10, 12		ns	2, 3
T_{H}	Input Hold Time from PCI_PCLK	0		0		ns	3
T_{RST}	Reset Active Time	1		1		ms	4
$T_{RST-CLK}$	Reset Active Time after PCI_PCLK stable	100		100		µs	4
$T_{RST-OFF}$	RST# active to output float delay		40		40	ns	4,5
T_{RRSU}	REQ64# to RST# setup time	10		10		clocks	
T_{RRH}	RST# to REQ64# hold time	0	50	0	50	ns	
T_{RHFA}	RST# high to first Configuration access	225		225		clocks	
T_{RHFF}	RST# high to first FRAME# assertion	5		5		clocks	
T_{PVRH}	Power valid to RST# high	100		100		ms	
T_{RLCX}	Delay from RST# low to PCI_PCLK frequency change					ns	

Notes for Table 23–2:

1. See timing measurement conditions in Figure 23–1.
2. Setup time for point-to-point signals applies to REQ# and GNT# only. All other signals are bused.
3. See timing measurement conditions in Figure 23–2.
4. RST# is asserted and deasserted asynchronously with respect to CLK.

5. All output drivers must be floated when RST# is active.
6. For purposes of Active/Float timing measurements, the Hi-Z or “off” state is defined to be when the total current delivered through the component pin is less than or equal to the leakage current specification.
7. Setup time applies only when the device is not driving the pin. Devices cannot drive and receive signals at the same time.
8. Maximum value is also limited by delay to the first transaction (T_{RHFF}).

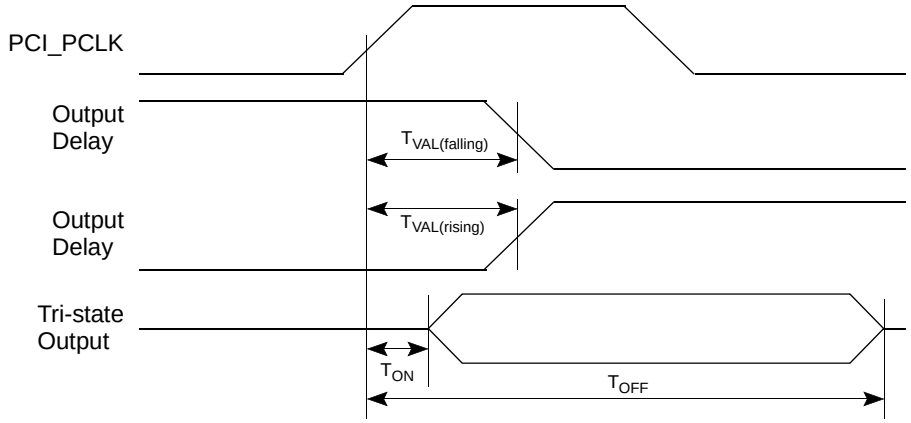


Figure 23–1 PCI Signal Output Timing Diagram

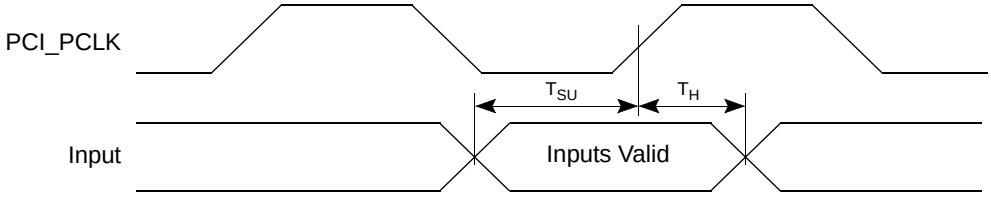


Figure 23–2 PCI Signal Input Timing Diagram

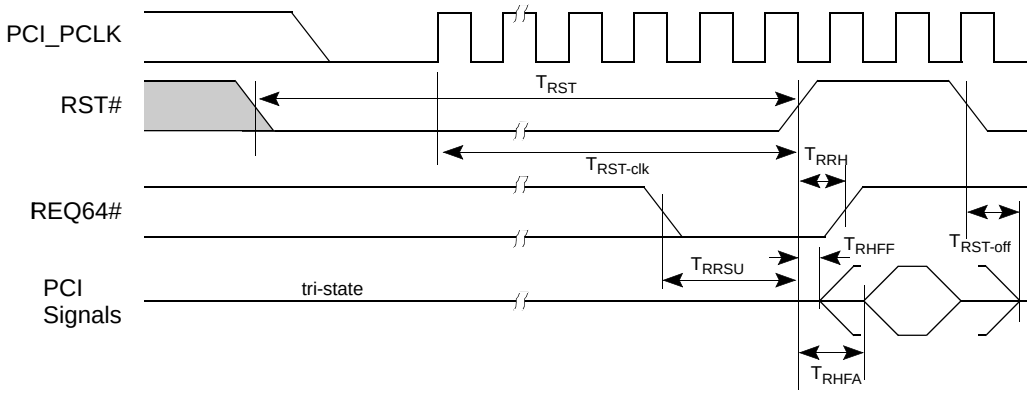


Figure 23–3 PCI Reset Timing Diagram

23.3 DDR2 SDRAM Interface

The SDRAM controller implements a subset of the operations allowed by the JEDEC DDR-SDRAM specification. CN50XX only implements and uses the READ with autoprecharge, WRITE with autoprecharge, and AUTOREFRESH operations and associated bus cycles. CN50XX also implements the complete initialization sequence required by DDR2 SDRAM parts.

NOTE: The DDR2 SDRAM controller only uses a burst size of 4 or 8. The DDR2 SDRAM parts used must be programmed to this value.

23.3.1 DDR2 SDRAM Bus-Cycle Commands

Tables 23–3 and 23–4 list CN50XX's DDR2 SDRAM bus-cycle commands.

Table 23–3 DDR SDRAM Bus Cycle Commands[1]

Parameter	RAS#	CAS#	WE#
NOP - no operation	H	H	H
ACT - active (select bank and activate row)	L	H	H
READ (select bank and column, and start read burst)	H	L	H
WRITE (select bank and column, and start write burst)	H	L	L
PRE - precharge (deactivate row in bank or banks) [2]	L	H	L
AR - autorefresh	L	L	H
LMR - load mode register	L	L	L

1. CKE is high for all commands shown.
2. CN50XX does not perform explicit precharge. It performs the latest possible autoprecharge from a prior Read-with-Precharge or Write-with-Precharge.

Table 23–4 DDR2 SDRAM Bus Cycle Commands

Parameter	CKE		CS#	RAS#	CAS#	WE#	BA0 BA1 BA2	[A15:A11]	[A10]	[A9:A0]
	Previous Cycle	Current Cycle								
(Extended) mode register set [1,2]	H	H	L	L	L	L	BA	Op code		
Refresh (REF)[1]	H	H	L	L	L	H	X[3]	X	X	X
Bank activate[1,2]	H	H	L	L	H	H	BA	Row address		
Write[1,2,4]	H	H	L	H	L	L	BA	Column	L	Column
Write with auto precharge[1,2,4]	H	H	L	H	L	L	BA	Column	H	Column
Read[1,2,4]	L	L	L	H	L	H	BA	Column	L	Column
Read with autoprecharge[1,2,4]	L	L	L	L	L	H	BA	Column	H	Column
NOP[1]	H	X	L	H	H	H	X	X	X	X
Device deselect[1]	H	X	H	X	X	X	X	X	X	X

1. All DDR2 SDRAM commands are defined by states of CS#, RAS#, CAS#, WE#, AND CKE at the rising edge of the clock.
2. Bank addresses BA0, BA1, and BA2 (BA) determine which bank is to be operated upon. For (e)MRS, BA selects an (extended) mode register.
3. X means "H or L (but a defined logic level)".
4. Burst reads or writes at BL=4 cannot be terminated or interrupted.

Table 23–5 lists CN50XX's DDR2 SDRAM I/O signal timing.

Notes for Table 23–5:

- All timing measured from 50% transition of signal to its reference signal.
- Edge rates and output delays are specified for a standard load on all output pins.

Table 23–5 DDR2 SDRAM I/O Signal Timing

Parameter	Description	Min	Typical	Max	Units
533 MHz					
T_{CK}	Clock cycle time	3.551	3.760	3.969	ns
T_{CH}	Clock high-level width	1.74	1.88	2.024	ns
T_{CL}	Clock low-level width	1.74	1.88	2.024	ns
T_{A}	A output delay from CK#	–127	0.0	127	ps
T_{BA}	BA output delay from CK#	–127	0.0	127	ps
T_{CMD}	RAS#N, CAS#N, WE#N output delay from CK#	–127	0.0	127	ps
T_{DQSH}	DQS high-level width	1.74	1.88	2.024	ns
T_{DQSL}	DQS low-level width	1.74	1.88	2.024	ns
T_{DS}	DQ setup time relative to DQS	–770	0.0	N/A	ps
T_{DH}	DQ hold time relative to DQS	1.13	1.88	N/A	ns
T_{DQS}	DQS output delay from CK	–127	0.0	127	ps
T_{DQ} [1]	DQ output delay from DQS	.763	.940	1.117	ns
CASLAT [2]	Effective CAS latency	2.0	—	5.5	T_{CK}
T_{HZ} [3]	DQ/DQS high-impedance window from CK/CK#	–500	—	500	ps
T_{LZ} [3]	DQ/DQS low-impedance window from CK/CK#	–500	—	500	ps
T_{RP}	Precharge command period	15	—		ns
T_{RFC}	Refresh-to-Active or Refresh-to-Refresh command interval	?	—	70,000	ns
667 MHz					
T_{CK}	Clock cycle time	2.810	3.000	3.190	ns
T_{CH}	Clock high-level width	1.377	1.500	1.627	ns
T_{CL}	Clock low-level width	1.377	1.500	1.627	ns
T_{A}	A output delay from CK#	–105	0.0	105	ps
T_{BA}	BA output delay from CK#	–105	0.0	105	ps
T_{CMD}	RAS#N, CAS#N, WE#N output delay from CK#	–105	0.0	105	ps
T_{DQSH}	DQS high-level width	1.377	1.500	1.627	ns
T_{DQSL}	DQS low-level width	1.377	1.500	1.627	ns
T_{DS}	DQ setup time relative to DQS	–580	0.0	N/A	ps
T_{DH}	DQ hold time relative to DQS	0.940	1.500	N/A	ns
T_{DQS}	DQS output delay from CK	–105	0.0	105	ps
T_{DQ} [1]	DQ output delay from DQS	605	750	895	ps
CASLAT [2]	Effective CAS latency	2.0	—	5.5	T_{CK}
T_{HZ} [3]	DQ/DQS high-impedance window from CK/CK#	–500	—	500	ps
T_{LZ} [3]	DQ/DQS low-impedance window from CK/CK#	–500	—	500	ps
T_{RP}	Precharge command period	15	—		ns

Table 23–5 DDR2 SDRAM I/O Signal Timing (Continued)

Parameter	Description	Min	Typical	Max	Units
T_{RFC}	Refresh-to-Active or Refresh-to-Refresh command interval	?	—	70,000	ns

1. Relative to DQS. This signal is logically driven $T_{CK}/4$ before the corresponding DQS.
2. CASLAT is a CN50XX setting that sets the effective CAS latency seen by CN50XX. CASLAT must have an integer value between 3.0 and 6.0. CASLAT must be chosen such that the first DQS latching transition falls within the allowed range.
3. Measured from the closest CK/CK#. The absolute transition time is dependent on the CASLAT parameter.

23.3.2 DDR2 SDRAM Read Operations

CN50XX always performs burst = (4 or 8) reads with final read being a read-with-precharge. Figure 23–4 demonstrates two back-to-back READ commands of BURST = 4 and CASLAT = 3.0

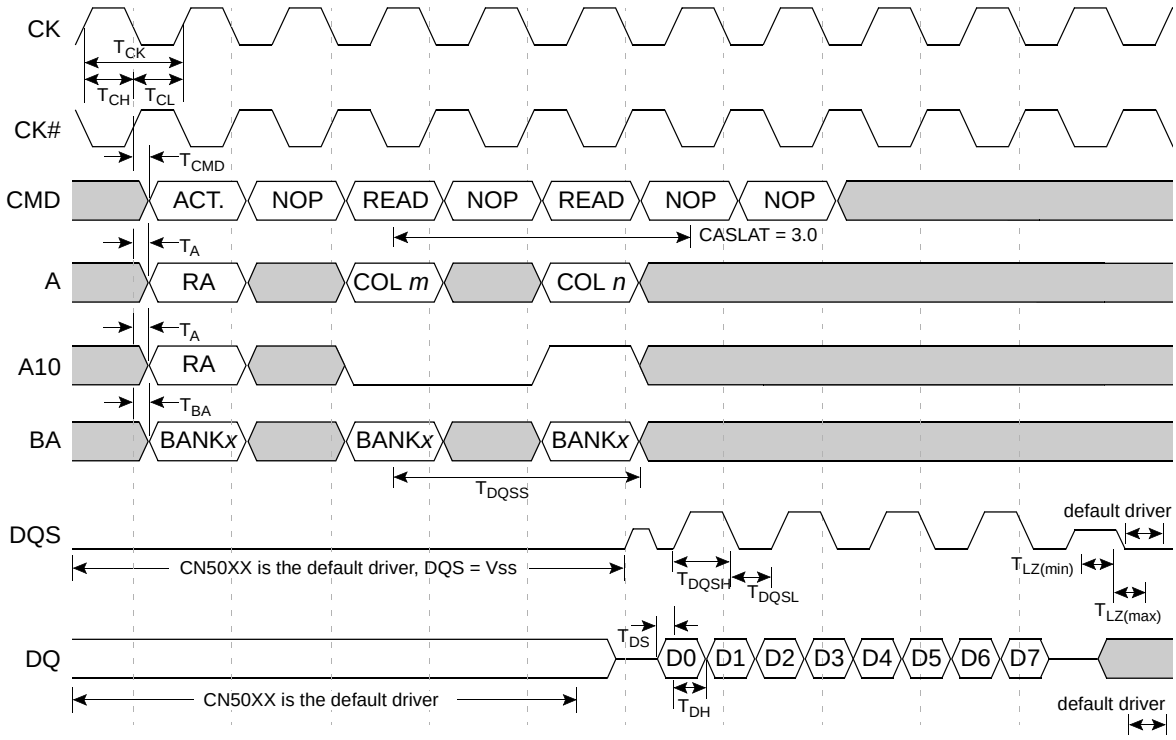


Figure 23–4 Read with Autoprecharge Timing Diagram

23.3.3 SDRAM Write Operations

CN50XX always performs burst = (4 or 8) writes with final write being a write-with-precharge. Burst = 4 is shown in Figure 23–5.

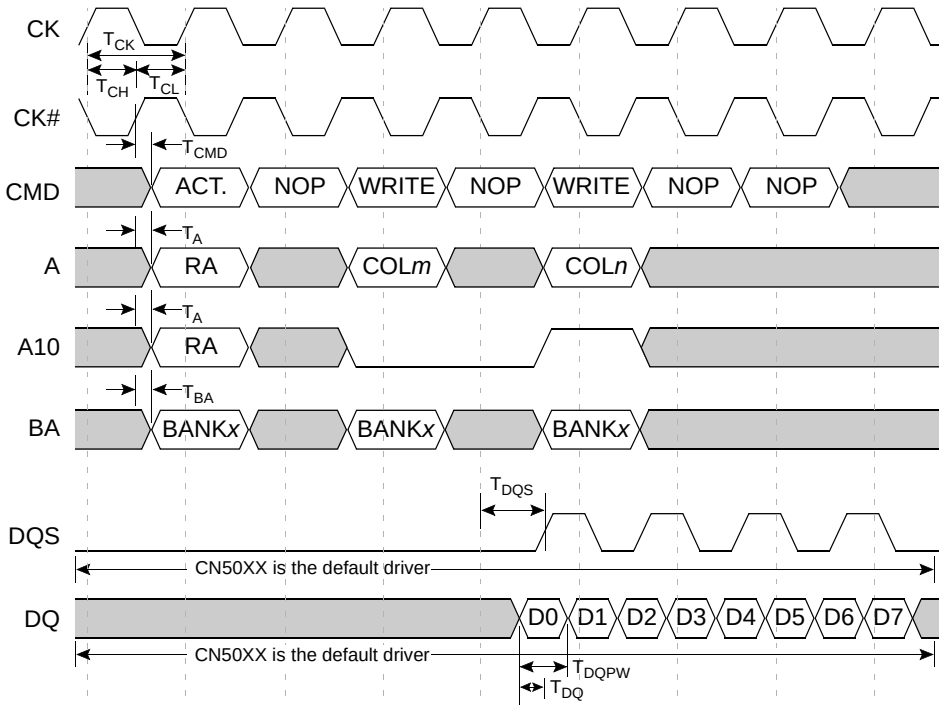


Figure 23–5 Write with Auto Precharge Timing Diagram

23.3.4 SDRAM Autorefresh Operations

Automatic refresh timing is shown in Figure 23–6. The following parameters are shown:

- PRE* - CN50XX does not do explicit precharges. PRE* specifies the latest possible autoprecharge from a prior read with precharge or write with precharge.
- ACT*, RA*, BANK* - Earliest possible subsequent activation for a read or a write.
- T_{RP} - Precharge command period; configuration-specific.
- T_{RFC} - Active to Active/Auto Refresh command period; configuration-specific.
- The T_A and the T_{BA} parameters are all relative to the CK falling edge nearest them.

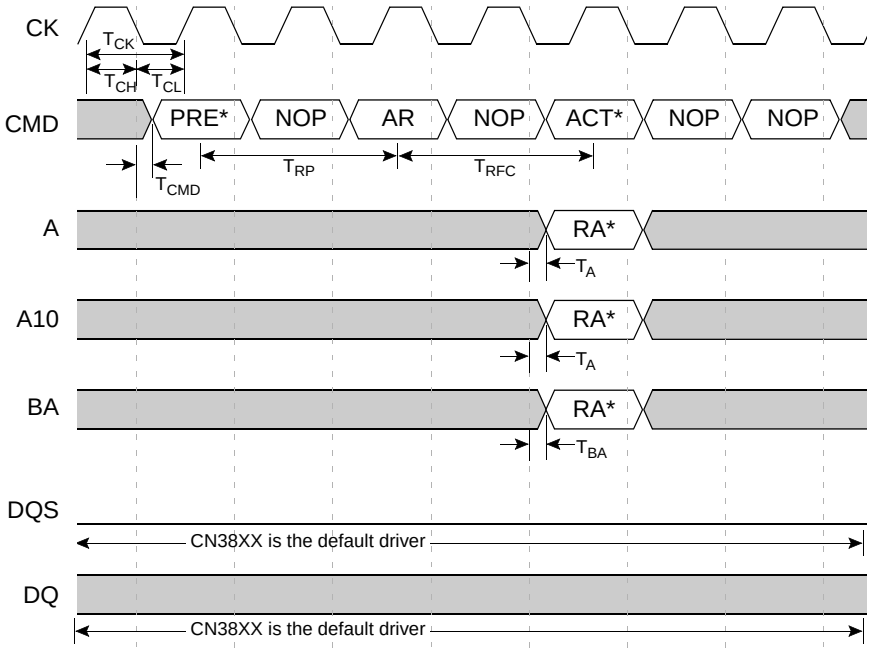
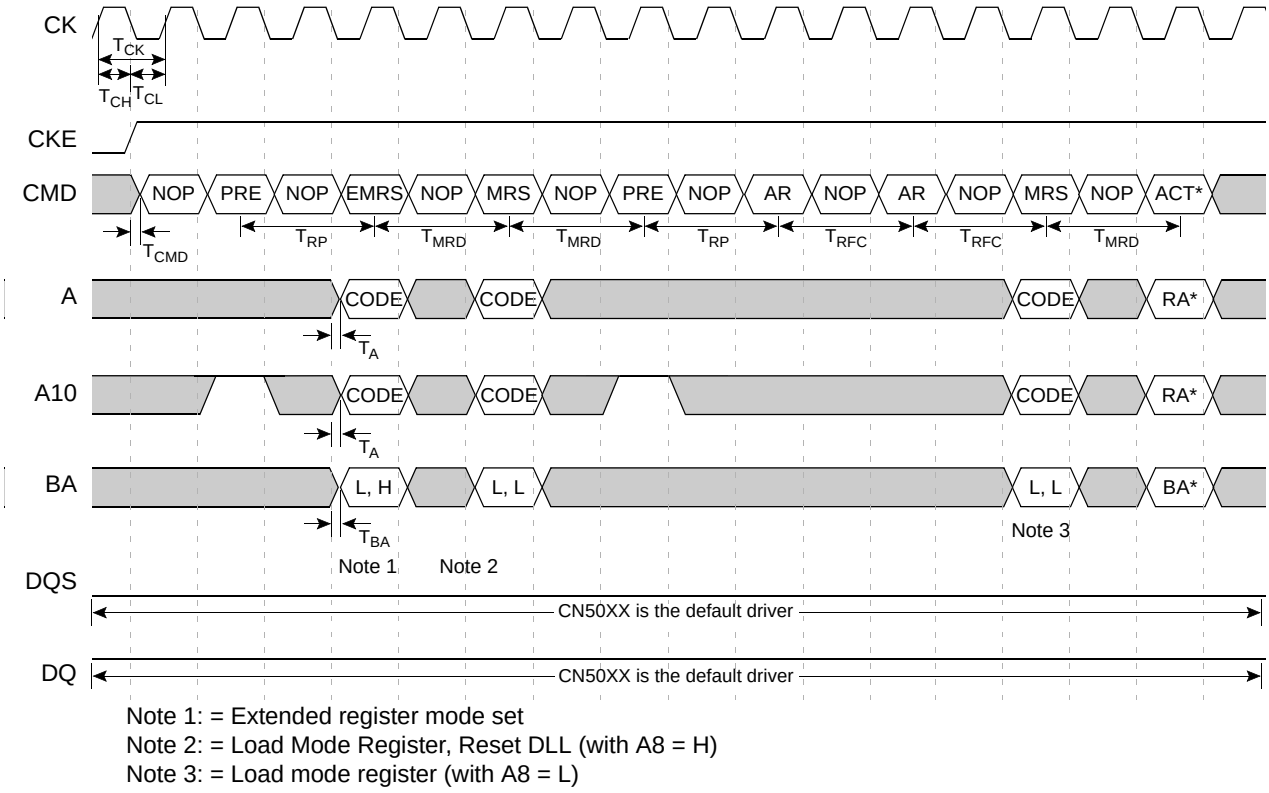


Figure 23–6 Auto Refresh Timing Diagram

23.3.5 SDRAM Initialize and Mode Register Operations

Initialize and mode register timing is shown in Figure 23–7. The following parameters are shown:

- CN50XX starts driving all outputs after (hardware) reset deasserts.
- Software assertion of T_{PU} (via power-up complete bit in the CMC register) causes CKE to assert and the init sequence to take place. CKE remains asserted from this point forward.
- ACT*, RA*, BA* - earliest possible activate from a subsequent write.
- T_{RP} - Precharge command period; configuration-specific.
- T_{RFC} - Active to Active/Auto Refresh command period; configuration-specific.
- T_{MRD} - Load Mode Register command cycle time; configuration-specific.
- The T_A and the T_{BA} are all relative to the CK falling edge nearest them.



Note 1: = Extended register mode set
Note 2: = Load Mode Register, Reset DLL (with A8 = H)
Note 3: = Load mode register (with A8 = L)

Figure 23–7 Initialize and Mode Register Timing Diagram

23.4 RGMII Interface

Timing for this interface will be such that the clock and data are generated simultaneously by the source of the signals and therefore skew between the clock and data is critical to proper operation.

- T_{TXC} adds skew between TXC and TXD/TX_CTL based on the value of ASX0_TX_CLK_SET*[SETTING].
- T_{RXC} adds skew between RXC and RXD/RX_CTL based on the value of ASX0_RX_CLK_SET*[SETTING].

Each CSR setting can select from 0 to 31 delay settings with each setting representing 65ps. This gives a range of 0 to 2.015ns.

Table 23–6 lists CN50XX's timing parameters for RGMII, Figure 23–8 shows transmit timing, and Figure 23–9 shows receive timing.

Table 23–6 RGMII Timing Parameters

Parameter	Description	Min	Typical	Max	Units
T_{TXC}	Programmable delay on TXC	0	1.7	2.0	ns
T_{RXC}	Programmable delay on RXC	0	1.7	2.0	ns
$T_{SKEW}T$	Data to clock output skew (at transmitter)[1]	T_{TXC} – 0.2	T_{TXC}	T_{TXC} + 0.2	ns
$T_{SETUP}R$	Data to clock input setup	0.5 – T_{RXC}	2.0	—	ns
$T_{HOLD}R$	Data to clock input setup	0.5	2.0	—	ns
T_{CYC}	Clock cycle duration[2]	7.2	8	8.8	ns
Duty G	Duty cycle for Gigabit[3]	45	50	55	%
Duty T	Duty cycle for 10/100T[3]	40	50	60	%
T_R/T_F	Rise/fall time (20-80%)	—	0.75	—	ns

1. This implies that the PC board design will require clocks to be routed such that an additional trace delay of greater than 1.5 ns and less than 2.0 ns will be added to the associated clock signal. For 10/100T, the max value is unspecified. T_{TXC} and T_{RXC} are programmable delays that can be used to lessen or negate the need for board trace delay.
2. For 10Mbps and 100 Mbps, T_{CYC} scales to 400 ns + (–40 ns) and 40 ns + (–4 ns) respectively.
3. Duty cycle may be stretched/shrunk during speed changes or while transmitting to a received packet's clock domain, as long as minimum duty cycle is not violated and stretching occurs for no more than three T_{CYC} of the lowest speed transitioned between.

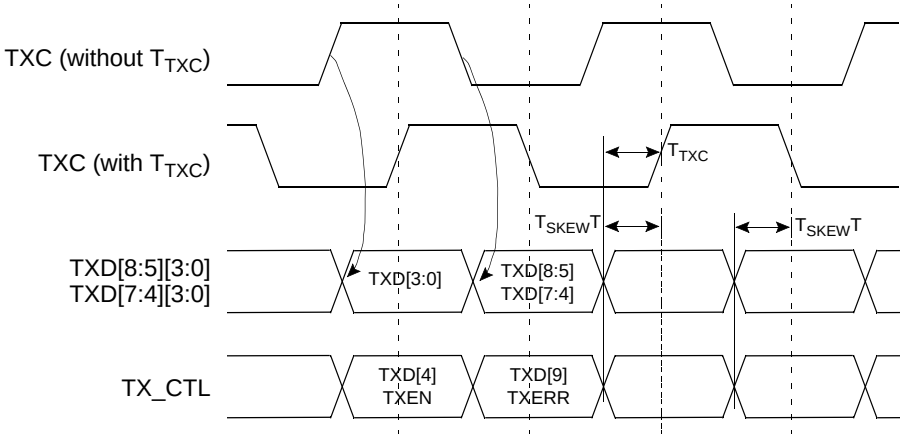


Figure 23–8 RGMII Transmit Multiplexing and Timing Diagram

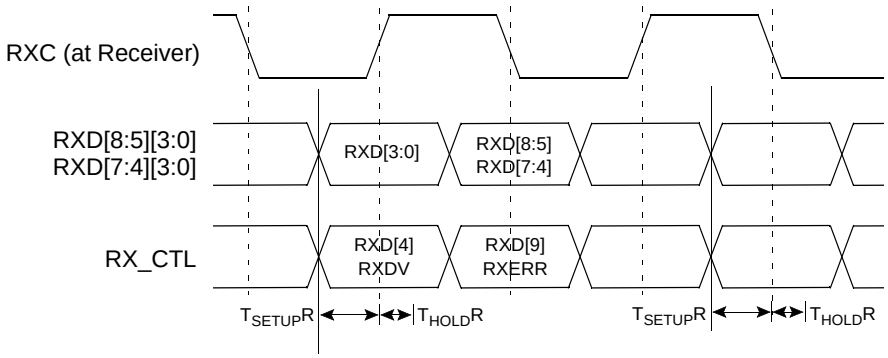


Figure 23–9 RGMII Receive Multiplexing and Timing Diagram

23.5 GMII Interface

Timing for this interface will be such that the clock and data are generated simultaneously by the source of the signals and therefore skew between the clock and data is critical to proper operation.

Table 23–7 lists CN50XX's timing parameters for GMII, Figure 23–10 shows transmit timing, and Figure 23–11 shows receive timing.

Table 23–7 GMII Timing Parameters

Parameter	Description	Min	Typical	Max	Units
T_{PERIOD}	GMI_GTXCLK period	7.50	8.0	8.50	ns
T_{PERIOD}	GMI_RXCLK period	7.50	—	—	ns
T_{HIGH}	GMI_GTXCLK, GMI_RXCLK time high	2.50	—	—	ns
T_{LOW}	GMI_GTXCLK, GMI_RXCLK time low	2.50	—	—	ns
T_{SETUP}	GMI_RXD, GMI_DV, and GMI_RXERR setup to GMI_RXCLK	2	—	—	ns
T_{HOLD}	GMI_RXD, GMI_DV, and GMI_RXERR hold from GMI_RXCLK	0	—	—	ns
T_{CQ}	GMI_TXD, GMI_TXERR, and GMI_TXEN output delay from GMI_GTXCLK	3.5	—	4.5	ns

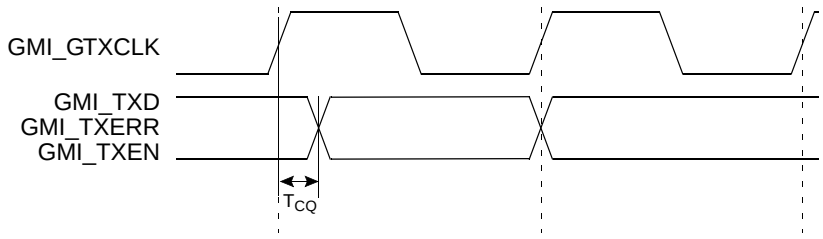


Figure 23–10 GMII Transmit Timing Diagram

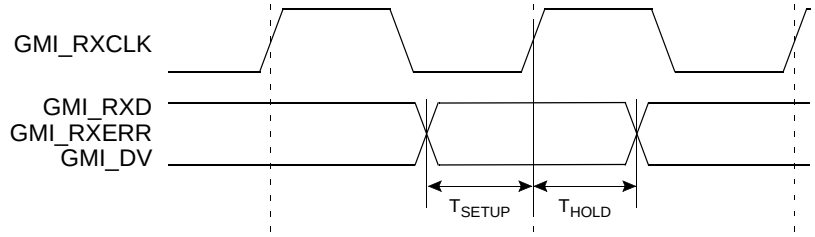


Figure 23–11 GMII Receive Timing Diagram

23.6 MII Interface

Timing for this interface will be such that the clock and data are generated simultaneously by the source of the signals and therefore skew between the clock and data is critical to proper operation.

Table 23–8 lists CN50XX's timing parameters for MII, Figure 23–12 shows transmit timing, and Figure 23–13 shows receive timing.

Table 23–8 MII Timing Parameters

Parameter	Description	Min	Typical	Max	Units
T_{PERIOD}	MI*n*_TXCLK period @ 10Mbps		400		ns
T_{PERIOD}	MI*n*_RXCLK period @ 10Mbps		400		ns
T_{PERIOD}	MI*n*_TXCLK period @ 100Mbps		40		ns
T_{PERIOD}	MI*n*_RXCLK period @ 100Mbps		40		ns
T_{SETUP}	MI*n*_RXD, MI*n*_DV, and MI*n*_RXERR setup to MI*n*_RXCLK	10	—	—	ns
T_{HOLD}	MI*n*_RXD, MI*n*_DV, and MI*n*_RXERR hold from MI*n*_RXCLK	10	—	—	ns
T_{CQ}	MI*n*_TXD, MI*n*_TXERR, and MI*n*_TXEN output delay from MI*n*_TXCLK	0	—	25	ns

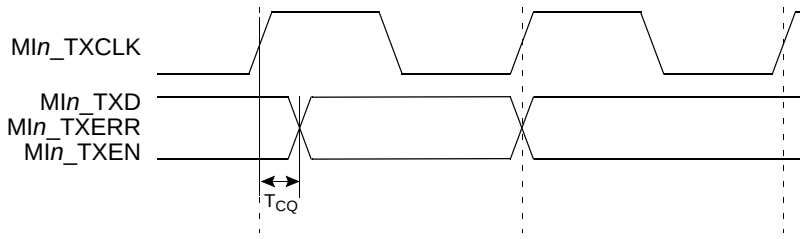


Figure 23–12 MII Transmit Timing Diagram

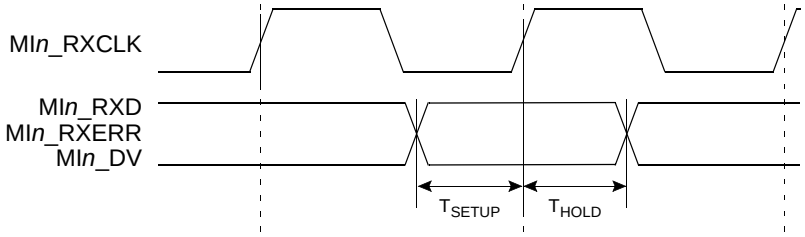


Figure 23–13 MII Receive Timing Diagram

23.7 EEPROM Interface

The CN50XX EEPROM interface is a serial interface to an external EEPROM, which can be used to initialize configuration data. At power-on, the PCI controller can load its read-only configuration registers from a 64 × 16 bit serial EEPROM (e.g. a standard 93C46 EEPROM).

NOTE: Only 64 × 16 bit EEPROMs are supported.

23.7.1 EEPROM Read Cycle

Table 23–9 lists CN50XX's EEPROM read-cycle timing, and Figures 23–14 and 23–15 provide the timing diagrams.

Table 23–9 EEPROM Read-Cycle Timing

Parameter	Description	Min	Max	Units
T_{DRV}	Drive Time(w.r.t selected delay offset)	0	10	ns
T_{SU}	Setup Time(w.r.t selected delay offset)	—	10	ns
T_{HD}	Hold Time(w.r.t selected delay offset)	10	—	ns

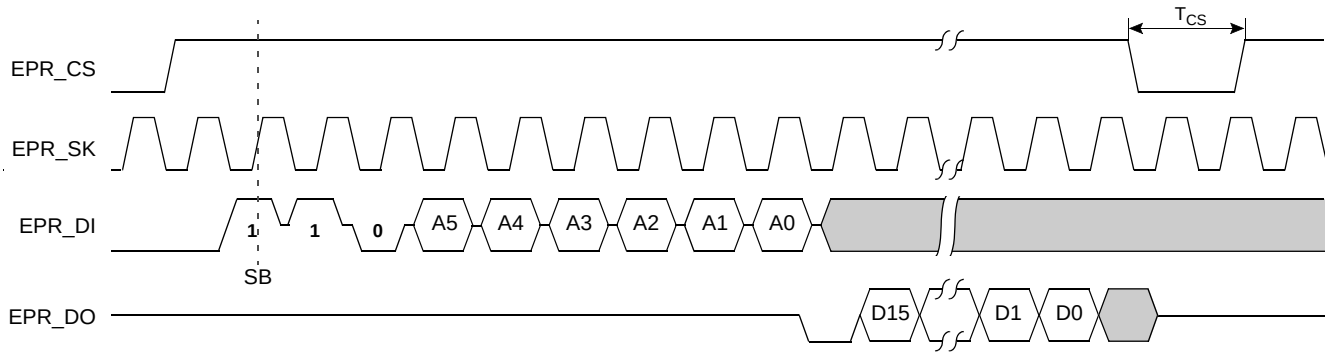


Figure 23–14 Initialize and Mode Register Timing Diagram

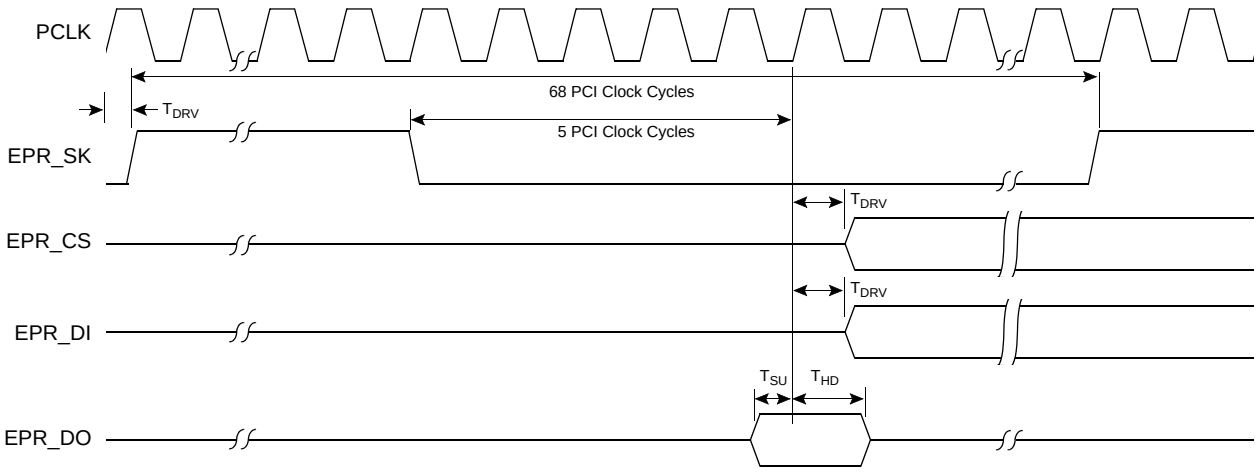


Figure 23–15 EEPROM AC Timing

23.7.2 EEPROM Signal I/O Timing

The EEPROM clock output signal, sourced by CN50XX, is derived from the PCI clock input (PCLK). PCLK can operate at 33MHz or 66MHz for PCI. The PCI system bus controller uses the power-on/reset state of the PCI bus signals PCIXCAP and M66EN to generate the PCI bus clock for all devices on the bus.

The CN50XX PCI logic is configured according to the timing of the PCLK signal. This automatic configuration of the PCI clock is reflected for the EEPROM clock. The EEPROM clock and associated timing is shown below.

ESK runs continually from the time the hard reset sequence terminates, until EEPROM data loading is complete (refer to Table 23–10). The EEPROM controller then stops ESK, in order to save power, since it will not be accessed again until another reset.

Table 23–10 EEPROM ESK Signal Timing

Mode of Operation	ESK Period, 'N' (PCLK Cycles)	EEPROM Clock Frequency (Max)
PCI 66 MHZ	68	0.98 MHz [1/(68 × 15ns)]
PCI 33 MHZ	68	0.49 MHz [1/(68 × 30ns)]

See Table 23–11 and Table 23–12. The ESK duty cycle is 50%. All EEPROM control signals are generated and sampled from the falling edge of ESK, plus a delay. This automatically configured delay (see Table 23–11) determines the number of PCI_PCLK cycles to wait, from the falling edge of ESK, before sampling and driving the EEPROM data/control pins. This delay helps facilitate adjustments of EEPROM control/data setup, hold and output delay times, allowing users to select various EEPROMs for different operating conditions. The control signal delay is based on PCI_PCLK.

Table 23–11 Delay Timing for Control Signals

Mode of Operation	EEPROM Delay (PCLK Cycles)	EEPROM Delay (ns)
PCI 66 MHZ	5	75ns (5×15ns)
PCI 33 MHZ	5	150ns (5×30ns)

Table 23–12 EEPROM Signal I/O Timing

Parameter	Description	Min	Max	Units
T_{DRV}	Drive time (w.r.t selected delay offset)	0	10	ns
T_{SU}	Setup time (w.r.t selected delay offset)	—	10	ns
T_{HD}	Hold time (w.r.t selected delay offset)	10	—	ns

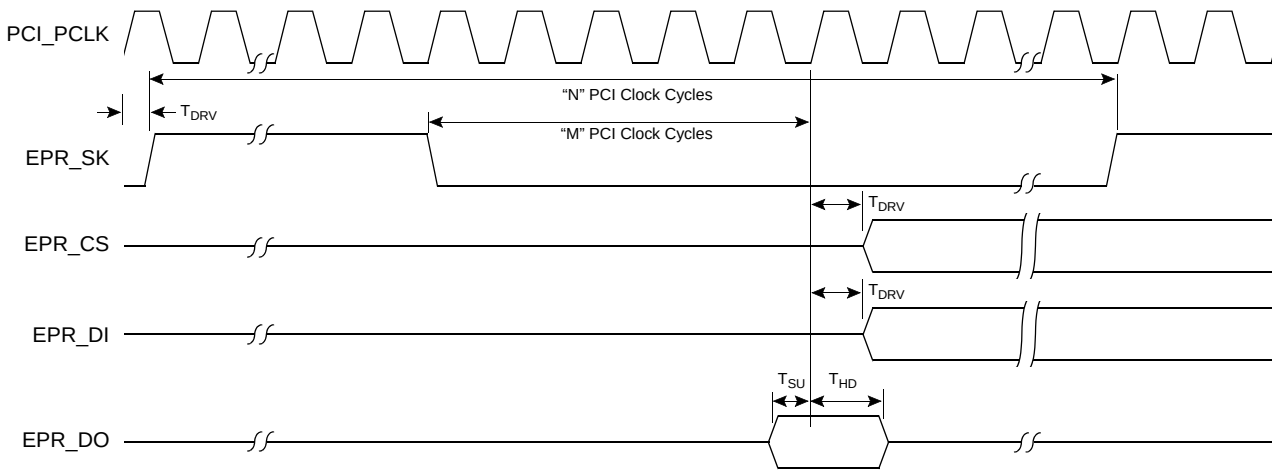


Figure 23–16 EEPROM Signal I/O Timing Diagram

NOTE: For designs not using the PCI interface, the PCI clock must be tied to ground. In this case, the EEPROM interface cannot be used.

23.8 Boot Bus Interface

For timing diagrams, refer to Section 12.4 in the Boot Bus chapter.

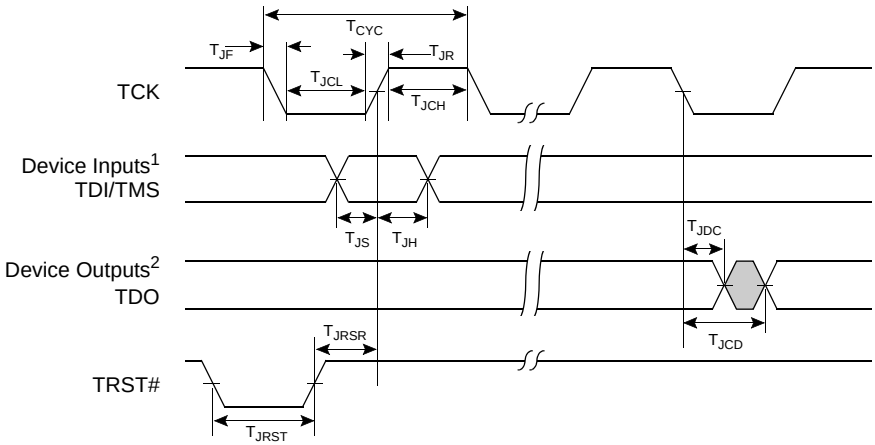
23.9 JTAG Interface

Table 23–13 lists CN50XX's JTAG signals I/O timing parameters.

Table 23–13 JTAG Signal I/O Timing Parameters

Parameter	Description	Min	Max	Units
T_{JCYC}	TCK cycle time	100	—	ns
T_{JCH}	TCK clock HIGH	40	—	ns
T_{JCL}	TCK clock LOW	40	—	ns
T_{JR} [1]	TCK rise time	—	5	ns
T_{JF} [1]	TCK fall time	—	5	ns
T_{JRST} [2]	TRST# assert time	50	—	ns
T_{JRSR}	TRST# recovery time	50		
T_{JCD} [3]	TCK to output data valid	5	50	ns
T_{JDC} [1, 3]	TCK to output data hold	5	—	ns
T_{JS} [4]	TCK to input setup time	50	—	ns
T_{JH} [5]	TCK to input hold time	50	—	ns

1. Guaranteed by design and characterization.
2. RST is an asynchronous level sensitive signal. Setup time for test purpose only.
3. Output = All device outputs including TDO.
4. Input = All device inputs include TDI and TMS.
5. Input = All device inputs include TDI and TMS.



1. Device inputs = All device inputs except TDI, TMS, and TRST#.
2. Device outputs = All device outputs except TDO.

Figure 23–17 JTAG Signal I/O Timing Diagram

23.10 MPI/SPI Interface

Table 23–13 lists CN50XX's MPI/SPI signals I/O timing parameters and Figure 23–18 shows the signal timing.

Table 23–14 MPI/SPI Signal I/O Timing Parameters

Parameter	Description	Min	Max	Units
T_{CYC}	MPI_CLK period	10	8191	ECLK cycles
T_{CYC}	MPI_CLK period	20	—	ns
T_{DV}	MPI_TX data valid	–1	1	ns
T_{ZD}	MPI_TX turn-on delay	–1	1	ns
T_{DZ}	MPI_TX turn-off delay	–1	1	ns
T_{IS}	MPI_RX/MPI_TX input setup	7	—	ns
T_{IH}	MPI_RX/MPI_TX input hold	0	—	ns

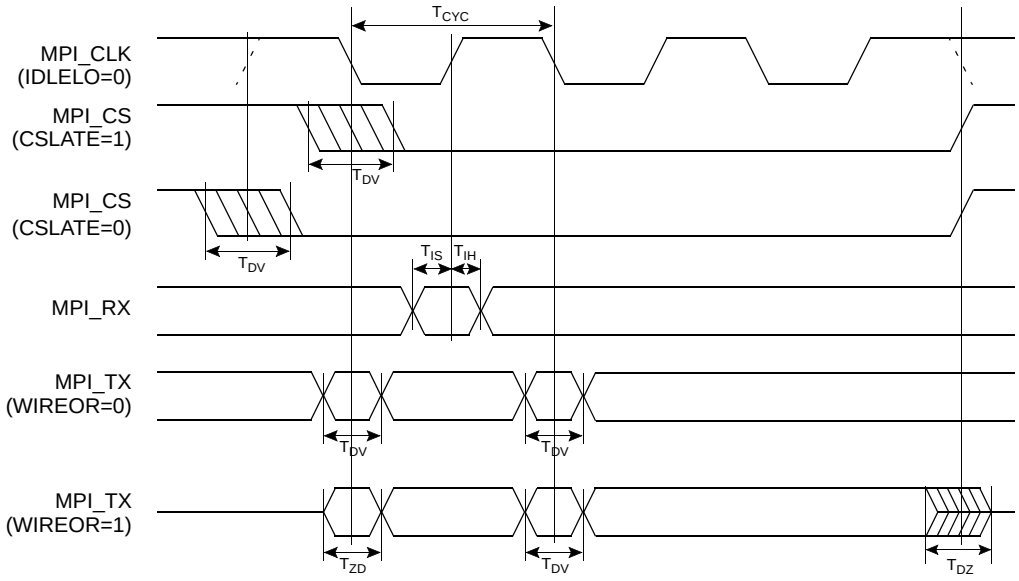


Figure 23–18 MPI/SPI Signal I/O Timing Diagram

23.11 TWSI Interface

Table 23–15 lists CN50XX's TWSI signals I/O timing parameters.

Table 23–15 TWSI Signal Parameters

Parameter	Description	Standard Mode		Fast Mode		Units
		Min	Max	Min	Max	
F_{SCL}	SCL clock frequency	0	100	0	400	kHz
$T_{HD;STA}$ [1]	Hold time (repeated) START condition	4.0	—	0.6	—	µs
T_{LOW}	Low period of the SCL clock	4.7	—	1.3	—	µs
T_{HIGN}	High period of the SCL clock	4.0	—	0.6	—	µs
$T_{SU;STA}$	Set-up time for a repeated START condition	4.7	—	0.6	—	µs
$T_{HD;DAT}$	Data hold time	0	—	0	—	µs
$T_{SU;DAT}$	Data set-up time	250	—	100	—	ns
T_R	Rise time of both SDA and SCL signals	—	1000	20 + 0.1Cb [2]	300	ns
T_F	Fall time of both SDA and SCL signals	—	300	20 + 0.1Cb [2]	300	ns
$T_{SU;STO}$	Set-up time for STOP condition	4.0	—	0.6	—	µs
T_{BUF}	Bus free time between a STOP and START condition	4.7	—	1.3	—	µs

1. After this period, the first clock pulse is generated.
2. Cb is the capacitance of one bus line in pF.

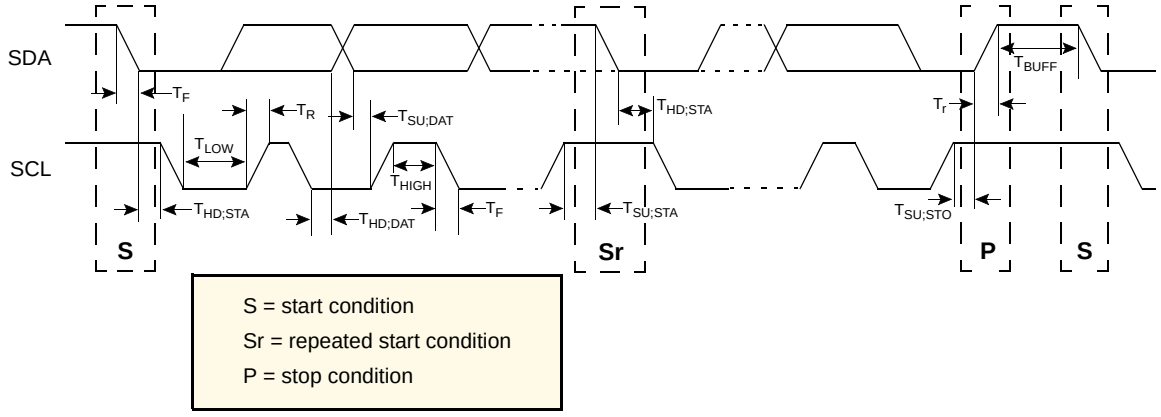


Figure 23–19 TWSI Signal Parameters

23.12 SMI/MDIO Interface

For timing diagrams, refer to Figures 18–2 and 18–3 in the SMI chapter.

Chapter 24

Mechanical Specifications

This chapter contains the following subjects:

Overview

This chapter contains the CN50XX package dimensions and ball grid array.

24.1 Ball Grid Array Package Diagram

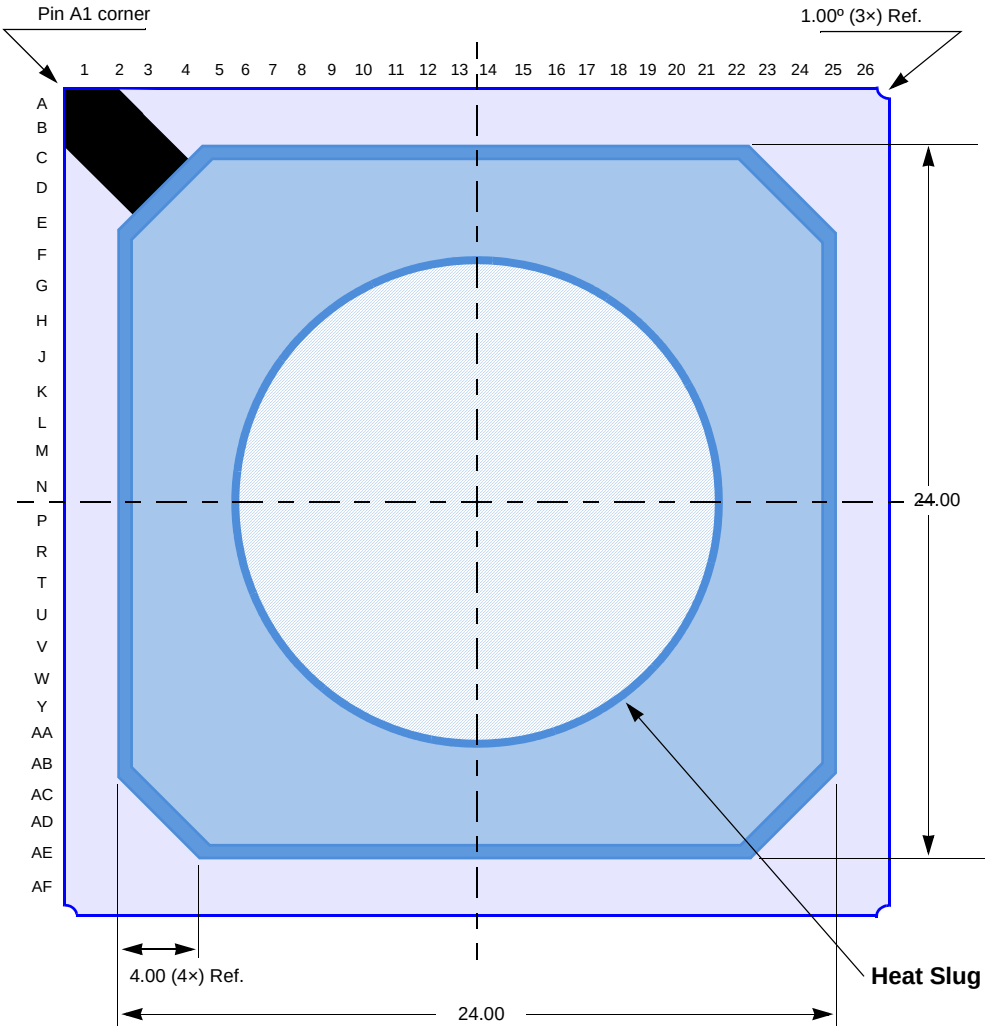


Figure 24–1 564L-HSBGA Package Diagram (Top View)

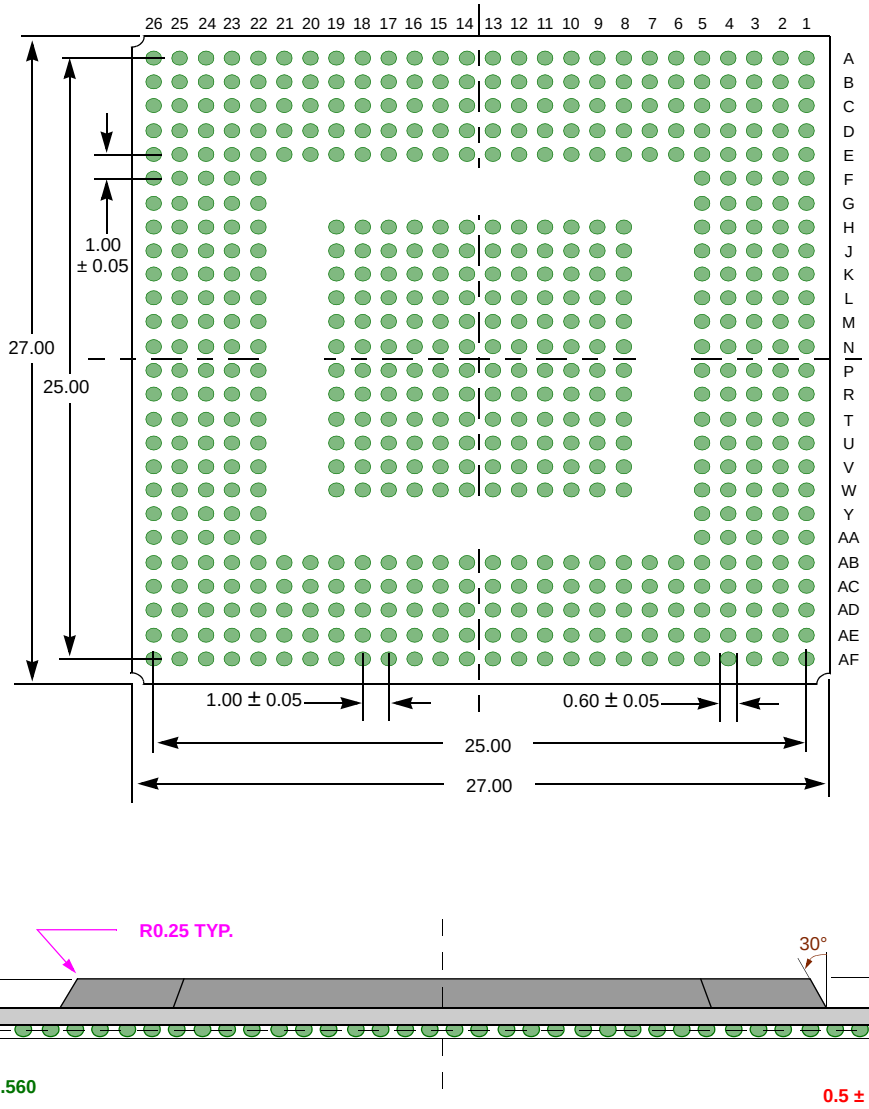


Figure 24–2 564-HSBGA Package Diagram (Bottom and Side Views)

24.2 Package Thermal Specifications

Table 24–1 lists CN50XX's thermal packaging specifications.

Table 24–1 Thermal Package Specification for CN50XX

Parameter	Description	Max	Units
θ_{JA0}	Thermal resistance – junction to ambient at 0 m/s or 0 LFM	13.0	°C/W
θ_{JA1}	Thermal resistance – junction to ambient at 1 m/s or 200 LFM	11.1	°C/W
θ_{JA2}	Thermal resistance – junction to ambient at 2 m/s or 400 LFM	10.2	°C/W
θ_{JC}	Thermal resistance – junction to case	4.8	°C/W
θ_{JB}	Thermal resistance – junction to board	7.3	°C/W

24.3 Package Thermal Management Requirements

NOTE: This device requires the use of a heat sink. Please read Section 24.5, "Heat Sink Selection for CN50XX-BG564," on page 797 for further details.

The management of thermal energy due to a microelectronic device's power dissipation is important to receive the best possible performance. The temperature at which a microelectronic device operates determines, among other things, the speed and reliability of the product. Therefore careful consideration of the factors affecting the device's operating temperature is recommended to achieve the best possible results.

The most important factors affecting device-operating temperature are power dissipation, air temperature, package construction, and cooling mechanisms. The combinations of these factors determine the temperature at which the product will operate.

24.4 Thermal Definitions

- **T_A: Ambient temperature (°C)**. Temperature of the ambient air around the device.

  $$T_A = T_J - P \times \theta_{JA}$$

  $$T_A = T_C - P \times (\theta_{JA} - \theta_{JC})$$

- **T_C: Case temperature (°C)**. Temperature of the case of the device package. The measurement is made by help of thermocouples placed on the warmest point of the package (in the middle of the upper cover).

  $$T_C = T_A + P \times (\theta_{JA} - \theta_{JC})$$

- **T_J: Junction temperature (°C)**. Average junction temperature of the die within the package.

  $$T_J = P \times \theta_{JA} + T_A$$

  $$T_J = P \times \theta_{JC} + T_C$$

- **P: Total power dissipation of the device (W)**. The sum of power dissipation from all device power supplies.
- θ_{JA}: Thermal resistance from the junction to the environment (°C/W)

  $$\theta_{JA} = \frac{T_J - T_A}{P}$$

  $$\theta_{JA} = \theta_{JC} + \theta_{CA}$$

- θ_{JC}: Thermal resistance from the junction to the case (°C/W)

  $$\theta_{JC} = \frac{T_J - T_C}{P}$$

- θ_{CA}: Thermal resistance from the surface of the cover to the environment (°C/W)

  $$\theta_{CA} = \frac{T_C - T_A}{P} \quad [1]$$

- θ_{SA}: Thermal resistance from the heat sink to the ambient air (°C/W)

  $$\theta_{SA} = \theta_{CA} - \theta_{CS}$$

- θ_{CS}: Thermal resistance of the gluing compound between the case and the heat sink (e.g. thermal compound) (°C/W)

24.5 Heat Sink Selection for CN50XX-BG564

To satisfy the T_J requirement with the given T_A, the θ_{JA} of the device package must be:

$$\theta_{JA} = \frac{T_J - T_A}{P}$$

Using CN50XX power consumption as an example, to meet the typical temperature requirements, the thermal resistance should be

$$\theta_{JA} = \frac{T_J - T_A}{P} = \frac{110 - 55}{4} = 13.75\ °C/ \quad \textbf{[1]}$$

For the BGA package, θ_{JA0} as specified in Table 24–1 is lower than what is calculated in Equation **[1]**. Without a cooling solution this is not going to satisfy the maximum junction temperature requirement. Using the simplified concept of thermal resistance, heat flows serially from the junction to the case then across the interface into the heat sink and is finally dissipated from the heat sink to the air stream.

Chapter 25

Signal Descriptions

This chapter provides descriptions of the OCTEON Plus CN50XX I/O signals. The signals are grouped into 14 groups, corresponding to the I/O interfaces, as shown in Table 25–1.

Table 25–1 CN50XX Interfaces

Interface	Number of Signals	Table
DDR memory interface signals	90	Table 25–3 on page 801
PCI interface signals	73	Table 25–4 on page 802
Packet Interface signals GMII interface signals RGMII interface signals MII interface signals	46	Table 25–5 on page 803 Table 25–6 on page 804 Table 25–8 on page 805 Table 25–7 on page 804
GPIO interface signals PCM/TDM interface signals MPI/SPI interface signals	24	Table 25–9 on page 806 Table 25–10 on page 806 Table 25–11 on page 806
Boot signals	40	Table 25–12 on page 807
MDIO interface signals	2	Table 25–13 on page 807
TWSI signals	2	Table 25–14 on page 807
Clock signals	5	Table 25–15 on page 808
UART interface signals	8	Table 25–16 on page 808
EEPROM interface signals	4	Table 25–17 on page 809
JTAG interface signals	7	Table 25–18 on page 809
USB interface signals	5	Table 25–19 on page 809
Miscellaneous signals	4	Table 25–20 on page 810
Power/ground/no connect signals	254	Table 25–21 on page 810
Total number of signals	**564**	

Overview

The CN50XX I/O signals are grouped by interface and described in tables with the following column headings:

Pin Name: Provides the name of the field or signal.

Reset: Indicates the value of the field or signal at reset.

If Not Used: Indicates what should be done with the balls if the field or signal is not used.

Dir: Indicates the direction of the field or signal: input, output, or input/output.

Type: Indicates what type of signal. Refer to Table 25–2

Up/Down: Indicates the direction of CN50XX's weak internal pullup/pulldown for the signal. If blank, CN50XX does not have a weak internal pullup/pulldown on the signal.

Description: Describes the function of the signal or field.

Signal Ball Types

Table 25–2 describes the connections of the I/O types used in the Type column of the signals list:

Table 25–2 CN50XX Pin Types

I/O Type	Description
ANALOG_D	1.8/2.5V tolerant Analog Input
ANALOG_P	3.3V tolerant Analog Input
ANALOG_R	1.8V tolerant Analog Input
CMOS33B	3.3V I/O
CMOS33I	3.3V Input
CMOS25I	2.5V Input
CMOS18I	1.8V Input
CMOS33I/CMOS25I	3.3V/2.5V Input
LVDSI	LVDS Input
LVDSI/CMOS33I	LVDS Input/3.3V input
LVDSI/CMOS25I	LVDS Input/2.5V input
LVDSO	LVDS Output
LVDSO/CMOS33O	LVDS Output/3.3V Output
LVDSO/CMOS25O	LVDS Output/2.5V Output
NTDIFFI	3.3V Differential Input
SSTL18	1.8V SSTL I/O
SSTL18R	1.8V SSTL I/O
SUPPLY	Power Supply

25.1 DRAM Interface Signals

The DRAM interface communicates with up to two DDR2 DIMMs across a 32-bit data bus. Some of the interface signals may be unused:

- The DRAM interface can be in 32-bit or 16-bit mode (LMC_CTL[MODE32b]).
- The DRAM interface can be used with or without ECC. To enable/disable ECC mode, refer to LMC_MEM_CFG0[ECC_ENA].

Table 25–3 describes the signals associated with DRAM interface.

Table 25–3 DDR DRAM Interface Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
DDR_DIMM<0>_CS0_L DDR_DIMM<0>_CS1_L DDR_DIMM<1>_CS0_L DDR_DIMM<1>_CS1_L		NC	O	SSTL18		DDR2 DIMM chip select, where <1:0> represents: <0> = enable DIMM0, bits <31:0> <1> = enable DIMM1, bits <31:0> CS0 indicates DIMM rank 0 and CS1 indicates DIMM rank 1. Example: DDR_DIMM<0>_CS0_L enables DIMM0, rank 0. DDR_DIMM<1>_CS1_L enables DIMM1, rank 1.
DDR_CAS_L		NC	O	SSTL18		DDR2 column address select.
DDR_RAS_L		NC	O	SSTL18		DDR2 row address select
DDR_A<14:0>		NC	O	SSTL18		DDR2 address lines.
DDR_BA<2:0>		NC	O	SSTL18		DDR2 bank address lines.
DDR_DQ<31:0>		NC	I/O	SSTL18		DDR2 data bits. **Note:** For data bits used in 16-bit mode, refer to LMC_CTL1[DATA_LAYOUT].
DDR_CB<1:0> DDR_CB<3:2>[1]		NC	I/O	SSTL18		DDR2 ECC bits. **Note:** For ECC bits used in 16-bit mode, refer to LMC_CTL1[DATA_LAYOUT].
DDR_CBS_0_P DDR_CBS_0_N		NC	I/O	SSTL18		Differential data strobe for ECC bits (positive and negative). **Note:** For data strobes used in 16-bit mode, refer to LMC_CTL1[DATA_LAYOUT].
DDR_CK_<3:0>_P DDR_CK_<3:0>_N		NC	O	SSTL18		DDR2 differential address clocks (positive and negative).
DDR_CKE		NC	O	SSTL18		DDR2 clock enable.
DDR_DQS_<3:0>_P DDR_DQS_<3:0>_N		NC	I/O	SSTL18		Differential data strobes (positive and negative). DDR_DQS_0 for byte 0,..., DDR_DQS_3 for byte 3. **Note:** For data strobes used in 16-bit mode, refer to LMC_CTL1[DATA_LAYOUT].
DDR_VREF		GND	AI	ANALOG_D		DDR2 Vref pin, hook to VDD_DDR/2
DDR_WE_L		NC	O	SSTL18		DDR2 write enable
DDR_PLL_VDD33		GND	I	SUPPLY		DDR2 PLL connection to a low-noise 3.3V supply. This pin also powers the core PLL.
DDR_REF_CLK_P DDR_REF_CLK_N		GND	I	NTDIFFI		DDR2 clocks are generated internally in the CN50XX. These pins should be NC on new designs using this chip. **Note:** CN3010/CN3020 required external DDR2 reference clock that must be left unconnected on the CN50XX.
DDR_DIMM<0>_ODT<1:0> DDR_DIMM<1>_ODT<1:0>		NC	O	SSTL18		DDR2 DIMM on-die termination enable, where DDR_DIMM*n*_ODT*m* enables ODT for DIMM*n*, rank *m*. Example: DDR_DIMM1_ODT0 enables ODT for DIMM1 rank 0.
DDR_COMP_DN		NC	O	SSTL18		DDR2 down compensation. Pull up to program the low drive strength
DDR_COMP_UP		NC	O	SSTL18		DDR up compensation. Pull down to program the high drive strength
Total = 90						

1. Not used in 16-bit mode.

25.2 PCI Interface Signals

Table 25–4 describes the signals associated with the PCI interface.

Table 25–4 PCI Interface Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
PCI_AD<31:0>		NC	I/O	CMOS33B		Lower PCI address lines
PCI_BOOT		NC	I	CMOS33B	DN	PCI boot select. 1 = boot from PCI bus, 0 = boot from boot bus.
PCI_CBE_<3:0>_L		NC	I/O	CMOS33B		PCI command/byte enables
PCI_CLK_OUT<2:0>		NC	O	CMOS33B		PCI out clock for device use in host mode (three copies). Enabled by DIAG_CLKOUT_ENABLE.
PCI_COMP_DN		NC	O	CMOS33BD		PCI down compensation. Pull up to program the low drive strength.
PCI_COMP_UP		NC	O	CMOS33BU		PCI up compensation. Pull down to program the high drive strength.
PCI_DEV_GNT_<2:0>_L		NC	O	CMOS33B		PCI device grants in host mode.
PCI_DEV_GNT_3_L/REQ_L		NC	I	CMOS33B		PCI device grant in host mode/ PCI request in target mode.
PCI_DEV_REQ_<2:0>_L		NC	I	CMOS33B	UP	PCI device requests in host mode.
PCI_DEV_REQ_3_L/GNT_L		NC	O	CMOS33B	UP	PCI device requests in host mode/PCI grant in target mode.
PCI_DEVSEL_L		NC	I/O	CMOS33B		PCI device select
PCI_DLL_VDD33		GND	I	SUPPLY		PCI connection to a low-noise 3.3V supply.
PCI_ENABLE		GND	I	CMOS33B	UP	Enables the PCI interface.
PCI_FRAME_L		NC	I/O	CMOS33B		PCI Frame
PCI_HOST_MODE		PU	I	CMOS33B	DN	Enable host mode. 1 = host mode, 0= target mode
PCI_IDSEL		GND	I	CMOS33B		PCI IDSEL in target mode. Connect to GND in host mode.
PCI_INTA_L[1]		PU	I/O (OD)	CMOS33B		Output interrupt request in target mode. INTA input in host mode.
PCI_INTB_L[1]		NC	I	CMOS33B	UP	Input interrupt request in host mode.
PCI_INTC_L[1]		NC	I	CMOS33B	UP	Input interrupt request in host mode.
PCI_INTD_L[1]		NC	I	CMOS33B	UP	Input interrupt request in host mode.
PCI_IRDY_L		NC	I/O	CMOS33B		PCI IRDY
PCI_LOCK_L		NC	I	CMOS33B	UP	PCI lock.
PCI_M66EN		NC	I	CMOS33B	DN	PCI M66 enable. 0 (low) = 33MHz enabled, 1 (high) = 66MHz enabled.
PCI_PAR		NC	I/O	CMOS33B		PCI parity bit for PCI_AD<31:0>.
PCI_PCI100		NC	I	CMOS33B	DN	Reserved. Should be NC.
PCI_PCIXCAP		GND	I	ANALOG_P		Reserved. Should be tied to ground.
PCI_PCLK		GND	I	CMOS33B		Input PCI clock signal (33 and 66 MHz)
PCI_PERR_L		NC	I/O	CMOS33B		PCI parity error.
PCI_REF_CLKIN		GND	I	CMOS33B		133-MHz reference input clock in host mode, used by CN50XX to produces PCI_CLK_OUT_*. When PCI_REF_CLKIN is 133 MHz, CN50XX produces either 66- or 33-MHz PCI_CLK_OUT_*, depending on the M66EN input pin, as per the PCI specification. In device mode (i.e. when the pin PCI_HOST_MODE=0), PCI_REF_CLKIN can be grounded.
PCI_RST_L		GND	I/O	CMOS33B		PCI reset output in host mode. Main chip reset in target mode. Minimum of 1000 core-clock cycles.
PCI_SERR_L		NC	I/O	CMOS33B		PCI system error.

Table 25–4 PCI Interface Signals (Continued)

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
PCI_STOP_L		NC	I/O	CMOS33B		PCI stop.
PCI_TRDY_L		NC	I/O	CMOS33B		PCI TRDY
Total = 73						

1. The primary purpose of the PCI_INT*_L pins is to provide interrupt inputs when CN50XX is a PCI host (PCI_HOST_MODE = 1) and to provide a PCI interrupt output pin (PCI_INTA_L) when CN50XX is a PCI device.

   When CN50XX is not a PCI host (PCI_HOST_MODE = 0), PCI_INTA_L is driven by CN50XX to request interrupts of the remote host. Since PCI_INT{B,C,D}_L are not used by the PCI-interface logic, they are available to use as input interrupt-request pins from other sources.

   When the PCI interface is not connected, the PCI_INT*_L pins can be used as general-purpose input interrupt-request pins if the following requirements are met:

   PCI_HOST_MODE must be set to 1.

   PCI_ENABLE must be asserted.

   PCI_PCLK must be supplied with a clock signal.

   PCI_REF_CLKIN can be grounded.

   PCI_DLL_VDD33 must be supplied with a voltage signal.

   Software should keep CIU_SOFT_PRST[SOFT_RESET] set so CN50XX's PCI logic remains in reset.

25.3 Packet Interface Signals

The CN50XX packet interface can be configured for the following formats:
- Gigabit Media Independent (GMII)
- Reduced Gigabit Media Independent (RGMII)
- Media Independent (MII)

The CN50XX packet interface can be configured with the following ports:
- Three RGMII ports, or
- One GMII port and either one RGMII port or one MII port, or
- Two MII ports, or
- One MII port and either one or two RGMII ports
- disabled.

The interface has four signal configuration pins (needed with all formats) and 42 interface pins that can be described as either GMII, RGMII, or MII signals.
- the GMII interface uses 24 of the 42 interface pins
- the RGMII interface uses 36 of the 42 interface pins.
- the MII interface uses 32 of the 42 interface pins.

The interface signals are described in the GMII, RGMII, and MII subsections.

Table 25–5 Packet Interface Compensation Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
RGM_COMP_DN		NC	I	CMOS25ID		Down compensation. Pull up to program the low drive strength
RGM_COMP_UP		NC	I	CMOS25IU		Up compensation. Pull down to program the high drive strength
GMI_REF_CLK_P			I			Differential reference clock.
GMI_REF_CLK_N			I			Differential reference clock.
Total = 4						

25.3.1 GMII Interface Signals

The packet interface can be configured for a GMII bus. Table 25–6 describes the signals associated with the system packet interface.

Table 25–6 GMII Interface Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
GMI_RXD<7:0>		GND	I	LVDSI/CMOS25I		GMII RX data.
GMI_RXERR		GND	I	LVDSI/CMOS25I		GMII RX error.
GMI_COL		GND	I	LVDSI/CMOS25I		GMII collision.
GMI_DV		GND	I	LVDSI/CMOS25I		GMII data valid.
GMI_CRS		GND	I	LVDSI/CMOS25I		GMII carrier sense.
GMI_RXCLK			I			GMII RX clock.
GMI_TXD<7:0>		NC	O	LVDSO/CMOS25O		GMII TX data
GMI_TXERR		NC	O	LVDSO/CMOS25O		GMII TX error.
GMI_TXEN		NC	O	LVDSO/CMOS25O		GMII TX enable.
GMI_GTXCLK		NC	O	LVDSO/CMOS25O		GMII TX clock.
Total = 24						

25.3.2 MII Interface Signals

The packet interface can be configured for one or two MII buses. Table 25–7 describes the signals associated with the system packet interface.

Table 25–7 MII Interface Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
MI0_RXD[3:0]			I	LVDSI/CMOS25I		MII0 RX data.
MI0_RXCLK			I	LVDSI/CMOS25I		MII0 RX clock.
MI0_RXERR			I	LVDSI/CMOS25I		MII0 RX error.
MI0_RXDV						MII0 RX data valid.
MI0_COL						MII0 collision.
MI0_CRS						MII0 carrier sense.
MI0_TXD[3:0]			O	LVDSO/CMOS25O		MII0 TX data.
MI0_TXCLK			I	LVDSO/CMOS25O		MII0 TX clock.
MI0_TXERR			O	LVDSO/CMOS25O		MII0 TX error.
MI0_TXEN			O	LVDSO/CMOS25O		MII0 TX enable.
MI1_RXD[3:0]			I	LVDSI/CMOS25I		MII1 RX data.
MI1_RXCLK			I	LVDSI/CMOS25I		MII1 RX clock.
MI1_RXERR			I	LVDSI/CMOS25I		MII1 RX error.
MI1_RXDV						MII1 RX data valid.
MI1_COL						MI1 collision.
MI1_CRS						MI1 carrier sense.
MI1_TXD[3:0]			O	LVDSO/CMOS25O		MII1 TX data.
MI1_TXCLK			I	LVDSO/CMOS25O		MII1 TX clock.
MI1_TXERR			O	LVDSO/CMOS25O		MII1 TX error.
MI1_TXEN			O	LVDSO/CMOS25O		MII1 TX enable.
Total = 32						

25.3.3 RGMII Interface Signals

The packet interface can be configured for three RGMII ports. Table 25–8 describes the signals associated with the RGMII interface.

Table 25–8 RGMII Interface Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
RGMII Interface 0						
RGM0_RXD[3:0]		GND	I	LVDSI/CMOS25I		RGMII Port 0 RX data.
RGM0_RXCTL		GND	I	LVDSI/CMOS25I		RGMII Port 0 RX control.
RGM0_RXC		GND	I	LVDSI/CMOS25I		RGMII Port 0 RX clock.
RGM0_TXD[3:0]		NC	O	LVDSO/CMOS25O		RGMII Port 0 TX data.
RGM0_TXCTL		NC	O	LVDSO/CMOS25O		RGMII Port 0 TX control.
RGM0_TXC		NC	O	LVDSO/CMOS25O		RGMII Port 0 TX clock.
RGM1_RXD[3:0]		GND	I	LVDSI/CMOS25I		RGMII Port 1 RX data.
RGM1_RXCTL		GND	I	LVDSI/CMOS25I		RGMII Port 1 RX control.
RGM1_RXC		GND	I	LVDSI/CMOS25I		RGMII Port 1 RX clock.
RGM1_TXD[3:0]		NC	O	LVDSO/CMOS25O		RGMII Port 1 TX data.
RGM1_TXCTL		NC	O	LVDSO/CMOS25O		RGMII Port 1 TX control.
RGM1_TXC		NC	O	LVDSO/CMOS25O		RGMII Port 1 TX clock.
RGM2_RXD[3:0]		GND	I	LVDSI/CMOS25I		RGMII Port 2 RX data.
RGM2_RXCTL		GND	I	LVDSI/CMOS25I		RGMII Port 2 RX control.
RGM2_RXC		GND	I	LVDSI/CMOS25I		RGMII Port 2 RX clock.
RGM2_TXD[3:0]		NC	O	LVDSO/CMOS25O		RGMII Port 2 TX data.
RGM2_TXCTL		NC	O	LVDSO/CMOS25O		RGMII Port 2 TX control.
RGM2_TXC		NC	O	LVDSO/CMOS25O		RGMII Port 2 TX clock.
Total = 36						

25.4 General Purpose I/O (GPIO) Interface Signals

The GPIO interface has 24 signals available for I/O. Some of these signals are shared with the boot bus, the MPI/SPI interface, and the PCM/TDM interface:

- bits<7:0> are strictly GPIO bits
- bits<11:8> can be used as boot-bus chip-select signals, or as GPIO bits
- bits <19:12> can be used with the PCM/TDM interface, or as GPIO bits
- bits <23:20> can be used with the MPI/SPI interface, or as GPIO bits

Table 25–9 describes the GPIO interface signals.

Table 25–9 GPIO Interface Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
GPI_GPIO_<11>/BOOT_CE_7_L		NC	I/O	CMOS33B	DN	General purpose I/O/Boot-bus chip enable
GPI_GPIO_<10>/BOOT_CE_6_L		NC	I/O	CMOS33B	DN	General purpose I/O/Boot-bus chip enable
GPI_GPIO_<9>/BOOT_CE_5_L		NC	I/O	CMOS33B	DN	General purpose I/O/Boot-bus chip enable
GPI_GPIO_<8>/BOOT_CE_4_L		NC	I/O	CMOS33B	DN	General purpose I/O/Boot-bus chip enable
GPI_GPIO_<7:0>		NC	I/O	CMOS33B	DN	General purpose I/O.
Total = 12						

25.4.1 PCM/TDM Interface Signals

Table 25–10 describes the signals available to the PCM/TDM interface.

Table 25–10 PCM/TDM Interface Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
GPIO_19/PCM_BCLK0		NC	I/O	CMOS33B		General purpose I/O/PCM bit clock for clock pair 0
GPIO_18/PCM_FSYNC0		NC	I/O	CMOS33B		General purpose I/O/PCM sync clock for clock pair 0
GPIO_<17:16>/PCM_DATA<1:0>		NC	I/O	CMOS33B		General purpose I/O/TDM engine data bits
GPIO_15/PCM_BCLK1		NC	I/O	CMOS33B		General purpose I/O/PCM bit clock for clock pair 1
GPIO_14/PCM_FSYNC1		NC	I/O	CMOS33B		General purpose I/O/PCM sync clock for clock pair 1
GPIO_<13:12>/PCM_DATA<3:2>		NC	I/O	CMOS33B		General purpose I/O/TDM engine data bits
Total = 8						

25.4.2 MPI/SPI Signals

Table 25–20 describes the signals available to the MPI/SPI interface.

Table 25–11 MPI/SPI Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
GPIO20/MPI_RX		NC	I	CMOS33B		General purpose I/O/MPI/SPI receive data
GPIO21/MPI_TX		NC	O	CMOS33B		General purpose I/O/MPI/SPI transmit data
GPIO22/MPI_CS		NC	O	CMOS33B		General purpose I/O/MPI/SPI chip select
GPIO23/MPI_CLK		NC	O	CMOS33B		General purpose I/O/MPI/SPI clock
Total = 4						

25.5 Boot-Bus Signals

Table 25–12 describes the signals associated with the boot bus.

Table 25–12 Boot-Bus Signals[1]

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
BOOT_AD<31:0>		NC	I/O	CMOS33B		Boot-bus address/data.
BOOT_CE_<3:0>_L		NC	O	CMOS33B		Boot-bus chip enable/Hook to boot device (bit <0> only).
BOOT_ALE		NC	O	CMOS33B	DN	Boot-bus address latch enable.
BOOT_OE_L		NC	O	CMOS33B		Boot-bus output enable.
BOOT_WAIT_L		NC	I	CMOS33B	UP	Compact flash wait/busy signal.
BOOT_WE_L		NC	O	CMOS33B		Boot-bus write enable
Total = 40						

1. If needed, there are four additional chip-enable signals that can be taken from the GPIO signals. See Table 25–9.

25.6 MDIO Interface Signals

Table 25–13 describes the signals associated with the MDIO interface.

Table 25–13 MDIO Interface Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
MDI_MDC		NC	O	CMOS33B		RGMII MDIO clock.
MDI_MDIO		NC	I/O	CMOS33B		RGMII MDIO serial data.
Total = 2						

25.7 Two-Wire Serial Interface (TWSI) Signals

Table 25–14 describes the TWSI signals.

Table 25–14 TWSI Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
TWS_SCL		PU	I/O (OD)	CMOS33B		TWSI serial clock.
TWS_SDA		PU	I/O (OD)	CMOS33B		TWSI serial data.
Total = 2						

25.8 Clock Signals

Table 25–15 describes the CN50XX clock signals.

Table 25–15 Clock Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
PLL_MUL_<2:0>		NC	I	CMOS33B		Core-clock PLL frequency selectors. Multiplier times PLL_REF_CLK: 000 = ×8 (400MHz) 100 = ×14 (700 MHz) 001 = Reserved 101 = Reserved 010 = ×10 (500 MHz) 110 = ×6 (300 MHz) 011 = ×12 (600 MHz) 111 = ×7 (350 MHz) These signals are internally pulled down. To specify a 0, leave unconnected or externally pull to ground through a 100Ω resistor. To specify a 1, externally pull up to VDD33 through a 10KΩ resistor.
PLL_VDD33			I	SUPPLY		Core PLL connection to a low-noise 3.3V supply. This pin is unused and is supplied just for compatibility with the CN30XX chip. The core PLL is powered by DDR_PLL_VDD33.
PLL_REF_CLK			I			Core input clock signal, which **must** be 50MHz.
Total = 5						

25.9 UART Interface Signals

Table 25–16 describes the UART interface signals.

Table 25–16 UART Interface Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
Universal Asynchronous Receive/Transmit Interface (UART) 0						
UART0_CTS_L		NC	I	CMOS33B	UP	UART0 clear to send.
UART0_RTS_L		NC	O	CMOS33B		UART0 request to send.
UART0_SIN		NC	I	CMOS33B	DN	UART0 serial data in.
UART0_SOUT		NC	O	CMOS33B		UART0 serial data out.
Universal Asynchronous Receive/Transmit Interface (UART) 1						
UART1_CTS_L		NC	I	CMOS33B	UP	UART1 clear to send.
UART1_RTS_L		NC	O	CMOS33B		UART1 request to send.
UART1_SIN		NC	I	CMOS33B	DN	UART1 serial data in.
UART1_SOUT		NC	O	CMOS33B		UART1 serial data out.
Total = 8						

25.10 EEPROM Signals

Table 25–20 describes the EEPROM interface signals.

Table 25–17 EEPROM Interface Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
EPR_CS		NC	O	CMOS33B		PCI configuration EEPROM chip select.
EPR_DI		NC	O	CMOS33B		PCI configuration EEPROM data in to EEPROM.
EPR_DO		NC	I	CMOS33B	UP	PCI configuration EEPROM data out from EEPROM.
EPR_SK		NC	O	CMOS33B		PCI configuration EEPROM chip clock.
Total = 4						

25.11 eJTAG/JTAG Signals

Table 25–18 describes the eJTAG/JTAG signals.

Table 25–18 eJTAG/JTAG Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
EJTG_TDO		NC	O	CMOS33B		E- JTAG data out.
EJTG_TRST_L		NC	I	CMOS33B	DN	E- JTAG reset.
JTG_TCK		NC	I	CMOS33B	UP	JTAG clock.
JTG_TDI		NC	I	CMOS33B	UP	JTAG test data in.
JTG_TDO		NC	O	CMOS33B		JTAG test data out.
JTG_TMS		NC	I	CMOS33B	UP	JTAG TMS.
JTG_TRST_L		NC	I	CMOS33B	DN	JTAG reset.
Total = 7						

25.12 USB Signals

Table 25–19 describes the USB signals.

Table 25–19 USB Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
USB_DP		NC	I/O	USB Analog I/O	N/A	D+ analog signal from the USB cable.
USB_DM		NC	I/O	USB Analog I/O	N/A	D– analog signal from the USB cable.
USB_REXT		3.3V	I/O	USB Analog I/O	N/A	External resistor connect (43.2 Ω ±1%). If there is no intent to use the USB interface, tie this pin to any 3.3V supply available on the board.
USB_XO		NC	I	USB Analog I/O	N/A	Crystal oscillator xo pin. For usage information, refer to Section 21.7.4. Either a crystal or a board clock can be used for this pin. If a board clock is used, the signal swing must be 2.5V ± 7%.
USB_XI		NC	I	USB Analog I/O	N/A	Crystal oscillator xi pin. For usage information, refer to Section 21.7.4. Either a crystal or a board clock can be used. If a board clock is used, this signal should be tied to ground.
Total = 5						

25.13 Miscellaneous Signals

Table 25–20 describes the miscellaneous signals.

Table 25–20 Miscellaneous Signals

Pin Name	Reset	If Not Used	Dir	Type	Up/ Down	Description
CHIP_RESET_L			I	CMOS33B		Global chip reset in host mode. Unused in target mode. Must assert for a minimum of 1000 core-clock cycles.
PLL_DCOK			I	CMOS33I		Power OK. Assert this signal after all the power supplies and clocks are stable.
DIAG_CLKOUT			O	CMOS33B		Diagnostic clock out, running at core-clock speed
DIAG_CLKOUT_ENABLE			I	CMOS33B		Enable the DIAG_CLKOUT and PCI_CLK_OUT<2:0> signals.
Total = 4						

25.14 Power/Ground/No Connect Signals

Table 25–21 lists the power/ground/reserved/no connect signals.

Table 25–21 Power/Ground/No Connect Signals

Pin Name	Numbers	If Not Used	Dir	Type	Up/ Down	Description
GND	163			Supply		Ground supply.
VDD	28			Supply		Core supply voltage
VDD18_DDR	20			Supply		DDR supply voltage
VDD25_RGM	6			Supply		Packet interface supply voltage (2.5V)
VDD33	30			Supply		PCI supply voltage (3.3V)
USB_VDDA33	1	3.3V		Supply		USB supply voltage (3.3V). If there is no intent to use the USB interface, tie this pin to any 3.3V supply available on the board.
USB_GND	1			Supply		USB ground supply
RSVD	5			—	N/A	Reserved, not connected.
NC	0			—	N/A	Not connected.
Total =	254					

Chapter 26

Ball Assignments

This chapter contains the following subjects:

Overview

This chapter provides the signal/ball assignments (pinouts) for the CN50XX Secure Communications Processor.

26.1 CN50XX Ball Grid Array

Figure 26–1 shows the CN50XX ball grid array.

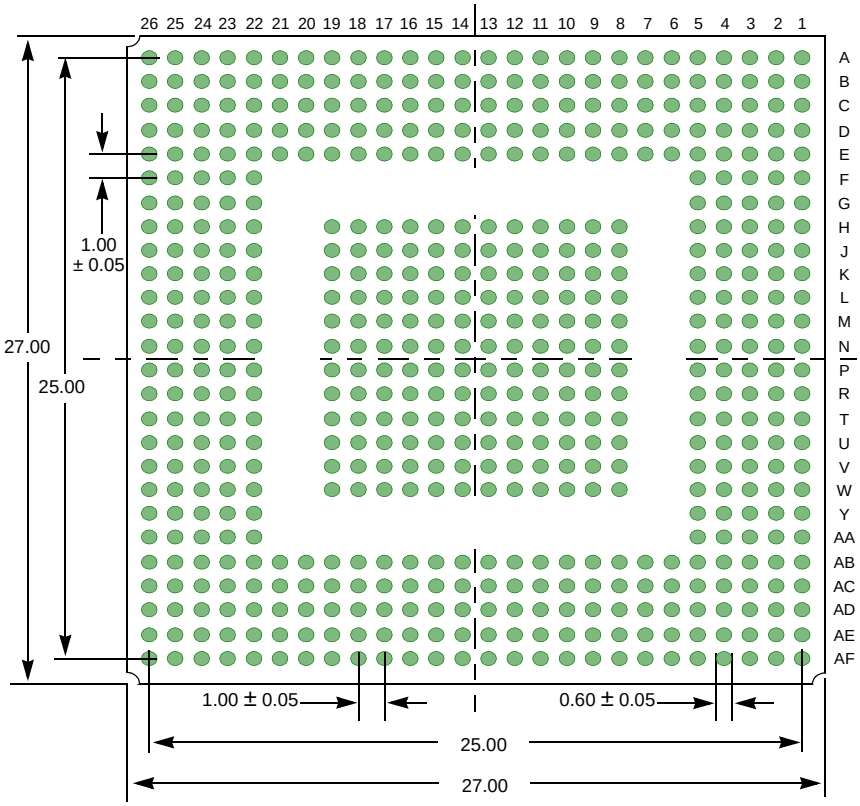


Figure 26–1 HSBGA Ball Assignment Diagram (Bottom View)

26.2 CN50XX Signal Mapping

Figure 26–2 shows the CN50XX signal map.

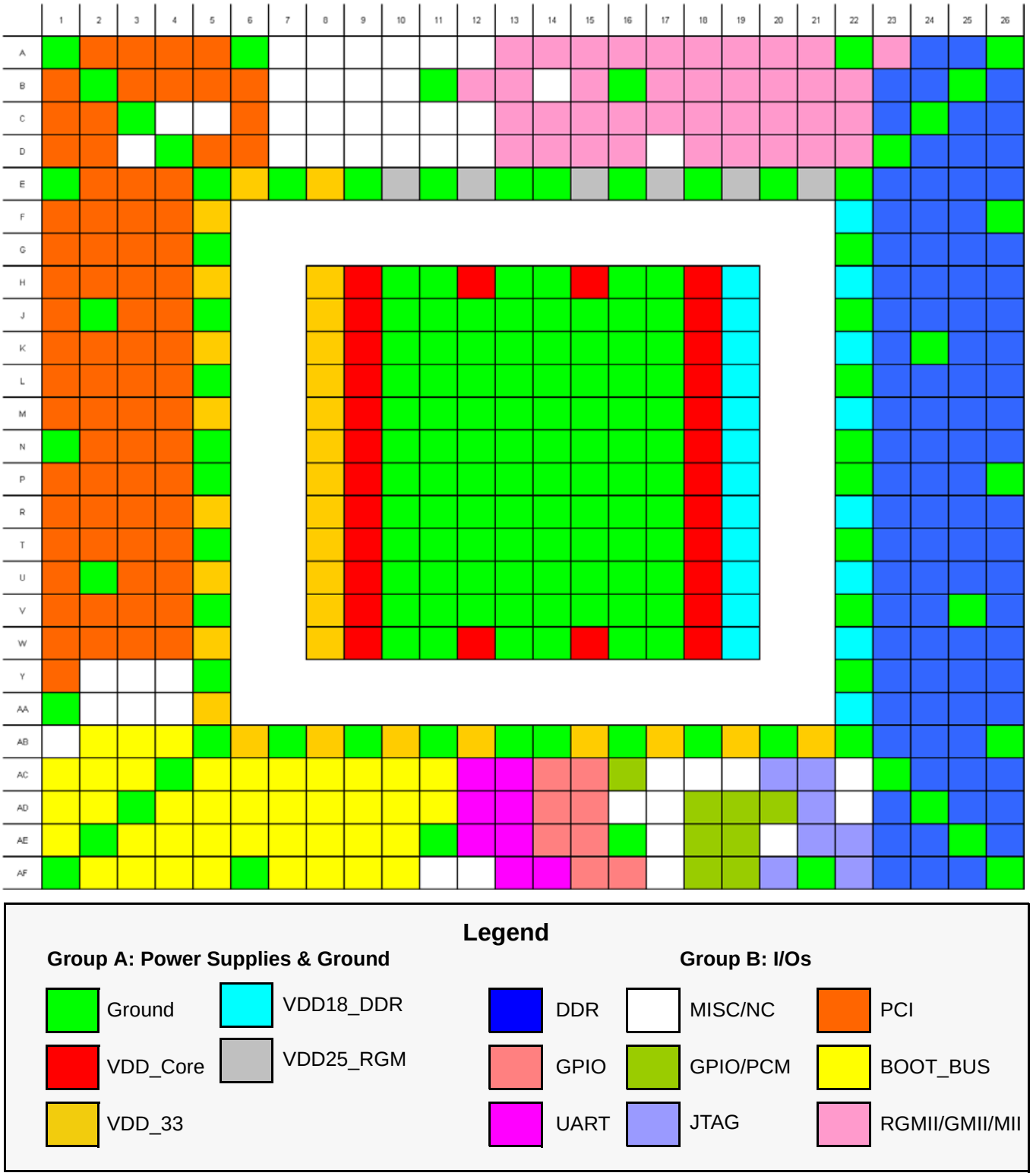


Figure 26–2 BGA Signal Map (Top View)

26.3 CN50XX Signals Sorted in Alphabetical Order

Pin Name	Ball
BOOT_AD<0>	AD9
BOOT_AD<1>	AF8
BOOT_AD<10>	AD6
BOOT_AD<11>	AE6
BOOT_AD<12>	AF5
BOOT_AD<13>	AC6
BOOT_AD<14>	AE5
BOOT_AD<15>	AD5
BOOT_AD<16>	AF4
BOOT_AD<17>	AC5
BOOT_AD<18>	AE4
BOOT_AD<19>	AD4
BOOT_AD<2>	AC9
BOOT_AD<20>	AF3
BOOT_AD<21>	AE3
BOOT_AD<22>	AF2
BOOT_AD<23>	AE1
BOOT_AD<24>	AD2
BOOT_AD<25>	AD1
BOOT_AD<26>	AB4
BOOT_AD<27>	AC3
BOOT_AD<28>	AC2
BOOT_AD<29>	AC1
BOOT_AD<3>	AE8
BOOT_AD<30>	AB3
BOOT_AD<31>	AB2
BOOT_AD<4>	AD8
BOOT_AD<5>	AC8
BOOT_AD<6>	AF7
BOOT_AD<7>	AE7
BOOT_AD<8>	AD7
BOOT_AD<9>	AC7
BOOT_ALE	AD11
BOOT_CE_0_L	AC11
BOOT_CE_1_L	AF10
BOOT_CE_2_L	AD10
BOOT_CE_3_L	AC10
BOOT_OE_L	AF9
BOOT_WAIT_L	AE9
BOOT_WE_L	AE10
CHIP_RESET_L	C4
DDR_A<0>	R23
DDR_A<1>	P24
DDR_A<10>	R24
DDR_A<11>	M23
DDR_A<12>	L26
DDR_A<13>	T23
DDR_A<14>	L24
DDR_A<2>	P23
DDR_A<3>	N24
DDR_A<4>	N23
DDR_A<5>	N26

Pin Name	Ball
DDR_A<6>	N25
DDR_A<7>	M25
DDR_A<8>	M26
DDR_A<9>	M24
DDR_BA<0>	R25
DDR_BA<1>	R26
DDR_BA<2>	L25
DDR_CAS_L	T24
DDR_CB<0>	H24
DDR_CB<1>	J23
DDR_CB<2>	K23
DDR_CB<3>	J24
DDR_CBS_0_N	K26
DDR_CBS_0_P	K25
DDR_CK_0_N	H26
DDR_CK_0_P	H25
DDR_CK_1_N	J26
DDR_CK_1_P	J25
DDR_CK_2_N	AD23
DDR_CK_2_P	AE24
DDR_CK_3_N	AF24
DDR_CK_3_P	AF25
DDR_CKE	L23
DDR_COMP_DN	AE23
DDR_COMP_UP	AF23
DDR_DIMM0_CS0_L	U26
DDR_DIMM0_CS1_L	W26
DDR_DIMM0_ODT_0	V26
DDR_DIMM0_ODT_1	U24
DDR_DIMM1_CS0_L	U25
DDR_DIMM1_CS1_L	U23
DDR_DIMM1_ODT_0	W25
DDR_DIMM1_ODT_1	V24
DDR_DQ<0>	B24
DDR_DQ<1>	C23
DDR_DQ<10>	G23
DDR_DQ<11>	F24
DDR_DQ<12>	E26
DDR_DQ<13>	H23
DDR_DQ<14>	G24
DDR_DQ<15>	F25
DDR_DQ<16>	V23
DDR_DQ<17>	W24
DDR_DQ<18>	W23
DDR_DQ<19>	AA26
DDR_DQ<2>	B26
DDR_DQ<20>	Y24
DDR_DQ<21>	Y23
DDR_DQ<22>	AA25
DDR_DQ<23>	AA24
DDR_DQ<24>	AA23
DDR_DQ<25>	AB25

Pin Name	Ball
DDR_DQ<26>	AB24
DDR_DQ<27>	AB23
DDR_DQ<28>	AC25
DDR_DQ<29>	AC24
DDR_DQ<3>	C25
DDR_DQ<30>	AD25
DDR_DQ<31>	AE26
DDR_DQ<4>	E23
DDR_DQ<5>	D24
DDR_DQ<6>	D25
DDR_DQ<7>	E24
DDR_DQ<8>	F23
DDR_DQ<9>	E25
DDR_DQS_0_N	D26
DDR_DQS_0_P	C26
DDR_DQS_1_N	G26
DDR_DQS_1_P	G25
DDR_DQS_2_N	Y25
DDR_DQS_2_P	Y26
DDR_DQS_3_N	AD26
DDR_DQS_3_P	AC26
DDR_PLL_VDD33	B23
DDR_RAS_L	T26
DDR_VREF	P25
DDR_WE_L	T25
DIAG_CLKOUT	D3
DIAG_CLKOUT_ENABLE	C5
EJTG_TDO	AC21
EJTG_TRST_L	AF22
EPR_CS	AA2
EPR_DI	Y2
EPR_DO	Y3
EPR_SK	AA3
GMI_REF_CLK_N	A8
GMI_REF_CLK_P	A9
GMI_RXCLK/MI1_RXCLK	B14
GND	A1
GND	A6
GND	A22
GND	A26
GND	AA1
GND	AB5
GND	AB7
GND	AB9
GND	AB11
GND	AB13
GND	AB14
GND	AB16
GND	AB18
GND	AB20
GND	AB22
GND	AB26

Pin Name	Ball
GND	AC4
GND	AC23
GND	AD3
GND	AD24
GND	AE2
GND	AE11
GND	AE16
GND	AE25
GND	AF1
GND	AF6
GND	AF21
GND	AF26
GND	B2
GND	B8
GND	B11
GND	B16
GND	B25
GND	C3
GND	C24
GND	D4
GND	D23
GND	E1
GND	E5
GND	E7
GND	E9
GND	E11
GND	E13
GND	E14
GND	E16
GND	E18
GND	E20
GND	E22
GND	F26
GND	G5
GND	G22
GND	H10
GND	H11
GND	H13
GND	H14
GND	H16
GND	H17
GND	J2
GND	J5
GND	J10
GND	J11
GND	J12
GND	J13
GND	J14
GND	J15
GND	J16
GND	J17
GND	J22
GND	K10
GND	K11
GND	K12
GND	K13

Pin Name	Ball
GND	K14
GND	K15
GND	K16
GND	K17
GND	K24
GND	L5
GND	L10
GND	L11
GND	L12
GND	L13
GND	L14
GND	L15
GND	L16
GND	L17
GND	L22
GND	M10
GND	M11
GND	M12
GND	M13
GND	M14
GND	M15
GND	M16
GND	M17
GND	N1
GND	N5
GND	N10
GND	N11
GND	N12
GND	N13
GND	N14
GND	N15
GND	N16
GND	N17
GND	N22
GND	P5
GND	P10
GND	P11
GND	P12
GND	P13
GND	P14
GND	P15
GND	P16
GND	P17
GND	P22
GND	P26
GND	R10
GND	R11
GND	R12
GND	R13
GND	R14
GND	R15
GND	R16
GND	R17
GND	T5
GND	T10

Pin Name	Ball
GND	T11
GND	T12
GND	T13
GND	T14
GND	T15
GND	T16
GND	T17
GND	T22
GND	U2
GND	U10
GND	U11
GND	U12
GND	U13
GND	U14
GND	U15
GND	U16
GND	U17
GND	V5
GND	V10
GND	V11
GND	V12
GND	V13
GND	V14
GND	V15
GND	V16
GND	V17
GND	V22
GND	V25
GND	W10
GND	W11
GND	W13
GND	W14
GND	W16
GND	W17
GND	Y5
GND	Y22
GPIO_0	AE14
GPIO_1	AD14
GPIO_10/BOOT_CE_6_L	AE17
GPIO_11/BOOT_CE_7_L	AD17
GPIO_12/PCM_DATA3	AF18
GPIO_13/PCM_DATA2	AE18
GPIO_14/PCM_FSYNC1	AF19
GPIO_15/PCM_BCLK1	AE19
GPIO_16/PCM_DATA1	AC16
GPIO_17/PCM_DATA0	AD18
GPIO_18/PCM_FSYNC0	AD19
GPIO_19/PCM_BCLK0	AD20
GPIO_2	AC14
GPIO_20/MPI_RX	AC17
GPIO_21/MPI_TX	AC19
GPIO_22/MPI_CS	AC18
GPIO_23/MPI_CLK	AE20
GPIO_3	AF15
GPIO_4	AE15

Pin Name	Ball
GPIO_5	AD15
GPIO_6	AC15
GPIO_7	AF16
GPIO_8/BOOT_CE_4_L	AD16
GPIO_9/BOOT_CE_5_L	AF17
JTG_TCK	AE21
JTG_TDI	AF20
JTG_TDO	AE22
JTG_TMS	AC20
JTG_TRST_L	AD21
MDI_MDC	AF12
MDI_MDIO	AF11
MI0_COL	C12
MI0_CRS	A12
MI0_RXDV	D17
MI0_TXCLK	D12
MI1_TXCLK	C11
NC	A25
NC	A24
PCI_AD<0>	W3
PCI_AD<1>	W1
PCI_AD<10>	T1
PCI_AD<11>	T3
PCI_AD<12>	T2
PCI_AD<13>	T4
PCI_AD<14>	R1
PCI_AD<15>	R3
PCI_AD<16>	N4
PCI_AD<17>	L2
PCI_AD<18>	M3
PCI_AD<19>	K1
PCI_AD<2>	W4
PCI_AD<20>	M4
PCI_AD<21>	K2
PCI_AD<22>	L3
PCI_AD<23>	J1
PCI_AD<24>	K3
PCI_AD<25>	H2
PCI_AD<26>	K4
PCI_AD<27>	G1
PCI_AD<28>	J3
PCI_AD<29>	G2
PCI_AD<3>	W2
PCI_AD<30>	J4
PCI_AD<31>	F1
PCI_AD<4>	V3
PCI_AD<5>	V1
PCI_AD<6>	V4
PCI_AD<7>	V2
PCI_AD<8>	U1
PCI_AD<9>	U4
PCI_BOOT	D5
PCI_CBE_0_L	U3
PCI_CBE_1_L	R2
PCI_CBE_2_L	L1
PCI_CBE_3_L	H1

Pin Name	Ball
PCI_CLK_OUT<0>	B3
PCI_CLK_OUT<1>	B4
PCI_CLK_OUT<2>	A3
PCI_COMP_DN	A4
PCI_COMP_UP	A5
PCI_DEV_GNT_0_L	H3
PCI_DEV_GNT_1_L	H4
PCI_DEV_GNT_2_L	F3
PCI_DEV_GNT_3_L/REQ_L	E2
PCI_DEV_REQ_0_L	G3
PCI_DEV_REQ_1_L	G4
PCI_DEV_REQ_2_L	E3
PCI_DEV_REQ_3_L/GNT_L	F2
PCI_DEVSEL_L	M1
PCI_DLL_VDD33	B1
PCI_ENABLE	Y1
PCI_FRAME_L	N3
PCI_HOST_MODE	B6
PCI_IDSEL	L4
PCI_INTA_L	E4
PCI_INTB_L	C1
PCI_INTC_L	F4
PCI_INTD_L	C2
PCI_IRDY_L	M2
PCI_LOCK_L	N2
PCI_M66EN	B5
PCI_PAR	R4
PCI_PCI100	A2
PCI_PCIXCAP	D6
PCI_PCLK	D1
PCI_PERR_L	P2
PCI_REF_CLKIN	C6
PCI_RST_L	D2
PCI_SERR_L	P1
PCI_STOP_L	P3
PCI_TRDY_L	P4
PLL_DCOK	D7
PLL_MUL_0	D8
PLL_MUL_1	C8
PLL_MUL_2	B7
PLL_REF_CLK	B9
PLL_VDD33	A7
RGM_COMP_DN	A23
RGM_COMP_UP	B22
RGM0_RXC/MI0_RXCLK	C16
RGM0_RXCTL/MI0_RXERR	D16
RGM0_RXD0/MI0_RXD0	C17
RGM0_RXD1/MI0_RXD1	B17
RGM0_RXD2/MI0_RXD2	A17
RGM0_RXD3/MI0_RXD3	A16
RGM0_TXC/MI0_TXEN	D21
RGM0_TXCTL/MI0_TXERR	C22
RGM0_TXD0/MI0_TXD0	D22
RGM0_TXD1/MI0_TXD1	C21
RGM0_TXD2/MI0_TXD2	B21

Pin Name	Ball
RGM0_TXD3/MI0_TXD3	A21
RGM1_RXC/GMI_COL/ MI1_COL	C15
RGM1_RXCTL/GMI_RXERR/ MI1_RXERR	D15
RGM1_RXD0/GMI_RXD0/ MI1_RXD0	D14
RGM1_RXD1/GMI_RXD1/ MI1_RXD1	B15
RGM1_RXD2/GMI_RXD2/ MI1_RXD2	C14
RGM1_RXD3/GMI_RXD3/ MI1_RXD3	A15
RGM1_TXC/GMI_TXEN/ MI1_TXEN	B20
RGM1_TXCTL/GMI_TXERR/ MI1_TXERR	D20
RGM1_TXD0/GMI_TXD0/ MI1_TXD0	C20
RGM1_TXD1/GMI_TXD1/ MI1_TXD1	D19
RGM1_TXD2/GMI_TXD2/ MI1_TXD2	A20
RGM1_TXD3/GMI_TXD3/ MI1_TXD3	C19
RGM2_RXC/GMI_CRS/ MI1_CRS	B12
RGM2_RXCTL/GMI_RXDV/ MI1_RXDV	D13
RGM2_RXD0/GMI_RXD4	A14
RGM2_RXD1/GMI_RXD5	A13
RGM2_RXD2/GMI_RXD6	C13
RGM2_RXD3/GMI_RXD7	B13
RGM2_TXC/GMI_GTXCLK	A19
RGM2_TXCTL	D18
RGM2_TXD0/GMI_TXD4	B19
RGM2_TXD1/GMI_TXD5	C18
RGM2_TXD2/GMI_TXD6	B18
RGM2_TXD3/GMI_TXD7	A18
RSVD1	AC22
RSVD2	AD22
RSVD3	D9
RSVD4	C7
RSVD5	AB1
TWS_SCL	AA4
TWS_SDA	Y4
UART0_CTS_L	AC13
UART0_RTS_L	AD12
UART0_SIN	AE12
UART0_SOUT	AC12
UART1_CTS_L	AF14
UART1_RTS_L	AD13
UART1_SIN	AF13
UART1_SOUT	AE13
USB_DM	A10
USB_DP	A11
USB_GND	D10
USB_REXT	B10

Pin Name	Ball
USB_VDDA33	D11
USB_XI	C9
USB_XO	C10
VDD	H9
VDD	H12
VDD	H15
VDD	H18
VDD	J9
VDD	J18
VDD	K9
VDD	K18
VDD	L9
VDD	L18
VDD	M9
VDD	M18
VDD	N9
VDD	N18
VDD	P9
VDD	P18
VDD	R9
VDD	R18
VDD	T9
VDD	T18
VDD	U9
VDD	U18
VDD	V9
VDD	V18
VDD	W9
VDD	W12
VDD	W15
VDD	W18
VDD18_DDR	F22
VDD18_DDR	H19
VDD18_DDR	H22
VDD18_DDR	J19
VDD18_DDR	K19
VDD18_DDR	K22
VDD18_DDR	L19
VDD18_DDR	M19
VDD18_DDR	M22
VDD18_DDR	N19
VDD18_DDR	P19
VDD18_DDR	R19
VDD18_DDR	R22
VDD18_DDR	T19
VDD18_DDR	U19
VDD18_DDR	U22
VDD18_DDR	V19
VDD18_DDR	W22
VDD18_DDR	AA22
VDD18_DDR	W19
VDD25_RGM	E12
VDD25_RGM	E15
VDD25_RGM	E17
VDD25_RGM	E19
VDD25_RGM	E21
VDD25_RGM	E10
VDD33	AA5
VDD33	AB6
VDD33	AB8
VDD33	AB10
VDD33	AB12
VDD33	AB15
VDD33	AB17
VDD33	AB19
VDD33	AB21
VDD33	E6
VDD33	E8
VDD33	F5
VDD33	H5
VDD33	H8
VDD33	J8
VDD33	K5
VDD33	K8
VDD33	L8
VDD33	M5
VDD33	M8
VDD33	N8
VDD33	P8
VDD33	R5
VDD33	R8
VDD33	T8
VDD33	U5
VDD33	U8
VDD33	V8
VDD33	W5
VDD33	W8

26.4 CN50XX Balls Sorted in Numerical Order

Ball	Pin Name
A1	GND
A10	USB_DM
A11	USB_DP
A12	MI0_CRS
A13	RGM2_RXD1/GMI_RXD5
A14	RGM2_RXD0/GMI_RXD4
A15	RGM1_RXD3/GMI_RXD3/ MI1_RXD3
A16	RGM0_RXD3/MI0_RXD3
A17	RGM0_RXD2/MI0_RXD2
A18	RGM2_TXD3/GMI_TXD7
A19	RGM2_TXC/GMI_GTXCLK
A2	PCI_PCI100
A20	RGM1_TXD2/GMI_TXD2/ MI1_TXD2
A21	RGM0_TXD3/MI0_TXD3
A22	GND
A23	RGM_COMP_DN
A24	NC
A25	NC
A26	GND
A3	PCI_CLK_OUT<2>
A4	PCI_COMP_DN
A5	PCI_COMP_UP
A6	GND
A7	PLL_VDD33
A8	GMI_REF_CLK_N
A9	GMI_REF_CLK_P
AA1	GND
AA2	EPR_CS
AA22	VDD18_DDR
AA23	DDR_DQ<24>
AA24	DDR_DQ<23>
AA25	DDR_DQ<22>
AA26	DDR_DQ<19>
AA3	EPR_SK
AA4	TWS_SCL
AA5	VDD33
AB1	RSVD5
AB10	VDD33
AB11	GND
AB12	VDD33
AB13	GND
AB14	GND
AB15	VDD33
AB16	GND
AB17	VDD33
AB18	GND
AB19	VDD33
AB2	BOOT_AD<31>
AB20	GND
AB21	VDD33

Ball	Pin Name
AB22	GND
AB23	DDR_DQ<27>
AB24	DDR_DQ<26>
AB25	DDR_DQ<25>
AB26	GND
AB3	BOOT_AD<30>
AB4	BOOT_AD<26>
AB5	GND
AB6	VDD33
AB7	GND
AB8	VDD33
AB9	GND
AC1	BOOT_AD<29>
AC10	BOOT_CE_3_L
AC11	BOOT_CE_0_L
AC12	UART0_SOUT
AC13	UART0_CTS_L
AC14	GPIO_2
AC15	GPIO_6
AC16	GPIO_16/PCM_DATA1
AC17	GPIO_20/MPI_RX
AC18	GPIO_22/MPI_CS
AC19	GPIO_21/MPI_TX
AC2	BOOT_AD<28>
AC20	JTG_TMS
AC21	EJTG_TDO
AC22	RSVD1
AC23	GND
AC24	DDR_DQ<29>
AC25	DDR_DQ<28>
AC26	DDR_DQS_3_P
AC3	BOOT_AD<27>
AC4	GND
AC5	BOOT_AD<17>
AC6	BOOT_AD<13>
AC7	BOOT_AD<9>
AC8	BOOT_AD<5>
AC9	BOOT_AD<2>
AD1	BOOT_AD<25>
AD10	BOOT_CE_2_L
AD11	BOOT_ALE
AD12	UART0_RTS_L
AD13	UART1_RTS_L
AD14	GPIO_1
AD15	GPIO_5
AD16	GPIO_8/BOOT_CE_4_L
AD17	GPIO_11/BOOT_CE_7_L
AD18	GPIO_17/PCM_DATA0
AD19	GPIO_18/PCM_FSYNC0
AD2	BOOT_AD<24>
AD20	GPIO_19/PCM_BCLK0
AD21	JTG_TRST_L

Ball	Pin Name
AD22	RSVD2
AD23	DDR_CK_2_N
AD24	GND
AD25	DDR_DQ<30>
AD26	DDR_DQS_3_N
AD3	GND
AD4	BOOT_AD<19>
AD5	BOOT_AD<15>
AD6	BOOT_AD<10>
AD7	BOOT_AD<8>
AD8	BOOT_AD<4>
AD9	BOOT_AD<0>
AE1	BOOT_AD<23>
AE10	BOOT_WE_L
AE11	GND
AE12	UART0_SIN
AE13	UART1_SOUT
AE14	GPIO_0
AE15	GPIO_4
AE16	GND
AE17	GPIO_10/BOOT_CE_6_L
AE18	GPIO_13/PCM_DATA2
AE19	GPIO_15/PCM_BCLK1
AE2	GND
AE20	GPIO_23/MPI_CLK
AE21	JTG_TCK
AE22	JTG_TDO
AE23	DDR_COMP_DN
AE24	DDR_CK_2_P
AE25	GND
AE26	DDR_DQ<31>
AE3	BOOT_AD<21>
AE4	BOOT_AD<18>
AE5	BOOT_AD<14>
AE6	BOOT_AD<11>
AE7	BOOT_AD<7>
AE8	BOOT_AD<3>
AE9	BOOT_WAIT_L
AF1	GND
AF10	BOOT_CE_1_L
AF11	MDI_MDIO
AF12	MDI_MDC
AF13	UART1_SIN
AF14	UART1_CTS_L
AF15	GPIO_3
AF16	GPIO_7
AF17	GPIO_9/BOOT_CE_5_L
AF18	GPIO_12/PCM_DATA3
AF19	GPIO_14/PCM_FSYNC1
AF2	BOOT_AD<22>
AF20	JTG_TDI
AF21	GND

Ball	Pin Name
AF22	EJTG_TRST_L
AF23	DDR_COMP_UP
AF24	DDR_CK_3_N
AF25	DDR_CK_3_P
AF26	GND
AF3	BOOT_AD<20>
AF4	BOOT_AD<16>
AF5	BOOT_AD<12>
AF6	GND
AF7	BOOT_AD<6>
AF8	BOOT_AD<1>
AF9	BOOT_OE_L
B1	PCI_DLL_VDD33
B10	USB_REXT
B11	GND
B12	RGM2_RXC/GMI_CRS/ MI1_CRS
B13	RGM2_RXD3/GMI_RXD7
B14	GMI_RXCLK/MI1_RXCLK
B15	RGM1_RXD1/GMI_RXD1/ MI1_RXD1
B16	GND
B17	RGM0_RXD1/MI0_RXD1
B18	RGM2_TXD2/GMI_TXD6
B19	RGM2_TXD0/GMI_TXD4
B2	GND
B20	RGM1_TXC/GMI_TXEN/ MI1_TXEN
B21	RGM0_TXD2/MI0_TXD2
B22	RGM_COMP_UP
B23	DDR_PLL_VDD33
B24	DDR_DQ<0>
B25	GND
B26	DDR_DQ<2>
B3	PCI_CLK_OUT<0>
B4	PCI_CLK_OUT<1>
B5	PCI_M66EN
B6	PCI_HOST_MODE
B7	PLL_MUL_2
B8	GND
B9	PLL_REF_CLK
C1	PCI_INTB_L
C10	USB_XO
C11	MI1_TXCLK
C12	MI0_COL
C13	RGM2_RXD2/GMI_RXD6
C14	RGM1_RXD2/GMI_RXD2/ MI1_RXD2
C15	RGM1_RXC/GMI_COL/ MI1_COL
C16	RGM0_RXC/MI0_RXCLK
C17	RGM0_RXD0/MI0_RXD0
C18	RGM2_TXD1/GMI_TXD5
C19	RGM1_TXD3/GMI_TXD3/ MI1_TXD3
C2	PCI_INTD_L
C20	RGM1_TXD0/GMI_TXD0/ MI1_TXD0
C21	RGM0_TXD1/MI0_TXD1
C22	RGM0_TXCTL/MI0_TXERR
C23	DDR_DQ<1>
C24	GND
C25	DDR_DQ<3>
C26	DDR_DQS_0_P
C3	GND
C4	CHIP_RESET_L
C5	DIAG_CLKOUT_ENABLE
C6	PCI_REF_CLKIN
C7	RSVD4
C8	PLL_MUL_1
C9	USB_XI
D1	PCI_PCLK
D10	USB_GND
D11	USB_VDDA33
D12	MI0_TXCLK
D13	RGM2_RXCTL/GMI_RXDV/ MI1_RXDV
D14	RGM1_RXD0/GMI_RXD0/ MI1_RXD0
D15	RGM1_RXCTL/GMI_RXERR/ MI1_RXERR
D16	RGM0_RXCTL/MI0_RXERR
D17	MI0_RXDV
D18	RGM2_TXCTL
D19	RGM1_TXD1/GMI_TXD1/ MI1_TXD1
D2	PCI_RST_L
D20	RGM1_TXCTL/GMI_TXERR/ MI1_TXERR
D21	RGM0_TXC/MI0_TXEN
D22	RGM0_TXD0/MI0_TXD0
D23	GND
D24	DDR_DQ<5>
D25	DDR_DQ<6>
D26	DDR_DQS_0_N
D3	DIAG_CLKOUT
D4	GND
D5	PCI_BOOT
D6	PCI_PCIXCAP
D7	PLL_DCOK
D8	PLL_MUL_0
D9	RSVD3
E1	GND
E10	VDD25_RGM
E11	GND
E12	VDD25_RGM
E13	GND
E14	GND
E15	VDD25_RGM
E16	GND
E17	VDD25_RGM
E18	GND
E19	VDD25_RGM
E2	PCI_DEV_GNT_3_L/REQ_L
E20	GND
E21	VDD25_RGM
E22	GND
E23	DDR_DQ<4>
E24	DDR_DQ<7>
E25	DDR_DQ<9>
E26	DDR_DQ<12>
E3	PCI_DEV_REQ_2_L
E4	PCI_INTA_L
E5	GND
E6	VDD33
E7	GND
E8	VDD33
E9	GND
F1	PCI_AD<31>
F2	PCI_DEV_REQ_3_L/GNT_L
F22	VDD18_DDR
F23	DDR_DQ<8>
F24	DDR_DQ<11>
F25	DDR_DQ<15>
F26	GND
F3	PCI_DEV_GNT_2_L
F4	PCI_INTC_L
F5	VDD33
G1	PCI_AD<27>
G2	PCI_AD<29>
G22	GND
G23	DDR_DQ<10>
G24	DDR_DQ<14>
G25	DDR_DQS_1_P
G26	DDR_DQS_1_N
G3	PCI_DEV_REQ_0_L
G4	PCI_DEV_REQ_1_L
G5	GND
H1	PCI_CBE_3_L
H10	GND
H11	GND
H12	VDD
H13	GND
H14	GND
H15	VDD
H16	GND
H17	GND
H18	VDD
H19	VDD18_DDR
H2	PCI_AD<25>
H22	VDD18_DDR
H23	DDR_DQ<13>
H24	DDR_CB<0>
H25	DDR_CK_0_P
H26	DDR_CK_0_N
H3	PCI_DEV_GNT_0_L
H4	PCI_DEV_GNT_1_L

Ball	Pin Name
H5	VDD33
H8	VDD33
H9	VDD
J1	PCI_AD<23>
J10	GND
J11	GND
J12	GND
J13	GND
J14	GND
J15	GND
J16	GND
J17	GND
J18	VDD
J19	VDD18_DDR
J2	GND
J22	GND
J23	DDR_CB<1>
J24	DDR_CB<3>
J25	DDR_CK_1_P
J26	DDR_CK_1_N
J3	PCI_AD<28>
J4	PCI_AD<30>
J5	GND
J8	VDD33
J9	VDD
K1	PCI_AD<19>
K10	GND
K11	GND
K12	GND
K13	GND
K14	GND
K15	GND
K16	GND
K17	GND
K18	VDD
K19	VDD18_DDR
K2	PCI_AD<21>
K22	VDD18_DDR
K23	DDR_CB<2>
K24	GND
K25	DDR_CBS_0_P
K26	DDR_CBS_0_N
K3	PCI_AD<24>
K4	PCI_AD<26>
K5	VDD33
K8	VDD33
K9	VDD
L1	PCI_CBE_2_L
L10	GND
L11	GND
L12	GND
L13	GND
L14	GND
L15	GND
L16	GND
L17	GND
L18	VDD
L19	VDD18_DDR
L2	PCI_AD<17>
L22	GND
L23	DDR_CKE
L24	DDR_A<14>
L25	DDR_BA<2>
L26	DDR_A<12>
L3	PCI_AD<22>
L4	PCI_IDSEL
L5	GND
L8	VDD33
L9	VDD
M1	PCI_DEVSEL_L
M10	GND
M11	GND
M12	GND
M13	GND
M14	GND
M15	GND
M16	GND
M17	GND
M18	VDD
M19	VDD18_DDR
M2	PCI_IRDY_L
M22	VDD18_DDR
M23	DDR_A<11>
M24	DDR_A<9>
M25	DDR_A<7>
M26	DDR_A<8>
M3	PCI_AD<18>
M4	PCI_AD<20>
M5	VDD33
M8	VDD33
M9	VDD
N1	GND
N10	GND
N11	GND
N12	GND
N13	GND
N14	GND
N15	GND
N16	GND
N17	GND
N18	VDD
N19	VDD18_DDR
N2	PCI_LOCK_L
N22	GND
N23	DDR_A<4>
N24	DDR_A<3>
N25	DDR_A<6>
N26	DDR_A<5>
N3	PCI_FRAME_L
N4	PCI_AD<16>
N5	GND
N8	VDD33
N9	VDD
P1	PCI_SERR_L
P10	GND
P11	GND
P12	GND
P13	GND
P14	GND
P15	GND
P16	GND
P17	GND
P18	VDD
P19	VDD18_DDR
P2	PCI_PERR_L
P22	GND
P23	DDR_A<2>
P24	DDR_A<1>
P25	DDR_VREF
P26	GND
P3	PCI_STOP_L
P4	PCI_TRDY_L
P5	GND
P8	VDD33
P9	VDD
R1	PCI_AD<14>
R10	GND
R11	GND
R12	GND
R13	GND
R14	GND
R15	GND
R16	GND
R17	GND
R18	VDD
R19	VDD18_DDR
R2	PCI_CBE_1_L
R22	VDD18_DDR
R23	DDR_A<0>
R24	DDR_A<10>
R25	DDR_BA<0>
R26	DDR_BA<1>
R3	PCI_AD<15>
R4	PCI_PAR
R5	VDD33
R8	VDD33
R9	VDD
T1	PCI_AD<10>
T10	GND
T11	GND
T12	GND
T13	GND
T14	GND
T15	GND
T16	GND
T17	GND

Ball	Pin Name
T18	VDD
T19	VDD18_DDR
T2	PCI_AD<12>
T22	GND
T23	DDR_A<13>
T24	DDR_CAS_L
T25	DDR_WE_L
T26	DDR_RAS_L
T3	PCI_AD<11>
T4	PCI_AD<13>
T5	GND
T8	VDD33
T9	VDD
U1	PCI_AD<8>
U10	GND
U11	GND
U12	GND
U13	GND
U14	GND
U15	GND
U16	GND
U17	GND
U18	VDD
U19	VDD18_DDR
U2	GND
U22	VDD18_DDR
U23	DDR_DIMM1_CS1_L
U24	DDR_DIMM0_ODT_1
U25	DDR_DIMM1_CS0_L
U26	DDR_DIMM0_CS0_L
U3	PCI_CBE_0_L
U4	PCI_AD<9>
U5	VDD33
U8	VDD33
U9	VDD
V1	PCI_AD<5>
V10	GND
V11	GND
V12	GND
V13	GND
V14	GND
V15	GND
V16	GND
V17	GND
V18	VDD
V19	VDD18_DDR
V2	PCI_AD<7>
V22	GND
V23	DDR_DQ<16>
V24	DDR_DIMM1_ODT_1
V25	GND
V26	DDR_DIMM0_ODT_0
V3	PCI_AD<4>
V4	PCI_AD<6>
V5	GND
V8	VDD33

Ball	Pin Name
V9	VDD
W1	PCI_AD<1>
W10	GND
W11	GND
W12	VDD
W13	GND
W14	GND
W15	VDD
W16	GND
W17	GND
W18	VDD
W19	VDD18_DDR
W2	PCI_AD<3>
W22	VDD18_DDR
W23	DDR_DQ<18>
W24	DDR_DQ<17>
W25	DDR_DIMM1_ODT_0
W26	DDR_DIMM0_CS1_L
W3	PCI_AD<0>
W4	PCI_AD<2>
W5	VDD33
W8	VDD33
W9	VDD
Y1	PCI_ENABLE
Y2	EPR_DI
Y22	GND
Y23	DDR_DQ<21>
Y24	DDR_DQ<20>
Y25	DDR_DQS_2_N
Y26	DDR_DQS_2_P
Y3	EPR_DO
Y4	TWS_SDA
Y5	GND

Appendix A

Cavium Networks-Specific Core Instructions

This appendix contains the following subjects:

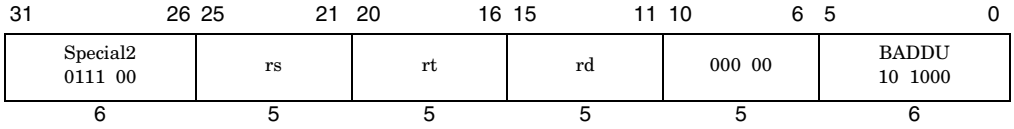
4.1 Core Instructions

Instruction	Detailed Description
CVM_MT_3DES_DEC_CBC	See page 858
CVM_MT_3DES_ENC	See page 859
CVM_MT_3DES_ENC_CBC	See page 860
CVM_MT_3DES_IV	See page 861
CVM_MT_3DES_KEY CVM_MT_KAS_KEY	See page 862
CVM_MT_3DES_RESULT CVM_MT_KAS_RESULT	See page 863
CVM_MT_AES_DEC_CBC0	See page 864
CVM_MT_AES_DEC_CBC1	See page 865
CVM_MT_AES_DEC0	See page 867
CVM_MT_AES_DEC1	See page 868
CVM_MT_AES_ENC_CBC0	See page 869
CVM_MT_AES_ENC_CBC1	See page 870
CVM_MT_AES_ENC0	See page 872
CVM_MT_AES_ENC1	See page 873
CVM_MT_AES_IV	See page 875
CVM_MT_AES_KEY	See page 876
CVM_MT_AES_KEYLENGTH	See page 877
CVM_MT_AES_RESINP	See page 878
CVM_MT_CRC_BYTE	See page 879
CVM_MT_CRC_BYTE_REFLECT	See page 880
CVM_MT_CRC_DWORD	See page 881
CVM_MT_CRC_DWORD_REFLECT	See page 882
CVM_MT_CRC_HALF	See page 883
CVM_MT_CRC_HALF_REFLECT	See page 884
CVM_MT_CRC_IV	See page 885
CVM_MT_CRC_IV_REFLECT	See page 886
CVM_MT_CRC_LEN	See page 887
CVM_MT_CRC_POLYNOMIAL	See page 888
CVM_MT_CRC_POLYNOMIAL_REFLECT	See page 889
CVM_MT_CRC_VAR	See page 890
CVM_MT_CRC_VAR_REFLECT	See page 891
CVM_MT_CRC_WORD	See page 892
CVM_MT_CRC_WORD_REFLECT	See page 893
CVM_MT_GFM_MUL	See page 894
CVM_MT_GFM_POLY	See page 895
CVM_MT_GFM_RESINP	See page 896
CVM_MT_GFM_XOR0	See page 897
CVM_MT_GFM_XORMUL1	See page 898
CVM_MT_HSH_DAT	See page 900
CVM_MT_HSH_DATW	See page 902
CVM_MT_HSH_IV	See page 904
CVM_MT_HSH_IVW	See page 905
CVM_MT_HSH_STARTMD5	See page 907
CVM_MT_HSH_STARTSHA	See page 909
CVM_MT_HSH_STARTSHA256	See page 911

Instruction	Detailed Description
CVM_MT_HSH_STARTSHA512	See page 913
CVM_MT_KAS_ENC	See page 914
CVM_MT_KAS_ENC_CBC	See page 915
DMUL	See page 916
DPOP	See page 917
EXTS	See page 918
EXTS32	See page 919
MTM0	See page 920
MTM1	See page 921
MTM2	See page 922
MTP0	See page 923
MTP1	See page 924
MTP2	See page 925
POP	See page 926
PREF	See page 927
RDHWR	See page 929
SAA	See page 931
SAAD	See page 933
SEQ	See page 935
SEQI	See page 936
SNE	See page 937
SNEI	See page 938
SYNCIOBDMA	See page 939
SYNCS	See page 940
SYNCW	See page 942
SYNCWS	See page 944
ULD	See page 946
ULW	See page 949
USD	See page 952
USW	See page 955
V3MULU	See page 958
VMM0	See page 959
VMULU	See page 961

A.1 Cavium Networks-Specific Instruction Descriptions

Unsigned Byte Add **BADDU**



Format: BADDU rd, rs, rt **CVM**

Purpose:
To do an unsigned byte (8-bit) add.

Description: The 8-bit byte value in GPR rt is added to the 8-bit byte value in GPR rs producing an 8-bit result that is zero- extended.

No integer overflow or any other exception occurs under any circumstance.

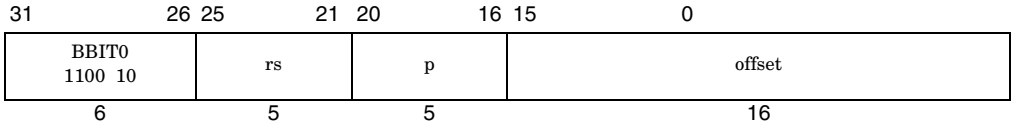
Operation:

```
GPR[rd] = (GPR[rs] + GPR[rt]) & 0xFF
```

Exceptions:

None.

Branch on Bit Clear — BBIT0



Format: BBIT0 rs, p, offset **CVM**

Purpose:

To do a PC-relative conditional branch if one of the lower 32-bits is clear

Description: `if !rs<p> then branch`

An 18-bit signed offset (the 16-bit offset field shifted left 2-bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.

If bit rs<p> is clear, branch to the effective target address after the instruction in the delay slot is executed.

Restrictions:

Processor operation is unpredictable if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

Operation:

```
I:    target_offset <- sign_extend(offset || 00)
      condition <- !GPR[rs]<p>
I+1:  if condition then
         PC <- PC + target_offset
     endif
```

Exceptions:

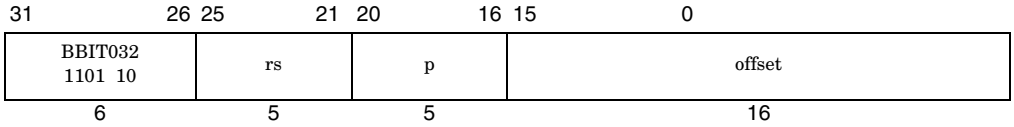
None.

Programming Notes:

With the 18-bit signed instruction offset, the conditional branch range is ± 128 KB. Use jump (J) or jump register (JR) instructions to branch to addresses outside this range.

BBIT0 consumes the major opcode of the MIPS LWC2 instruction. BBIT0 is not considered a COP2 instruction so does not take an exception when COP2 is not enabled.

Branch on Bit Clear Plus 32 — BBIT032



Format: BBIT032 rs, p, offset **CVM**

Purpose:

To do a PC-relative conditional branch if one of the upper 32 bits is clear

Description: `if !rs<p+32> then branch`

An 18-bit signed offset (the 16-bit offset field shifted left 2-bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.

If bit rs<p+32> is clear, branch to the effective target address after the instruction in the delay slot is executed.

Restrictions:

Processor operation is unpredictable if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

Operation:

```
I:    target_offset <- sign_extend(offset || 00)
      condition <- !GPR[rs]<p+32>
I+1:  if condition then
         PC <- PC + target_offset
     endif
```

Exceptions:

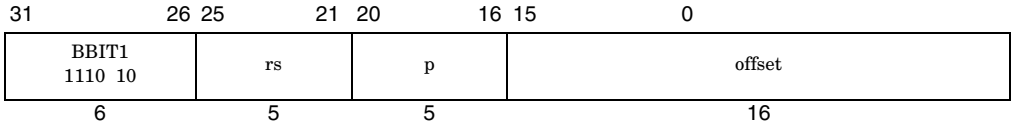
None.

Programming Notes:

With the 18-bit signed instruction offset, the conditional branch range is ± 128 KB. Use jump (J) or jump register (JR) instructions to branch to addresses outside this range.

BBIT032 consumes the major opcode of the MIPS LDC2 instruction. BBIT032 is not considered a COP2 instruction so does not take an exception when COP2 is not enabled.

Branch on Bit Set — BBIT1



Format: BBIT1 rs, p, offset **CVM**

Purpose:

To do a PC-relative conditional branch if one of the lower 32 bits is set

Description: `if rs<p> then branch`

An 18-bit signed offset (the 16-bit offset field shifted left 2-bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.

If-bit rs<p> is set, branch to the effective target address after the instruction in the delay slot is executed.

Restrictions:

Processor operation is unpredictable if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

Operation:

```
I:    target_offset <- sign_extend(offset || 00)
      condition <- GPR[rs]<p>
I+1:  if condition then
         PC <- PC + target_offset
     endif
```

Exceptions:

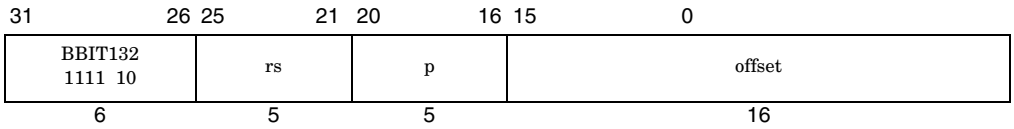
None.

Programming Notes:

With the 18-bit signed instruction offset, the conditional branch range is ± 128 KB. Use jump (J) or jump register (JR) instructions to branch to addresses outside this range.

BBIT1 consumes the major opcode of the MIPS SWC2 instruction. BBIT1 is not considered a COP2 instruction so does not take an exception when COP2 is not enabled.

Branch on Bit Set Plus 32 — BBIT132



Format: BBIT132 rs, p, offset **CVM**

Purpose:

To do a PC-relative conditional branch if one of the upper 32 bits is set

Description: `if rs<p+32> then branch`

An 18-bit signed offset (the 16-bit offset field shifted left 2-bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.

If bit rs<p+32> is set, branch to the effective target address after the instruction in the delay slot is executed.

Restrictions:

Processor operation is unpredictable if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

Operation:

```
I:    target_offset <- sign_extend(offset || 00)
      condition <- GPR[rs]<p+32>
I+1:  if condition then
         PC <- PC + target_offset
     endif
```

Exceptions:

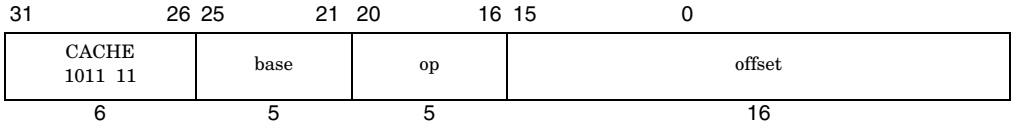
None.

Programming Notes:

With the 18-bit signed instruction offset, the conditional branch range is ± 128 KB. Use jump (J) or jump register (JR) instructions to branch to addresses outside this range.

BBIT132 consumes the major opcode of the MIPS SDC2 instruction. BBIT132 is not considered a COP2 instruction so does not take an exception when COP2 is not enabled.

Perform Cache Operation — CACHE



Format: CACHE op, offset(base) **MIPS32/CVM**

Purpose:

To perform the cache operation specified by op.

Description:

The operation depends on the value of the op field. The possible values are shown in the table below.

NOTE: This is a standard MIPS32 instruction for which OCTEON chips implement all the functionality defined in the MIPS standard instruction set, as well as additional, cnMIPS-specific operations.

Op Value	Name	Description
0, 8, 16, 20	L1 Icache Invalidate	All blocks in the entire L1 Icache become invalid.
1, 9	L1 Dcache Invalidate	All blocks in the entire L1 Dcache become invalid.
2-3, 5-7, 10-11, 13-15, 18-19, 22-31		No operation.
4	Icache Index Load Tag	The 16-bit signed offset is added to the contents of the base register to form an effective address. ● Bits<12:7> of the effective address select the set. ● Bits<14:13> select the way of the cache block involved in the operation. ● Bits<6:3> of the effective address select a 64-bit word within the cache block. The HW loads the virtual tag for the Icache block into the Icache TagLo register and 66-bits of raw Icache data into the Icache DataLo and DataHi registers. The raw data is prior to the repair algorithm indicated by CacheErr[badcol]. If the instruction cache is perfect (i.e. if no repair is required), then: ● bits<63:0> of the raw Icache data contain the two instructions in the word in big-endian format ● bit<64> is the parity bit for the instructions ● <65> is unused. This operation does not cause a cache error exception.
5	Dcache Index Load Tag	The 16-bit signed offset is added to the contents of the base register to form an effective address. ● Bits<13:8> of the effective address select the particular cache block involved in the operation (i.e. the way) ● Bit<7> selects the set. ● Bits<6:3> selects a 64-bit word within the cache block. The HW loads the virtual and physical tags for the Dcache block into the Dcache TagLo and TagHi registers and the 64-bit data word and corresponding parity bits into the Dcache DataLo and DataHi registers. This operation does not cause a cache error exception.

Op Value	Name	Description
12	Icache Bitmap Store	The 16-bit signed offset is added to the contents of the base register to form an effective address. ● Bits<12:7> of the effective address selects the set. ● Bits<14:13> selects the way of the cache block involved in the operation. ● Bits<6:3> of the effective address selects a 64-bit word within the cache block. The HW fills the virtual tag for the Icache block with the contents of the Icache TagLo register and fills 66-bits into the selected Icache word as raw data. The 66 bits written are 33 duplicates of DataLo<1:0>, so only values 0x0000000000000000, 0x15555555555555555, 0x2AAAAAAAAAAAAAAA, and 0x3FFFFFFFFFFFFFFF are possible. The raw data is independent of the repair algorithm indicated by CacheErr[badcol]. If the instruction cache is perfect (i.e. if no repair is required), then: ● bits<63:0> of the 66-bits written correspond to the the two instructions of the word in big-endian format ● bit<64> is the parity bit ● bit<65> is not used.
17, 21	L1 Dcache Virtual Tag Invalidate	All valid bits for all virtual tags in the entire L1 Dcache become invalid. The valid bits for the physical tags in the Dcache remain unchanged.

Restrictions:

If access to Coprocessor 0 is not enabled, a Coprocessor Unusable Exception is signaled.

Operation:

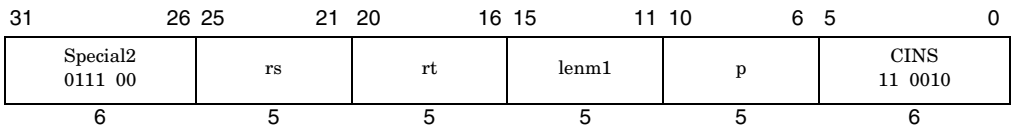
```
if IsCoprocessorEnabled(0) then
   vAddr = GPR[base] + sign_extend(offset)
   CacheOp(op, vAddr)
else
   SignalException(CoprocessorUnusable, 0)
endif
```

Exceptions:

Coprocessor Unusable Exception

CACHE instructions never match watches/breakpoints on OCTEON.

Clear and Insert a Bit Field — CINS



Format: CINS rt, rs, p, lenm1 **CVM**

Purpose:

To insert a bit field that starts in the lower 32 bits of a register and clear the rest of the register.

Description: `rt = rs<lenm1:0> << p`
The contents of general-purpose register rt from bit location p + lenm1 to bit location p are filled with the contents of general-purpose register rs from bit location lenm1 to bit location 0. The remaining bits in rt are zeroed by this instruction.

Restrictions:

The p and lenm1 fields are only 5-bits, so the widest allowed bit field is 32-bits.

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

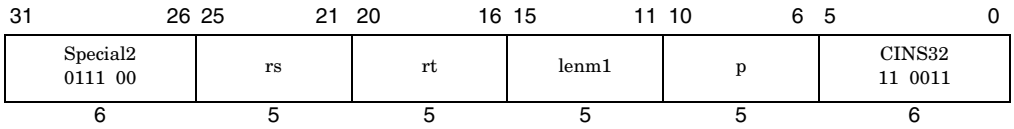
Operation:

```
if Are64bitOperationsEnabled() then
   GPR[rt] <- GPR[rs]<lenm1:0> << p
else
   SignalException(ReservedInstruction)
endif
```

Exceptions:

Reserved Instruction.

Clear and Insert a Bit Field Plus 32 — CINS32



Format: CINS32 rt, rs, p, lenm1 **CVM**

Purpose:

To insert a bit field that starts in the upper 32 bits of a register and clear the rest of the register.

Description: `rt = rs<lenm1:0> << (p + 32)`

The contents of general-purpose register rt from bit location p + 32 + lenm1 to bit location p + 32 are filled with the contents of general-purpose register rs from bit location lenm1 to bit location 0. The remaining bits in rt are zeroed by this instruction.

Restrictions:

The p and lenm1 fields are only 5-bits, so the widest allowed bit field is 32-bits.

The result register GPR[rt] is unpredictable when p+32+lenm1 is larger than 63.

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

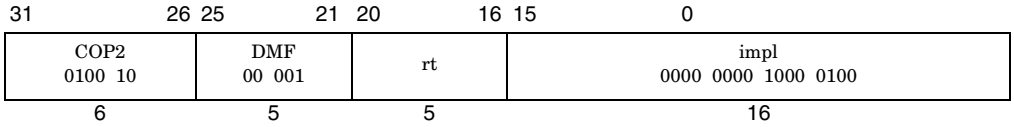
Operation:

```
if Are64bitOperationsEnabled() then
   GPR[rt] <- GPR[rs]<lenm1:0> << (p + 32)
else
   SignalException(ReservedInstruction)
endif
```

Exceptions:

Reserved Instruction.

Load IV from 3DES Unit — CVM_MF_3DES_IV



Format: DMFC2 rt, 0x0084 **CVM**

Purpose:

To read 3DESIV from the 3DES coprocessor.

Description: `rt = 3DESIV<63:0>`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

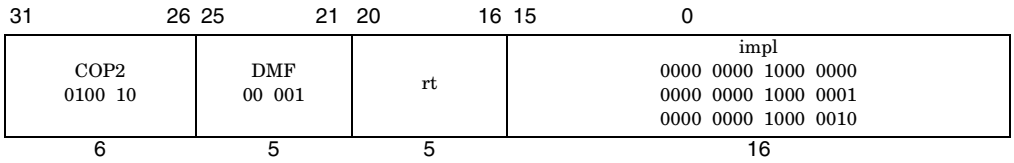
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CVMCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      GPR[rt] = 3DESIV<63:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Key from 3DES Unit **CVM_MF_3DES_KEY**
Load Key from KASUMI Unit **CVM_MF_KAS_KEY**



Format: DMFC2 rt, 0x0080 **CVM**
DMFC2 rt, 0x0081
DMFC2 rt, 0x0082

Purpose:

To read 3DESKEY values from the 3DES and KASUMI coprocessors. 3DESKEY[2:0] are used by the 3DES unit, 3DESKEY[1:0] are used by the KASUMI unit.

Description:
```
rt = 3DESKEY[0]  // DMFC2 rt, 0x0080
rt = 3DESKEY[1]  // DMFC2 rt, 0x0081
rt = 3DESKEY[2]  // DMFC2 rt, 0x0082
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

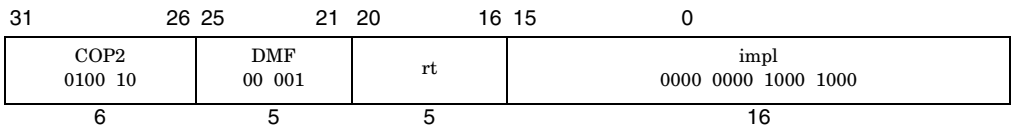
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0080: GPR[rt] = 3DES_KEY[0]
      case 0x0081: GPR[rt] = 3DES_KEY[1]
      case 0x0082: GPR[rt] = 3DES_KEY[2]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Result from 3DES Unit **CVM_MF_3DES_RESULT**
Load Result from KASUMI Unit **CVM_MF_KAS_RESULT**



Format: DMFC2 rt, 0x0088 **CVM**

Purpose:

To read 3DESRESULT from the 3DES and KASUMI coprocessors.

Description: `rt = 3DESRESULT<63:0>`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

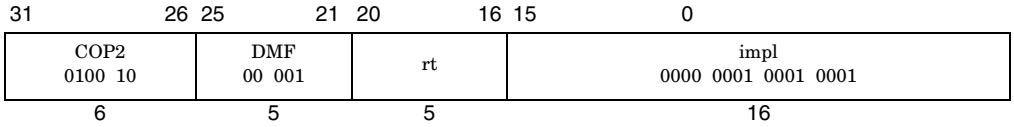
```
if IsCoprocessorEnabled(2) then
     if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
        SignalException(ReservedInstruction)
     else
        GPR[rt] = 3DESRESULT<63:0>
     endif
  else
     SignalException(CoprocessorUnusable, 2) endif
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load INP0 from AES Unit — CVM_MF_AES_INP0



Format: DMFC2 rt, 0x0111 **CVM**

Purpose:

To read AES input from the AES coprocessor.

Description: `rt = AESRESINP[0]`

CVM_MF_AES_RESINP0 should be used instead of this instruction. This operation is deprecated, and is present only for backward compatibility with earlier revisions of the architecture.

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

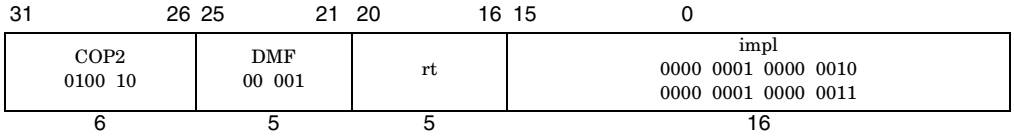
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      GPR[rt] = AESRESINP[0]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load IV from AES Unit — CVM_MF_AES_IV



Format: DMFC2 rt, 0x0102 **CVM**

DMFC2 rt, 0x0103

Purpose:

To read AESIV from the AES coprocessor.

Description:

```
rt = AESIV[0]   // DMFC2 rt, 0x0102
rt = AESIV[1]   // DMFC2 rt, 0x0103
```

Restrictions:

CVM_MF_AES_IV1 (i.e. “DMFC2 rt, 0x0103”) must not be the first AES-related instruction that immediately follows a prior CVM_MT_AES_ENC_CBC1, CVM_MT_AES_ENC1, CVM_MT_AES_DEC1, or CVM_MT_AES_DEC1 instruction. CVM_MF_AES_IV1 should always follow CVM_MF_AES_IV0.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

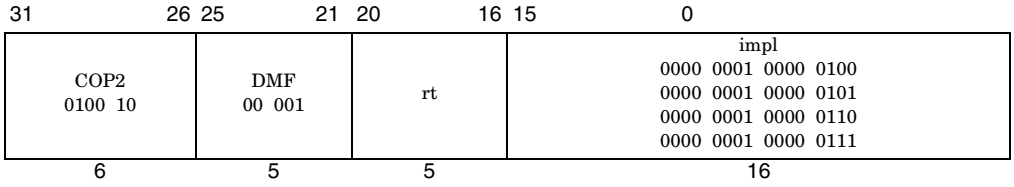
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0102: GPR[rt] = AESIV[0]
      case 0x0103: GPR[rt] = AESIV[1]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Key from AES Unit — CVM_MF_AES_KEY



Format: DMFC2 rt, 0x0104 **CVM**

DMFC2 rt, 0x0105

DMFC2 rt, 0x0106

DMFC2 rt, 0x0107

Purpose:

To read AESKEY from the AES coprocessor.

Description:

```
rt = AESKEY[0]  // DMFC2 rt, 0x0104
rt = AESKEY[1]  // DMFC2 rt, 0x0105
rt = AESKEY[2]  // DMFC2 rt, 0x0106
rt = AESKEY[3]  // DMFC2 rt, 0x0107
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

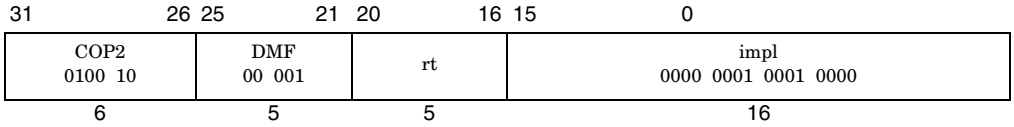
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0104: GPR[rt] = AESKEY[0]
      case 0x0105: GPR[rt] = AESKEY[1]
      case 0x0106: GPR[rt] = AESKEY[2]
      case 0x0107: GPR[rt] = AESKEY[3]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Keylength from AES Unit — CVM_MF_AES_KEYLENGTH



Format: DMFC2 rt, 0x0110 **CVM**

Purpose:

To read AESKEYLEN from the AES coprocessor.

Description: `rt = AESKEYLEN<1:0>`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

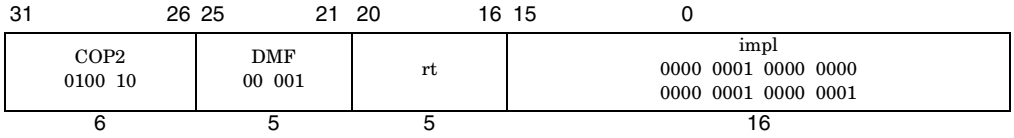
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      GPR[rt] = 0^62 || AESKEYLEN<1:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Result/Input from AES Unit — CVM_MF_AES_RESINP



Format: DMFC2 rt, 0x0100 **CVM**

DMFC2 rt, 0x0101

Purpose:

To read AESRESINP from the AES coprocessor.

Description:

```
rt = AESRESINP[0]   // DMFC2 rt, 0x0100
rt = AESRESINP[1]   // DMFC2 rt, 0x0101
```

Restrictions:

CVM_MF_AES_RESINP1 (i.e. “DMFC2 rt, 0x0101”) must not be the first AES-related instruction that immediately follows a prior CVM_MT_AES_ENC_CBC1, CVM_MT_AES_ENC1, CVM_MT_AES_DEC1, or CVM_MT_AES_DEC1 instruction. CVM_MF_AES_RESINP1 should always follow CVM_MF_AES_RESINP0.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

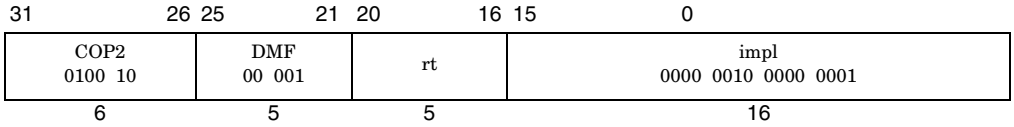
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0100: GPR[rt] = AESRESINP[0]
      case 0x0101: GPR[rt] = AESRESINP[1]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load IV from CRC Unit — CVM_MF_CRC_IV



Format: DMFC2 rt, 0x0201 **CVM**

Purpose:

To read CRCIV from the CRC coprocessor.

Description: `rt = sign_extend(CRCIV<31:0>)`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

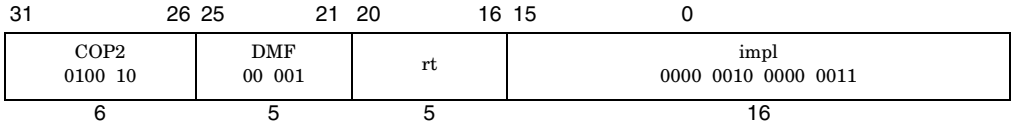
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) then
      SignalException(ReservedInstruction)
   else
      GPR[rt] = sign_extend(CRCIV<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load IV from CRC Unit Reflected — CVM_MF_CRC_IV_REFLECT



Format: DMFC2 rt, 0x0203 **CVM**

Purpose:

To read CRCIV from the CRC coprocessor. The bits within the entire CRCIV word are reflected.

Description: `rt = sign_extend(word_bit_reflect(CRCIV<31:0>))`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

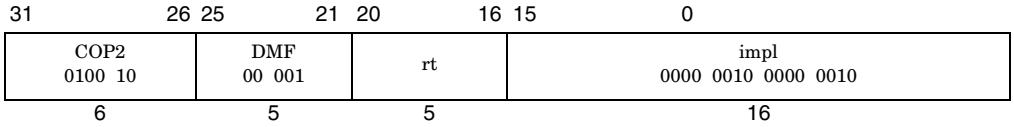
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) then
      SignalException(ReservedInstruction)
   else
      GPR[rt] = sign_extend(word_bit_reflect(CRCIV<31:0>))
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Length from CRC Unit — CVM_MF_CRC_LEN



Format: DMFC2 rt, 0x0202 **CVM**

Purpose:

To read CRCLEN from the CRC coprocessor.

Description: `rt = CRCLEN<3:0>`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

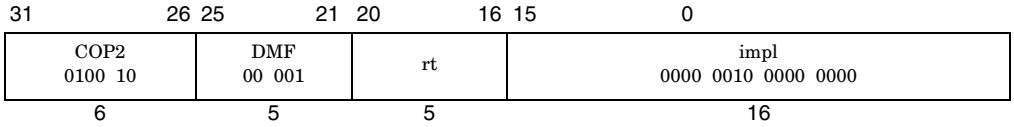
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) then
      SignalException(ReservedInstruction)
   else
      GPR[rt] = 0^60 || CRCLEN<3:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Polynomial from CRC Unit — CVM_MF_CRC_POLYNOMIAL



Format: DMFC2 rt, 0x0200 **CVM**

Purpose:

To read CRCPOLY from the CRC coprocessor.

Description: `rt = sign_extend(CRCPOLY<31:0>)`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

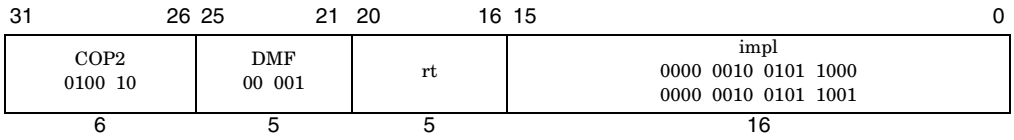
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) then
      SignalException(ReservedInstruction)
   else
      GPR[rt] = sign_extend(CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Multiplier from GFM Unit — CVM_MF_GFM_MUL



Format: DMFC2 rt, 0x0258 **CVM**

DMFC2 rt, 0x0259

Purpose:

To read GFMMUL from the GFM coprocessor.

Description:

```
rt = GFMMUL[0] // DMFC2 rt, 0x0258
rt = GFMMUL[1] // DMFC2 rt, 0x0259
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

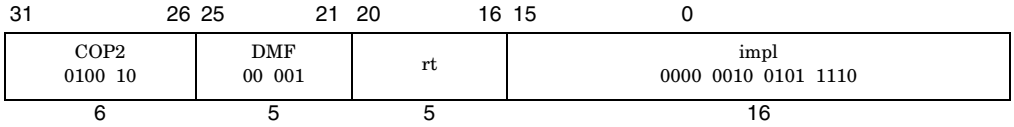
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0258: GPR[rt]<63:0> = GFMMUL[0]<63:0>)
      case 0x0259: GPR[rt]<63:0> = GFMMUL[1]<63:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Polynomial from GFM Unit — CVM_MF_GFM_POLY



Format: DMFC2 rt, 0x025E **CVM**

Purpose:

To read GFMPOLY from the GFM coprocessor.

Description: `rt = GFMPOLY<15:0>`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

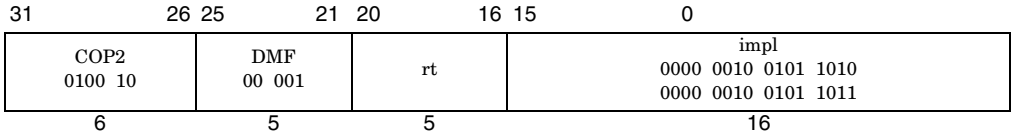
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      GPR[rt]<63:0> = 0^48 || GFMPOLY<15:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Result/Input from GFM Unit — CVM_MF_GFM_RESINP



Format: DMFC2 rt, 0x025A **CVM**

DMFC2 rt, 0x025B

Purpose:

To read GFMMUL from the GFM coprocessor.

Description:

```
rt = GFMRESINP[0] // DMFC2 rt, 0x025A
rt = GFMRESINP[1] // DMFC2 rt, 0x025B
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

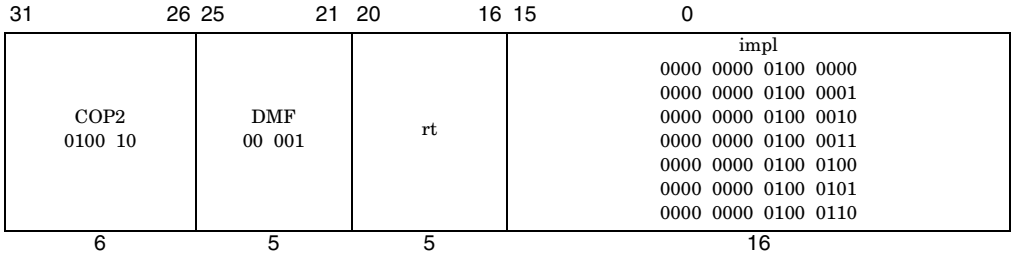
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x025A: GPR[rt]<63:0> = GFMRESINP[0]<63:0>
      case 0x025B: GPR[rt]<63:0> = GFMRESINP[1]<63:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Data from HSH Unit (narrow mode) CVM_MF_HSH_DAT



Format: DMFC2 rt, 0x0040 **CVM**

DMFC2 rt, 0x0041

DMFC2 rt, 0x0042

DMFC2 rt, 0x0043

DMFC2 rt, 0x0044

DMFC2 rt, 0x0045

DMFC2 rt, 0x0046

Purpose:

To read HASHDAT from the HSH coprocessor in narrow mode. MD5, SHA-1, and SHA-256 use the narrow mode. The most-significant and least-significant 32 bits of rt are written with the least-significant 32 bits of separate 64-bit HASHDAT double-words.

Description:

```
rt = HASHDAT[0]<31:0>  || HASHDAT[1]<31:0>  // DMFC2 rt, 0x0040
rt = HASHDAT[2]<31:0>  || HASHDAT[3]<31:0>  // DMFC2 rt, 0x0041
rt = HASHDAT[4]<31:0>  || HASHDAT[5]<31:0>  // DMFC2 rt, 0x0042
rt = HASHDAT[6]<31:0>  || HASHDAT[7]<31:0>  // DMFC2 rt, 0x0043
rt = HASHDAT[8]<31:0>  || HASHDAT[9]<31:0>  // DMFC2 rt, 0x0044
rt = HASHDAT[10]<31:0> || HASHDAT[11]<31:0> // DMFC2 rt, 0x0045
rt = HASHDAT[12]<31:0> || HASHDAT[13]<31:0> // DMFC2 rt, 0x0046
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0040: GPR[rt]<63:32> = HASHDAT[ 0]<31:0>
                   GPR[rt]<31:0>  = HASHDAT[ 1]<31:0>
      case 0x0041: GPR[rt]<63:32> = HASHDAT[ 2]<31:0>
                   GPR[rt]<31:0>  = HASHDAT[ 3]<31:0>
      case 0x0042: GPR[rt]<63:32> = HASHDAT[ 4]<31:0>
                   GPR[rt]<31:0>  = HASHDAT[ 5]<31:0>
```

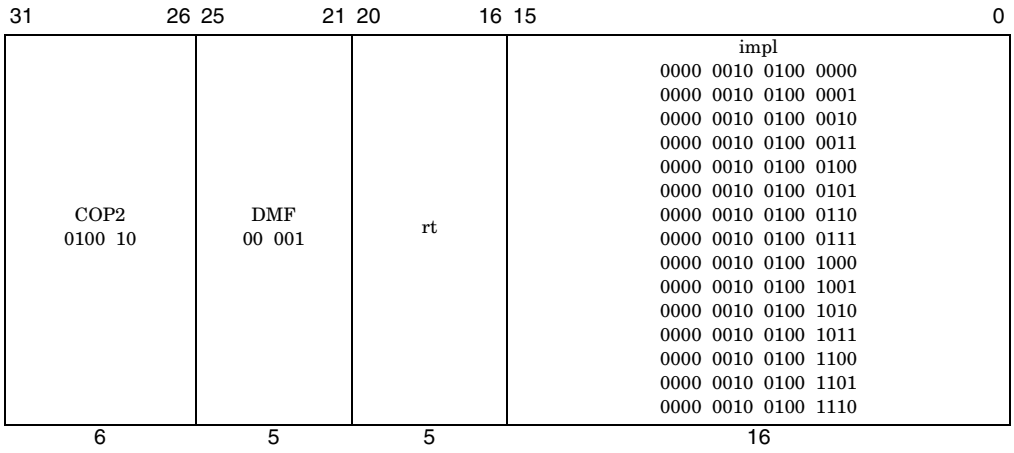
```
      case 0x0043: GPR[rt]<63:32> = HASHDAT[ 6]<31:0>
                   GPR[rt]<31:0>  = HASHDAT[ 7]<31:0>
      case 0x0044: GPR[rt]<63:32> = HASHDAT[ 8]<31:0>
                   GPR[rt]<31:0>  = HASHDAT[ 9]<31:0>
      case 0x0045: GPR[rt]<63:32> = HASHDAT[10]<31:0>
                   GPR[rt]<31:0>  = HASHDAT[11]<31:0>
      case 0x0046: GPR[rt]<63:32> = HASHDAT[12]<31:0>
                   GPR[rt]<31:0>  = HASHDAT[13]<31:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Data from HSH Unit (wide mode) CVM_MF_HSH_DATW



Format: DMFC2 rt, 0x0240 **CVM**

DMFC2 rt, 0x0241

DMFC2 rt, 0x0242

DMFC2 rt, 0x0243

DMFC2 rt, 0x0244

DMFC2 rt, 0x0245

DMFC2 rt, 0x0246

DMFC2 rt, 0x0247

DMFC2 rt, 0x0248

DMFC2 rt, 0x0249

DMFC2 rt, 0x024A

DMFC2 rt, 0x024B

DMFC2 rt, 0x024C

DMFC2 rt, 0x024D

DMFC2 rt, 0x024E

Purpose:

To read HASHDAT from the HSH coprocessor in wide mode. SHA-512 uses the wide mode.

Description:

```
rt = HASHDAT[0]  // DMFC2 rt, 0x0240
rt = HASHDAT[1]  // DMFC2 rt, 0x0241
rt = HASHDAT[2]  // DMFC2 rt, 0x0242
rt = HASHDAT[3]  // DMFC2 rt, 0x0243
rt = HASHDAT[4]  // DMFC2 rt, 0x0244
rt = HASHDAT[5]  // DMFC2 rt, 0x0245
rt = HASHDAT[6]  // DMFC2 rt, 0x0246
rt = HASHDAT[7]  // DMFC2 rt, 0x0247
rt = HASHDAT[8]  // DMFC2 rt, 0x0248
rt = HASHDAT[9]  // DMFC2 rt, 0x0249
rt = HASHDAT[10] // DMFC2 rt, 0x024A
rt = HASHDAT[11] // DMFC2 rt, 0x024B
rt = HASHDAT[12] // DMFC2 rt, 0x024C
rt = HASHDAT[13] // DMFC2 rt, 0x024D
rt = HASHDAT[14] // DMFC2 rt, 0x024E
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

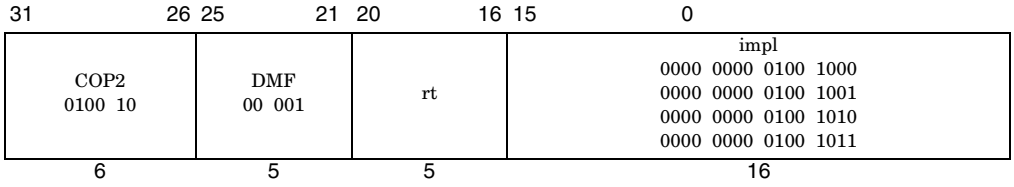
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0040: GPR[rt] = HASHDAT[0]
      case 0x0041: GPR[rt] = HASHDAT[1]
      case 0x0042: GPR[rt] = HASHDAT[2]
      case 0x0043: GPR[rt] = HASHDAT[3]
      case 0x0044: GPR[rt] = HASHDAT[4]
      case 0x0045: GPR[rt] = HASHDAT[5]
      case 0x0046: GPR[rt] = HASHDAT[6]
      case 0x0047: GPR[rt] = HASHDAT[7]
      case 0x0048: GPR[rt] = HASHDAT[8]
      case 0x0049: GPR[rt] = HASHDAT[9]
      case 0x004A: GPR[rt] = HASHDAT[10]
      case 0x004B: GPR[rt] = HASHDAT[11]
      case 0x004C: GPR[rt] = HASHDAT[12]
      case 0x004D: GPR[rt] = HASHDAT[13]
      case 0x004E: GPR[rt] = HASHDAT[14]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load IV from HSH Unit (narrow mode) CVM_MF_HSH_IV



Format: DMFC2 rt, 0x0048 **CVM**

DMFC2 rt, 0x0049

DMFC2 rt, 0x004A

DMFC2 rt, 0x004B

Purpose:

To read HASHIV from the HSH coprocessor in narrow mode. MD5, SHA-1, and SHA-256 use the narrow mode. The most-significant and least-significant 32 bits of rt are written with the least-significant 32 bits of separate 64-bit HASHIV double-words.

Description:

```
rt = HASHIV[0]<31:0> || HASHIV[1]<31:0> // DMFC2 rt, 0x0048
rt = HASHIV[2]<31:0> || HASHIV[3]<31:0> // DMFC2 rt, 0x0049
rt = HASHIV[4]<31:0> || HASHIV[5]<31:0> // DMFC2 rt, 0x004A
rt = HASHIV[6]<31:0> || HASHIV[7]<31:0> // DMFC2 rt, 0x004B
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

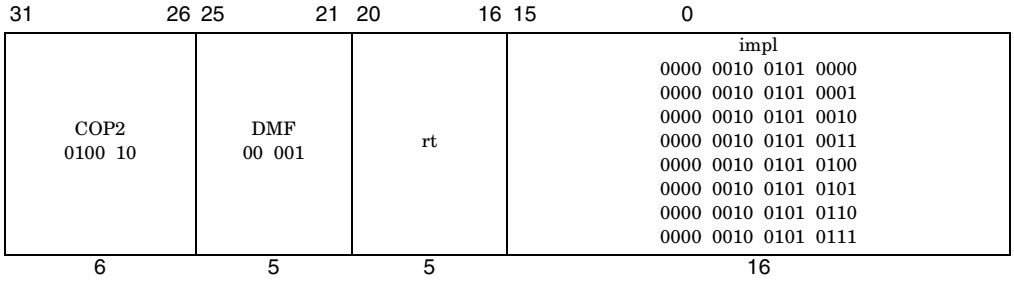
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0048: GPR[rt]<63:32> = HASHIV[0]<31:0>
                   GPR[rt]<31:0>  = HASHIV[1]<31:0>
      case 0x0049: GPR[rt]<63:32> = HASHIV[2]<31:0>
                   GPR[rt]<31:0>  = HASHIV[3]<31:0>
      case 0x004A: GPR[rt]<63:32> = HASHIV[4]<31:0>
                   GPR[rt]<31:0>  = HASHIV[5]<31:0>
      case 0x004B: GPR[rt]<63:32> = HASHIV[6]<31:0>
                   GPR[rt]<31:0>  = HASHIV[7]<31:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load IV from HSH Unit (wide mode) CVM_MF_HSH_IVW



Format: DMFC2 rt, 0x0250 **CVM**

DMFC2 rt, 0x0251

DMFC2 rt, 0x0252

DMFC2 rt, 0x0253

DMFC2 rt, 0x0254

DMFC2 rt, 0x0255

DMFC2 rt, 0x0256

DMFC2 rt, 0x0257

Purpose:

To read HASHIV from the HSH coprocessor in wide mode. SHA-512 use the wide mode.

Description:

```
rt = HASHIV[0] // DMFC2 rt, 0x0250
rt = HASHIV[1] // DMFC2 rt, 0x0251
rt = HASHIV[2] // DMFC2 rt, 0x0252
rt = HASHIV[3] // DMFC2 rt, 0x0253
rt = HASHIV[4] // DMFC2 rt, 0x0254
rt = HASHIV[5] // DMFC2 rt, 0x0255
rt = HASHIV[6] // DMFC2 rt, 0x0256
rt = HASHIV[7] // DMFC2 rt, 0x0257
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0250: GPR[rt] = HASHIV[0]
      case 0x0251: GPR[rt] = HASHIV[1]
      case 0x0252: GPR[rt] = HASHIV[2]
      case 0x0253: GPR[rt] = HASHIV[3]
      case 0x0254: GPR[rt] = HASHIV[4]
      case 0x0255: GPR[rt] = HASHIV[5]
```

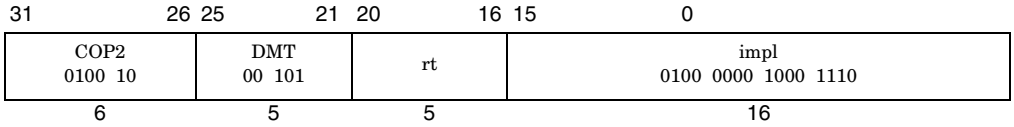
```
      case 0x0256: GPR[rt] = HASHIV[6]
      case 0x0257: GPR[rt] = HASHIV[7]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

3DES Decrypt — CVM_MT_3DES_DEC



Format: DMTC2 rt, 0x408E **CVM**

Purpose:

To do a 3DES EDE decrypt. GPR[rt] is the input to a 3DES EDE decrypt (using 3DESKEY). 3DESRESULT is set to the result of the decrypt, and 3DESIV is unpredictable.

Description: `3DESRESULT<63:0> = 3DES_EDE_DEC(rt, 3DESKEY[2:0])`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

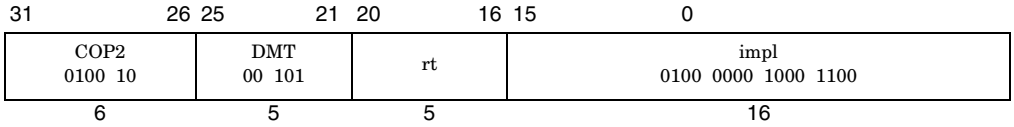
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      3DESRESULT<63:0> = 3DES_EDE_DEC(GPR[rt], 3DESKEY[2:0])
      3DESIV<63:0> = unpredictable
      KASUMIRESULT<63:0> = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

3DES CBC Decrypt — CVM_MT_3DES_DEC_CBC



Format: DMTC2 rt, 0x408C **CVM**

Purpose:

To do a 3DES EDE CBC decrypt. GPR[rt] is the input to a 3DES EDE decrypt (using 3DESKEY). 3DESRESULT is set to the result of the decrypt XOR 3DESIV, and 3DESIV is set to GPR[rt].

Description:

```
3DESRESULT<63:0> = 3DES_EDE_DEC(rt, 3DESKEY[2:0]) ⊕ 3DESIV<63:0>
3DESIV<63:0> = rt
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

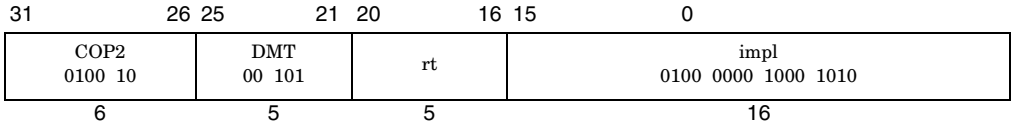
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      3DESRESULT<63:0> = 3DES_EDE_DEC(GPR[rt], 3DESKEY[2..0]) ⊕ 3DESIV<63:0>
      3DESIV<63:0> = GPR[rt]
      KASUMIRESULT<63:0> = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

3DES Encrypt — CVM_MT_3DES_ENC



Format: DMTC2 rt, 0x408A **CVM**

Purpose:

To do a 3DES EDE encrypt. GPR[rt] is the input to a 3DES EDE encrypt (using 3DESKEY). 3DESRESULT is set to the result of the encrypt and 3DESIV is unpredictable.

Description: `3DESRESULT<63:0> = 3DES_EDE_ENC(rt, 3DESKEY[2:0])`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

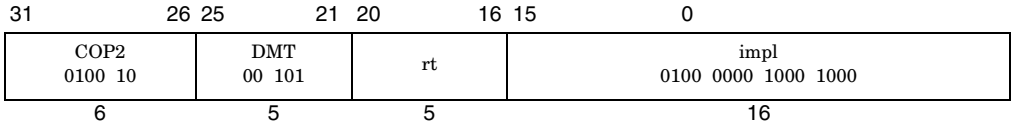
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      3DESRESULT<63:0> = 3DES_EDE_ENC(GPR[rt], 3DESKEY[2:0])
      3DESIV<63:0> = unpredictable
      KASUMIRESULT<63:0> = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

3DES CBC Encrypt **CVM_MT_3DES_ENC_CBC**



Format: Format: DMTC2 rt, 0x4088 **CVM**

Purpose:

To do a 3DES EDE CBC encrypt. The XOR of GPR[rt] and 3DESIV is the input to a 3DES EDE encrypt (using 3DESKEY). 3DESRESULT and 3DESIV are set to the result of the encrypt.

Description:
```
3DESRESULT<63:0> = 3DESIV<63:0> =
                   3DES_EDE_ENC(rt ⊕ 3DESIV<63:0>, 3DESKEY[2:0])
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

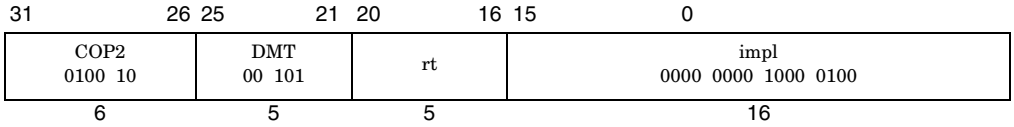
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      3DESRESULT<63:0> = 3DESIV<63:0> =
             3DES_EDE_ENC(GPR[rt] ⊕ 3DESIV<63:0>, 3DESKEY[2:0])
      KASUMIRESULT<63:0> = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load IV into 3DES Unit — CVM_MT_3DES_IV



Format: DMTC2, rt, 0x0084 **CVM**

Purpose:

To load a value into 3DESIV.

Description: `3DESIV<63:0> = rt`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

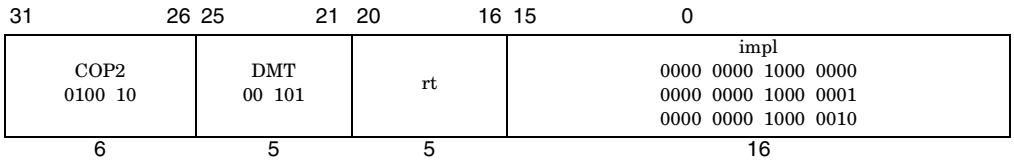
```
if IsCoprocessorEnabled(2) then
     if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
        SignalException(ReservedInstruction)
     else
        3DESIV<63:0> = GPR[rt]
     endif
  else
     SignalException(CoprocessorUnusable, 2)
  endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Key into 3DES Unit **CVM_MT_3DES_KEY**
Load Key into KASUMI Unit **CVM_MT_KAS_KEY**



Format: DMTC2 rt, 0x0080 **CVM**
DMTC2 rt, 0x0081
DMTC2 rt, 0x0082

Purpose:

To load values into 3DESKEY. 3DESKEY[2:0] are used by the 3DES unit, 3DESKEY[1:0] are used by the KASUMI unit.

Description:
```
3DESKEY[0]  = rt // DMTC2 rt,0x0080
3DESKEY[1]  = rt // DMTC2 rt,0x0081
3DESKEY[2]  = rt // DMTC2 rt,0x0082
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

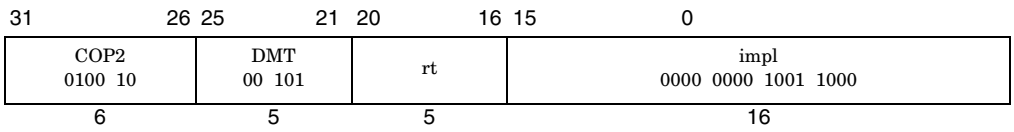
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0080: 3DESKEY[0] = GPR[rt]
      case 0x0081: 3DESKEY[1] = GPR[rt]
      case 0x0082: 3DESKEY[2] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Result into 3DES Unit **CVM_MT_3DES_RESULT**
Load Result into KASUMI Unit **CVM_MT_KAS_RESULT**



Format: DMTC2 rt, 0x0098 **CVM**

Purpose:

To load a value into 3DESRESULT.

Description: `3DESRESULT<63:0> = rt`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

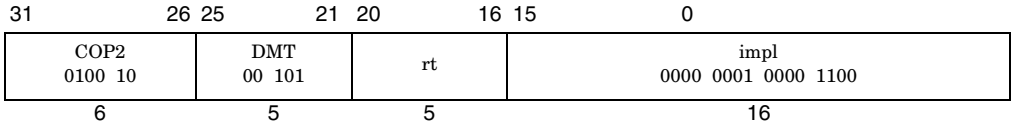
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      3DESRESULT<63:0> = GPR[rt]
      KASUMIRESULT<63:0> = 3DESRESULT<63:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

AES CBC Decrypt (part 1) — CVM_MT_AES_DEC_CBC0



Format: DMTC2 rt, 0x010C **CVM**

Purpose:

To load AESRESINP[0] in preparation for a subsequent CVM_MT_AES_DEC_CBC1.

Description: `AESRESINP[0] = rt`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

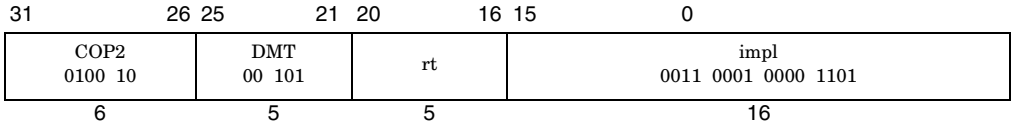
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      AESRESINP[0] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

AES CBC Decrypt (part 2) — CVM_MT_AES_DEC_CBC1



Format: DMTC2 rt, 0x310D **CVM**

Purpose:

To perform an AES CBC decrypt operation. GPR[rt] and AESRESINP[0] are the inputs to an AES decryption operation (using AESKEY and AESKEYLEN). AESRESINP is updated with the result of the decryption XORed with AESIV. AESIV is set to the inputs of the decryption. Unused AESKEY locations become unpredictable.

Description:

```
TIN[1:0] = rt || AESRESINP[0]
AESRESINP[1:0] = AES_DEC(TIN, AESKEY[3:0], AESKEYLEN<1:0>) ⊕
                 AESIV[1:0]
AESIV[1:0] = TIN[1:0]
```

Restrictions:

CVM_MT_AES_DEC_CBC0 (or CVM_MT_AES_RESINP0 in the save/restore case) must be executed between the execution of this CVM_MT_AES_DEC_CBC1 and the immediately preceding CVM_MT_AES_ENC_CBC1, CVM_MT_AES_ENC1, CVM_MT_AES_DEC_CBC1, or CVM_MT_AES_DEC1 instruction.

AESIV[1:0] must be predictable prior to this CVM_MT_AES_DEC_CBC1 instruction. This means that AESIV[1..0] must be written (with CVM_MT_AES_IV* instructions) between execution of this CVM_MT_AES_DEC_CBC1 and the execution of the immediately preceding CVM_MT_AES_ENC1 or CVM_MT_AES_DEC1.

AESKEY[AESKEYLEN<1:0>..0] must be loaded (with CVM_MT_AES_KEY* instructions) between this CVM_MT_AES_DEC_CBC1 and any prior CVM_MT_AES_ENC_CBC1 or CVM_MT_AES_ENC1 instruction.

AESKEY[AESKEYLEN<1:0>..0] must be predictable prior to this CVM_MT_AES_DEC_CBC1 instruction. This means that when AESKEYLEN<1:0> increases, AESKEY[AESKEYLEN<1:0>..X] must be written (with CVM_MT_AES_KEY* instructions) between the execution of this CVM_MT_AES_DEC_CBC1 and any immediately preceding CVM_MT_AES_DEC_CBC1 or CVM_MT_AES_DEC1 instruction that used the smaller AESKEYLEN<1:0> value of X.

Results are unpredictable when AESKEYLEN<1:0> is zero with the CVM_MT_AES_DEC_CBC1 instruction.
If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

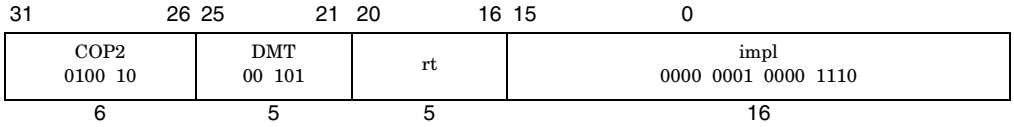
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      TIN[1:0]<63:0> = GPR[rt] || AESRESINP[0]
      AESRESINP[1:0] = AES_DEC(TIN[1:0], AESKEY[3:0], AESKEYLEN<1:0>) ⊕
                       AESIV[1:0]
      AESIV[1:0] = TIN[1:0]
      for(i = AESKEYLEN<1:0>+1; i < 4; i++)
         AESKEY[i] = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

AES Decrypt (part 1) CVM_MT_AES_DEC0



Format: DMTC2 rt, 0x010E **CVM**

Purpose:

To load AESRESINP[0] in preparation for a subsequent CVM_MT_AES_DEC1.

Description: `AESRESINP[0] = rt`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

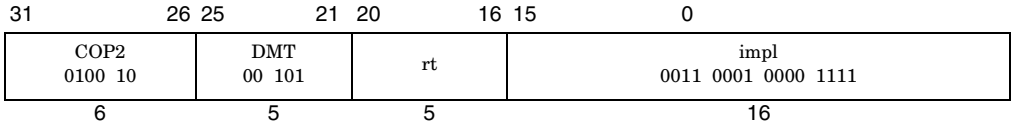
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      AESRESINP[0] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

AES Decrypt (part 2) — CVM_MT_AES_DEC1



Format: DMTC2 rt, 0x310F **CVM**

Purpose:

To perform an AES decrypt operation. GPR[rt] and AESRESINP[0] are the inputs to an AES decryption operation (using AESKEY and AESKEYLEN). AESRESINP is updated with the result of the decryption. AESIV and unused AESKEY locations become unpredictable.

Description:
```
AESRESINP[1:0] = AES_DEC((rt || AESRESINP[0]), AESKEY[3:0],
                         AESKEYLEN<1:0>)
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

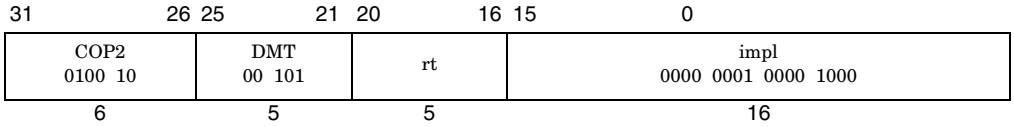
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      AESRESINP[1..0] = AES_DEC((GPR[rt] || AESRESINP[0]), AESKEY[3..0],
                        AESKEYLEN<1:0>)
      AESIV[1..0] = unpredictable
      for(i = AESKEYLEN<1:0>+1; i < 4; i++)
         AESKEY[i] = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

AES CBC Encrypt (part 1) — CVM_MT_AES_ENC_CBC0



Format: DMTC2 rt, 0x0108 **CVM**

Purpose:

To load AESRESINP[0] in preparation for a subsequent CVM_MT_AES_ENC_CBC1 instruction. The implementation may either deposit (rt XOR AESIV[0]) or rt into AESRESINP[0].

Description: `AESRESINP[0] = rt ⊕ AESIV[0], or  AESRESINP[0] = rt`

Restrictions:

AESIV[0] must be predictable prior to the CVM_MT_AES_ENC_CBC0 instruction. This means that AESIV[0] must be written between execution of this CVM_MT_AES_ENC_CBC0 and the execution of the immediately preceding CVM_MT_AES_ENC1 or CVM_MT_AES_DEC1.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

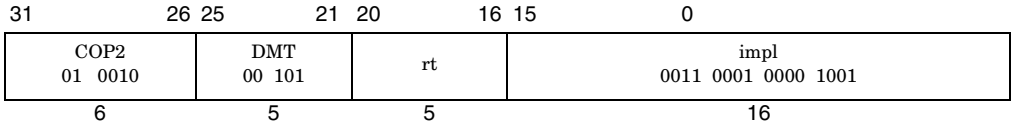
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      AESRESINP[0] = GPR[rt] ⊕ AESIV[0], or
      AESRESINP[0] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

AES CBC Encrypt (part 2) — CVM_MT_AES_ENC_CBC1



Format: DMTC2 rt, 0x3109 **CVM**

Purpose:

To perform an AES CBC encrypt operation. GPR[rt] XOR AESIV[1] is the second 64-bit input to an AES CBC encryption operation (using AESKEY and AESKEYLEN). Either AESRESINP[0] or (AESRESINP[0] XOR AESIV[0]) is the first 64-bit input to the AES operation, depending on whether the CVM_MT_AES_ENC_CBC0 implementation pre-XORed AESIV[0] or not, respectively. AESRESINP and AESIV are updated with the result. Unused AESKEY locations become unpredictable.

Description:

```
AESRESINP[1:0] = AESIV[1..0] =
      AES_ENC((rt ⊕ AESIV[1]) || AESRESINP[0], AESKEY[3:0],
      AESKEYLEN<1:0>), or
AESRESINP[1:0] = AESIV[1..0] =
      AES_ENC((rt || AESRESINP[0]) ⊕ AESIV[1:0], AESKEY[3..0],
      AESKEYLEN<1:0>)
```

Restrictions:

CVM_MT_AES_ENC_CBC0 (or CVM_MT_AES_RESINP0 in the save/restore case) must be executed between the execution of this CVM_MT_AES_ENC_CBC1 and the immediately preceding CVM_MT_AES_ENC_CBC1, CVM_MT_AES_ENC1, CVM_MT_AES_DEC_CBC1, or CVM_MT_AES_DEC1 instruction.

AESIV[1:0] must be predictable prior to this CVM_MT_AES_ENC_CBC1 instruction. This means that AESIV[1..0] must be written (with CVM_MT_AES_IV* instructions) between execution of this CVM_MT_AES_ENC_CBC1 and the execution of the immediately preceding CVM_MT_AES_ENC1 or CVM_MT_AES_DEC1.

AESKEY[AESKEYLEN<1:0>..0] must be loaded (with CVM_MT_AES_KEY* instructions) between this CVM_MT_AES_ENC_CBC1 and any prior CVM_MT_AES_DEC_CBC1 or CVM_MT_AES_DEC1 instruction.

AESKEY[AESKEYLEN<1:0>..0] must be predictable prior to this CVM_MT_AES_ENC_CBC1 instruction. This means that when AESKEYLEN<1:0> increases, AESKEY[AESKEYLEN<1:0>..X] (at least) must be written (with CVM_MT_AES_KEY* instructions) between the execution of this CVM_MT_AES_ENC_CBC1 and any immediately preceding CVM_MT_AES_ENC_CBC1 or CVM_MT_AES_ENC1 instruction that used the smaller AESKEYLEN<1:0> value of X.

Results are unpredictable when AESKEYLEN<1:0> is zero with the CVM_MT_AES_ENC_CBC1 instruction.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

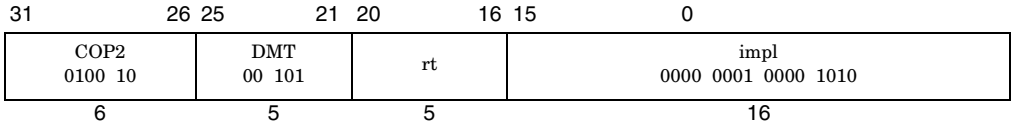
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      TIN[1..0]<63:0> = (GPR[rt] ⊕ AESIV[1]) || AESRESINP[0], or
      TIN[1..0]<63:0> = (GPR[rt] || AESRESINP[0]) ⊕ AESIV[1:0]
      AESRESINP[1:0] = AES_ENC(TIN[1:0], AESKEY[3:0], AESKEYLEN<1:0>)
      AESIV[1:0] = AESRESINP[1:0]
      for(i = AESKEYLEN<1:0>+1; i < 4; i++)
         AESKEY[i] = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

AES Encrypt (part 1) — CVM_MT_AES_ENC0



Format: DMTC2 rt, 0x010A **CVM**

Purpose:

To load AESRESINP[0] in preparation for a subsequent CVM_MT_AES_ENC1.

Description: `AESRESINP[0] = rt`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

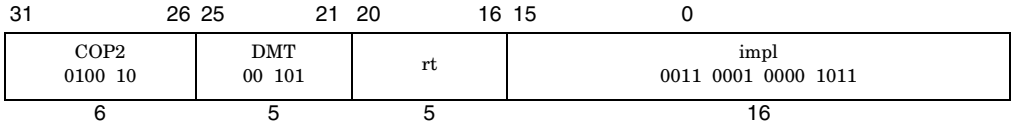
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      AESRESINP[0] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

AES Encrypt (part 2) **CVM_MT_AES_ENC1**



Format: DMTC2 rt, 0x310B **CVM**

Purpose:

To perform an AES encrypt operation. GPR[rt] and AESRESINP[0] are the inputs to an AES encryption operation (using AESKEY and AESKEYLEN). AESRESINP is updated with the result. AESIV and unused AESKEY locations become unpredictable.

Description:
```
AESRESINP[1:0] =
        AES_ENC((rt || AESRESINP[0]), AESKEY[3:0], AESKEYLEN<1:0>)
```

Restrictions:

CVM_MT_AES_ENC0 (or CVM_MT_AES_RESINP0 in the save/restore case) must be executed between the execution of this CVM_MT_AES_ENC1 and the immediately preceding CVM_MT_AES_ENC_CBC1, CVM_MT_AES_ENC1, CVM_MT_AES_DEC_CBC1, or CVM_MT_AES_DEC1 instruction.

AESKEY[AESKEYLEN<1:0>..0] must be loaded (with CVM_MT_AES_KEY* instructions) between this CVM_MT_AES_ENC1 and any prior CVM_MT_AES_DEC_CBC1 or CVM_MT_AES_DEC1 instruction.

AESKEY[AESKEYLEN<1:0>..0] must be predictable prior to this CVM_MT_AES_ENC1 instruction. This means that when AESKEYLEN<1:0> increases, AESKEY[AESKEYLEN<1:0>..X] must be written (with CVM_MT_AES_KEY* instructions) between the execution of this CVM_MT_AES_ENC_CBC1 and any immediately preceding CVM_MT_AES_ENC_CBC1 or CVM_MT_AES_ENC1 instruction that used the smaller AESKEYLEN<1:0> value of X.

Results are unpredictable when AESKEYLEN<1:0> is zero with the CVM_MT_AES_ENC1 instruction.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
```

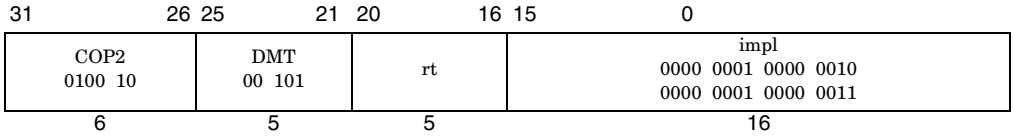
```
      AESRESINP[1:0] = AES_ENC((GPR[rt] || AESRESINP[0]), AESKEY[3:0],
                AESKEYLEN<1:0>)
      AESIV[1..0] = unpredictable
      for(i = AESKEYLEN<1:0>+1; i < 4; i++)
         AESKEY[i] = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load IV into AES Unit — CVM_MT_AES_IV



Format: DMTC2 rt, 0x0102 **CVM**

DMTC2 rt, 0x0103

Purpose:

To load AESIV.

Description:

```
AESIV[0] = rt // DMTC2 rt, 0x0102
AESIV[1] = rt // DMTC2 rt, 0x0103
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

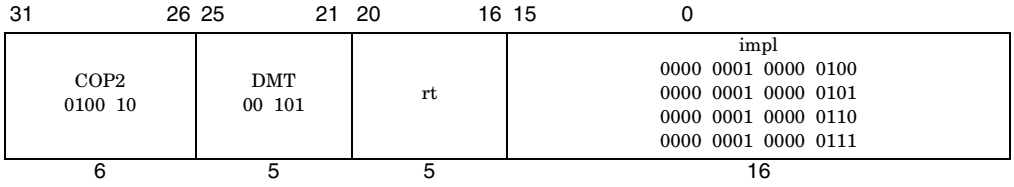
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0102: AESIV[0] = GPR[rt]
      case 0x0103: AESIV[1] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Key into AES Unit — CVM_MT_AES_KEY



Format: DMTC2 rt, 0x0104 **CVM**

DMTC2 rt, 0x0105

DMTC2 rt, 0x0106

DMTC2 rt, 0x0107

Purpose:

To load AESKEY.

Description:

```
AESKEY[0] = rt // DMTC2 rt, 0x0104
AESKEY[1] = rt // DMTC2 rt, 0x0105
AESKEY[2] = rt // DMTC2 rt, 0x0106
AESKEY[3] = rt // DMTC2 rt, 0x0107
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

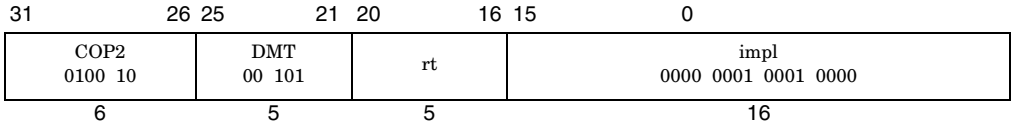
```
if IsCoprocessorEnabled(2) then
    if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
       SignalException(ReservedInstruction)
    else
       case 0x0104: AESKEY[0] = GPR[rt];
                    AESKEYLEN<2:0> = 0 // 0 is illegal key length
       case 0x0105: AESKEY[1] = GPR[rt];
                    AESKEYLEN<2:0> = 1 // 1 is 128-bit key
       case 0x0106: AESKEY[2] = GPR[rt];
                    AESKEYLEN<2:0> = 2 // 2 is 192-bit key
       case 0x0107: AESKEY[3] = GPR[rt];
                    AESKEYLEN<2:0> = 3 // 3 is 256-bit key
    endif
 else
    SignalException(CoprocessorUnusable, 2)
 endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Key Length into AES Unit — CVM_MT_AES_KEYLENGTH



Format: DMTC2 rt, 0x0110 **CVM**

Purpose:

To load AESKEYLEN.

Description: `AESLEYLEN<1:0> = rt<1:0>`

AESKEYLEN	Length of AES key
00	unpredictable
01	128 bits
10	192 bits
11	256 bits

Restrictions:

Results are unpredictable when an AESKEYLEN value of 0 is used.

Results are unpredictable when GPR[rt]<63:2> are not zero.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

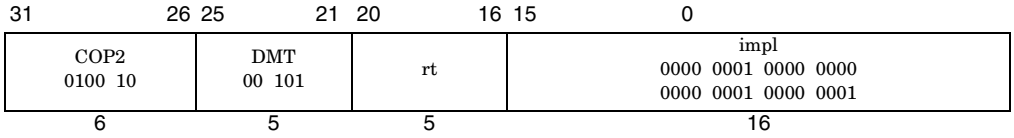
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      AESLEYLEN<1:0> = GPR[rt]<1:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Result/Input into AES Unit — CVM_MT_AES_RESINP



Format: DMTC2 rt, 0x0100 **CVM**

DMTC2 rt, 0x0101

Purpose:

To load AESRESINP.

Description:

```
AESRESINP[0]  = rt // DMTC2 rt, 0x0100
AESRESINP[1]  = rt // DMTC2 rt, 0x0101
```

Restrictions:

Either CVM_MF_AES_RESINP0 or CVM_MF_AES_RESINP1 must execute between this CVM_MT_AES_RESINP* instruction and the last CVM_MT_AES_ENC_CBC1, CVM_MT_AES_ENC1, CVM_MT_AES_DEC1, or CVM_MT_AES_DEC1 instruction.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

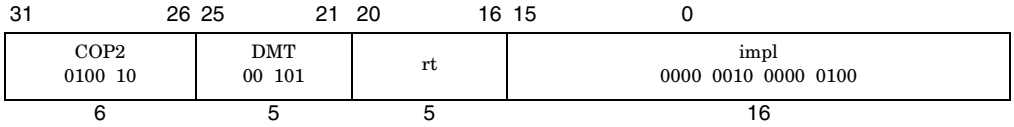
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0102: AESRESINP[0] = GPR[rt]
      case 0x0103: AESRESINP[1] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

CRC for a Byte — CVM_MT_CRC_BYTE



Format: DMTC2 rt, 0x0204 **CVM**

Purpose:

To calculate CRC for a byte. The least-significant byte of rt is the input to the CRC engine (using CRCIV and CRCPOLY). CRCIV is updated with the CRC result for the given CRCPOLY.

Description: `CRCIV<31:0> = CRC(rt<7:0>, CRCIV<31:0>, CRCPOLY<31:0>)`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

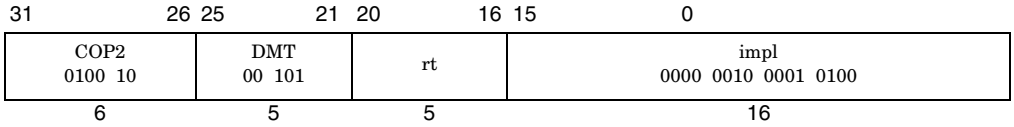
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      CRCIV<31:0> = CRC_BYTE(GPR[rt]<7:0>, CRCIV<31:0>, CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

CRC for a Byte Reflected — CVM_MT_CRC_BYTE_REFLECT



Format: DMTC2 rt, 0x0214 **CVM**

Purpose:

To calculate CRC for a byte. The bits the byte are reflected. The least-significant byte of rt is the input to the CRC engine (using CRCIV and CRCPOLY) after bit reflection. CRCIV is updated with the CRC result for the given CRCPOLY.

Description:
```
CRCIV<31:0> = CRC(byte_bit_reflect(rt<7:0>), CRCIV<31:0>,
                  CRCPOLY<31:0>)
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

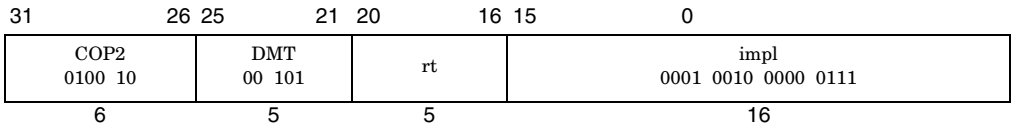
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      CRCIV<31:0> = CRC_BYTE(byte_bit_reflect(GPR[rt]<7:0>),
                             CRCIV<31:0>, CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

CRC for a Double-word — CVM_MT_CRC_DWORD



Format: DMTC2 rt, 0x1207 **CVM**

Purpose:

To calculate CRC for a double-word. The eight bytes of rt is the input to the CRC engine (using CRCIV and CRCPOLY). CRCIV is updated with the CRC result for the given CRCPOLY.

Description: `CRCIV<31:0> = CRC(rt<63:0>, CRCIV<31:0>, CRCPOLY<31:0>)`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

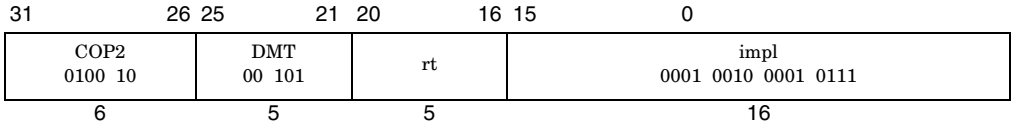
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      for(i = 0; i < 8; i++)
         CRCIV<31:0> = CRC_BYTE(GPR[rt]<63-i*8:56-i*8>, CRCIV<31:0>,
                       CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

CRC for a Double-word Reflected CVM_MT_CRC_DWORD_REFLECT



Format: DMTC2 rt, 0x1217 **CVM**

Purpose:

To calculate CRC for a double-word. The bits in each byte are reflected. The eight bytes of rt is the input to the CRC engine (using CRCIV and CRCPOLY) after bit reflection. CRCIV is updated with the CRC result for the given CRCPOLY.

Description:
```
CRCIV<31:0> = CRC(byte_bit_reflect(rt<63:0>), CRCIV<31:0>,
                  CRCPOLY<31:0>)
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

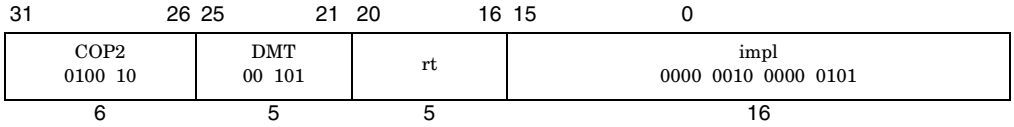
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      for(i = 0; i < 8; i++)
         CRCIV<31:0> = CRC_BYTE(byte_bit_reflect(GPR[rt]<63-i*8:56-i*8>),
                       CRCIV<31:0>, CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

CRC for a Halfword — CVM_MT_CRC_HALF



Format: DMTC2 rt, 0x0205 **CVM**

Purpose:

To calculate CRC for a halfword. The least-significant two bytes of rt is the input to the CRC engine (using CRCIV and CRCPOLY). CRCIV is updated with the CRC result for the given CRCPOLY.

Description: `CRCIV<31:0> = CRC(rt<15:0>, CRCIV<31:0>, CRCPOLY<31:0>)`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

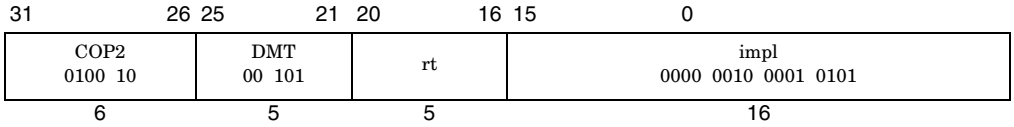
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      for(i = 0; i < 2; i++)
         CRCIV<31:0> = CRC_BYTE(GPR[rt]<15-i*8:8-i*8>, CRCIV<31:0>,
                       CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

CRC for a Halfword Reflected — CVM_MT_CRC_HALF_REFLECT



Format: DMTC2 rt, 0x0215 **CVM**

Purpose:

To calculate CRC for a halfword. The bits in each byte are reflected. The least-significant two bytes of rt is the input to the CRC engine (using CRCIV and CRCPOLY) after bit reflection. CRCIV is updated with the CRC result for the given CRCPOLY.

Description:
```
CRCIV<31:0> = CRC(byte_bit_reflect(rt<15:0>), CRCIV<31:0>,
                  CRCPOLY<31:0>)
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

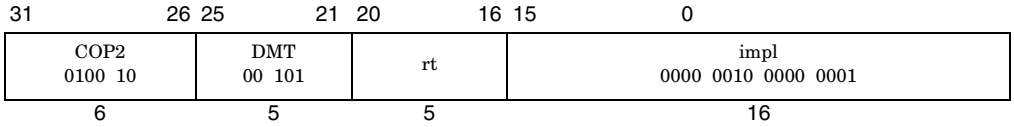
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      for(i = 0; i < 2; i++)
         CRCIV<31:0> = CRC_BYTE(byte_bit_reflect(GPR[rt]<15-i*8:8-i*8>),
                       CRCIV<31:0>, CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load IV into CRC Unit — CVM_MT_CRC_IV



Format: DMTC2 rt, 0x0201 **CVM**

Purpose:

To load a value into CRCIV.

Description: `CRCIV<31:0> = rt<31:0>`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      CRCIV<31:0> = GPR[rt]<31:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

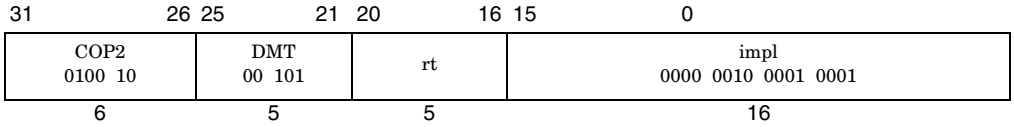
Coprocessor Unusable

Reserved Instruction

Programming Notes:

CRC polynomials smaller than 32-bits should be supported by placing the IV in the most-significant CRCIV bits. For example, the IV with a 16-bit polynomial should reside in CRCIV<31:16>, and CRCIV<15:0> should be zero.

Load IV into CRC Unit Reflected — CVM_MT_CRC_IV_REFLECT



Format: DMTC2 rt, 0x0211 **CVM**

Purpose:

To load a value into CRCIV. The bits in each byte are reflected.

Description: `CRCIV<31:0> = byte_bit_reflect(rt<31:0>)`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      CRCIV<31:0> = byte_bit_reflect(GPR[rt]<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

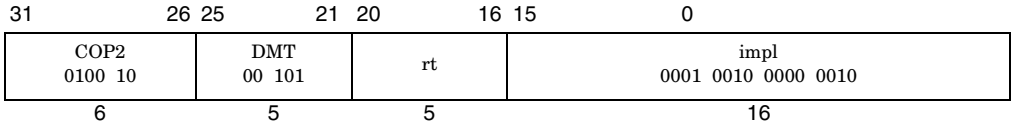
Coprocessor Unusable

Reserved Instruction

Programming Notes:

CRC polynomials smaller than 32-bits should be supported by placing the IV in the most-significant CRCIV bits. For example, the IV with a 16-bit polynomial should reside in CRCIV<31:16>, and CRCIV<15:0> should be zero.

Load Length into CRC Unit — CVM_MT_CRC_LEN



Format: DMTC2 rt, 0x1202 **CVM**

Purpose:

To load a value into CRCLEN.

Description: `CRCLEN<3:0> = rt<3:0>`

Restrictions:

The results of CVM_MT_CRC_LEN are unpredictable when GPR[rt]<63:4> is not zero.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

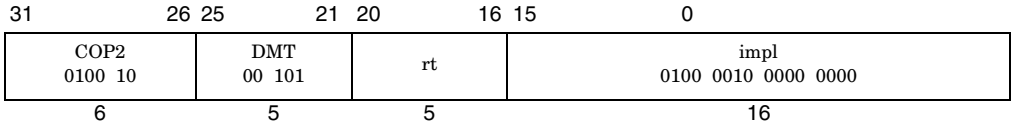
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      CRCLEN<3:0> = GPR[rt]<3:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Polynomial into CRC Unit — CVM_MT_CRC_POLYNOMIAL



Format: DMTC2 rt, 0x4200 **CVM**

Purpose:

To load a value into CRCPOLY.

Description: `CRCPOLY<31:0> = rt<31:0>`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      CRCPOLY<31:0> = GPR[rt]<31:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

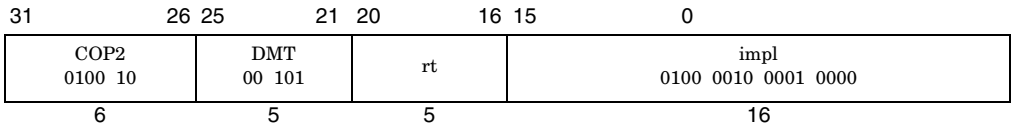
Coprocessor Unusable

Reserved Instruction

Programming Notes:

CRC polynomials smaller than 32-bits should be supported by placing the polynomial in the most-significant CRCPOLY bits. For example, a 16-bit polynomial should reside in CRCPOLY<31:16>, and CRCPOLY<15:0> should be zero.

Load Polynomial into CRC Unit Reflected	CVM_MT_CRC_POLYNOMIAL_REFLECT



Format: DMTC2 rt, 0x4210 **CVM**

Purpose:

To load a value into CRCPOLY. The bits in each byte are reflected.

Description: `CRCPOLY<31:0> = byte_bit_reflect(rt<31:0>)`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      CRCPOLY<31:0> = byte_bit_reflect(GPR[rt]<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

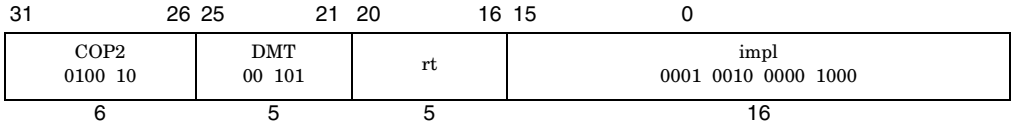
Coprocessor Unusable

Reserved Instruction

Programming Notes:

CRC polynomials smaller than 32-bits should be supported by placing the polynomial in the most-significant CRCPOLY bits. For example, a 16-bit polynomial should reside in CRCPOLY<31:16>, and CRCPOLY<15:0> should be zero.

CRC for Variable Length — CVM_MT_CRC_VAR



Format: DMTC2 rt, 0x1208 **CVM**

Purpose:

To calculate CRC for CRCLEN bytes. The CRCLEN most-significant bytes of rt is the input to the CRC engine (using CRCIV and CRCPOLY). CRCIV is updated with the CRC result for the given CRCPOLY.

Description:
```
CRCIV<31:0> = CRC(rt<63:0>, CRCLEN<3:0>, CRCIV<31:0>,
                  CRCPOLY<31:0>)
```

Restrictions:

The results of CVM_MT_CRC_VAR are unpredictable when CRCLEN<3:0> is larger than eight.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

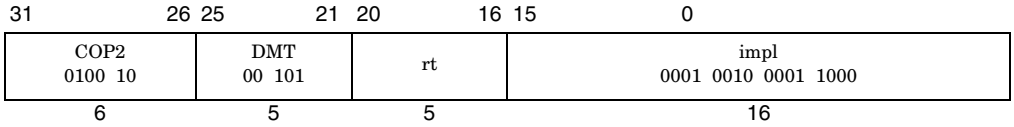
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      len = CRCLEN<3:0>;
      for(i = 0; i < len; i++)
         CRCIV<31:0> = CRC_BYTE(GPR[rt]<63-i*8:56-i*8>, CRCIV<31:0>,
                       CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

CRC for Variable Length Reflected **CVM_MT_CRC_VAR_REFLECT**



Format: DMTC2 rt, 0x1218 **CVM**

Purpose:

To calculate CRC for length bytes. The bits in each byte are reflected. The CRCLEN most-significant bytes of rt is the input to the CRC engine (using CRCIV and CRCPOLY) after bit reflection. CRCIV is updated with the CRC result for the given CRCPOLY.

Description:
```
CRCIV<31:0> = CRC(byte_bit_reflect(rt<63:0>), CRCLEN<3:0>,
                  CRCIV<31:0>, CRCPOLY<31:0>)
```

Restrictions:

The results of CVM_MT_CRC_VAR_REFLECT are unpredictable when CRCLEN<3:0> is larger than eight.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

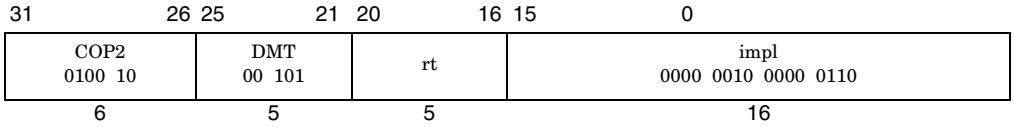
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      len = CRCLEN<3:0>;
      for(i = 0; i < len; i++)
         CRCIV<31:0> = CRC_BYTE(byte_bit_reflect(GPR[rt]<63-i*8:56-i*8>),
                       CRCIV<31:0>, CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

CRC for a Word — CVM_MT_CRC_WORD



Format: DMTC2 rt, 0x0206 **CVM**

Purpose:

To calculate CRC for a word. The least-significant four bytes of rt is the input to the CRC engine (using CRCIV and CRCPOLY). CRCIV is updated with the CRC result for the given CRCPOLY.

Description: `CRCIV<31:0> = CRC(rt<31:0>, CRCIV<31:0>, CRCPOLY<31:0>)`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

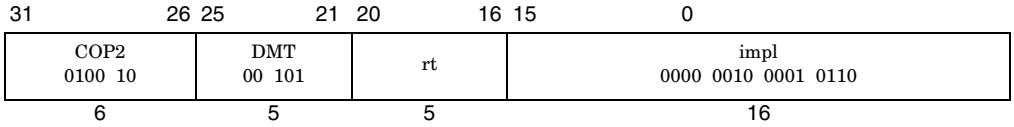
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      for(i = 0; i < 4; i++)
         CRCIV<31:0> = CRC_BYTE(GPR[rt]<31-i*8:24-i*8>, CRCIV<31:0>,
                       CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

CRC for a Word Reflected — CVM_MT_CRC_WORD_REFLECT



Format: DMTC2 rt, 0x0216 **CVM**

Purpose:

To calculate CRC for a word. The bits in each byte are reflected. The least-significant four bytes of rt is the input to the CRC engine (using CRCIV and CRCPOLY) after bit reflection. CRCIV is updated with the CRC result for the given CRCPOLY.

Description:
```
CRCIV<31:0> = CRC(byte_bit_reflect(rt<31:0>), CRCIV<31:0>,
                  CRCPOLY<31:0>)
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

Operation:

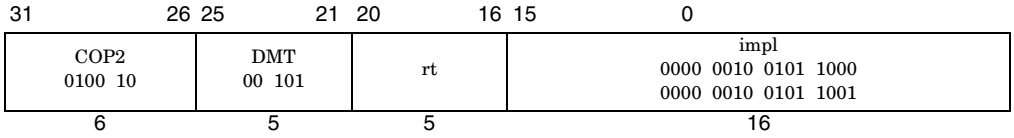
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled())) then
      SignalException(ReservedInstruction)
   else
      for(i = 0; i < 4; i++)
         CRCIV<31:0> = CRC_BYTE(byte_bit_reflect(GPR[rt]<31-i*8:24-i*8>),
                       CRCIV<31:0>, CRCPOLY<31:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Multiplier into GFM Unit — CVM_MT_GFM_MUL



Format: DMTC2 rt, 0x0258 **CVM**

DMTC2 rt, 0x0259

Purpose:

To load GFMMUL.

Description:

```
GFMMUL[0] = rt  // DMTC2 rt, 0x0258
GFMMUL[1] = rt  // DMTC2 rt, 0x0259
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x258: GFMMUL[0] = GPR[rt]
      case 0x259: GFMMUL[1] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

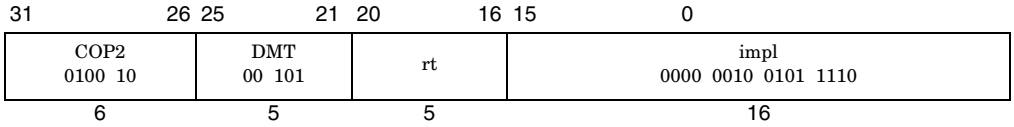
Coprocessor Unusable

Reserved Instruction

Programming Notes:

For GF(2^{64}) multiplication, GFMMUL[1] should be set to zero and GFMMUL[0] should contain the 64-bit multiplier.

Load Polynomial into GFM Unit — CVM_MT_GFM_POLY



Format: DMTC2 rt, 0x025E **CVM**

Purpose:

To load GFMPOLY.

Description: `GFMPOLY<15:0> = rt<15:0>`

Restrictions:

Results are unpredictable when GPR[rt]<63:16> are not zero.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      GFMPOLY<15:0> = GPR[rt]<15:0>
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

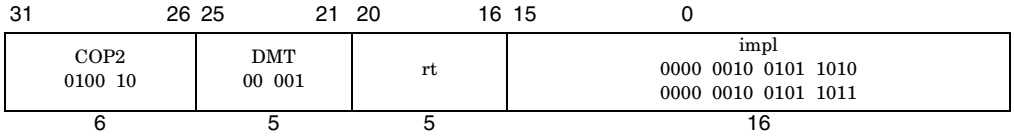
Programming Notes:

There are at least two interesting GFMPOLY values, corresponding to the two polynomials covered in “The Galois/Counter Mode of Operation” by McGrew and Viega:

$GFMPOLY = 0xE100$ for $1 + \alpha + \alpha^2 + \alpha^7 + \alpha^{128}$ in $GF(2^{128})$

$GFMPOLY = 0x00D8$ for $1 + \alpha + \alpha^3 + \alpha^4 + \alpha^{64}$ in $GF(2^{64})$

Load Result/Input into GFM Unit — CVM_MT_GFM_RESINP



Format: DMTC2 rt, 0x025A **CVM**

DMTC2 rt, 0x025B

Purpose:

To load GFMRESINP.

Description:

```
GFMRESINP[0] = rt  // DMFC2 rt, 0x025A
GFMRESINP[1] = rt  // DMFC2 rt, 0x025B
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x25A: GFMRESINP[0] = GPR[rt]
      case 0x25B: GFMRESINP[1] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

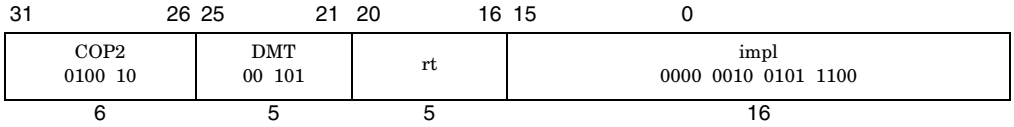
Coprocessor Unusable

Reserved Instruction

Programming Notes:

For GF(2^{64}) multiplication, GFMRESINP[0] should be set to zero and GFMRESINP[1] should contain the 64-bit value.

XOR into GFM Unit — CVM_MT_GFM_XOR0



Format: DMTC2 rt, 0x025C **CVM**

Purpose:

To XOR the contents of GFMRESINP[0] with rt.

Description: `GFMRESINP[0] = GFMRESINP[0] ⊕ rt`

Restrictions:

Results are unpredictable when GPR[rt]<63:16> are not zero.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      GFMPRESINP[0] = GFMPRESINP[0] ⊕ GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

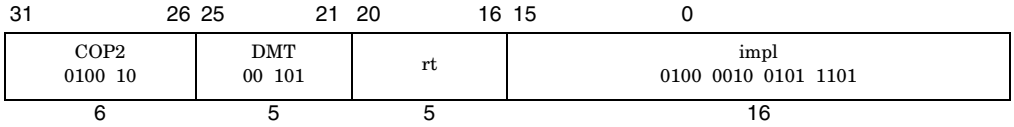
Coprocessor Unusable

Reserved Instruction

Programming Notes:

CVM_MT_GFM_XOR0 should not be used for GF(2^{64}) multiplications. (GFMRESINP[0] should be zero.)

XOR and GF Multiply — CVM_MT_GFM_XORMUL1



Format: DMTC2 rt, 0x425D **CVM**

Purpose:

To XOR the contents of GFMRESINP[1] with rt and perform a GF(2^{128}) multiplication of GFMRESINP and GFMMUL (using the polynomial selected by GFMPOLY), depositing the result in GFMRESINP.

Description:
```
GFMRESINP[1] = GFMRESINP[1] ⊕ rt
GFMRESINP[1:0] = GFMMULT(GFMRESINP, GFMMUL, GFMPOLY)
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to Coprocessor 2 is enabled but access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      Z[1:0]<63:0> = 0
      V[0]<63:0> = doubleword_bit_reflect(GFMRESINP[0]<63:0>)
      V[1]<63:0> = doubleword_bit_reflect(GFMRESINP[1]<63:0>)
      Y[0]<63:0> = doubleword_bit_reflect(GFMMUL[0]<63:0>)
      Y[1]<63:0> = doubleword_bit_reflect(GFMMUL[1]<63:0>)
      T<15:0> = halfword_bit_reflect(GFMPOLY<15:0>);
      POLY[0]<63:0> = 0^56 || T<7:0>
      POLY[1]<63:0> = 0^56 || T<15:8>
      for(i = 0; i < 128; i++) {
         if(Y[i >> 6]<i & 0x3F>)
            Z[1:0] = Z[1:0] ⊕ V[1:0]
         R[1:0]<63:0> = V[1]<63> ? POLY[1:0]<63:0> : 0^128
         V[1]<63:0> || V[0]<63:0> = V[1]<62:0> || V[0]<63:0> || 0^1
         V[1:0] = V[1:0] ⊕ R[1:0]
      }
      GFMRESINP[0]<63:0> = doubleword_bit_reflect(Z[0]<63:0>)
      GFMRESINP[1]<63:0> = doubleword_bit_reflect(Z[1]<63:0>)
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

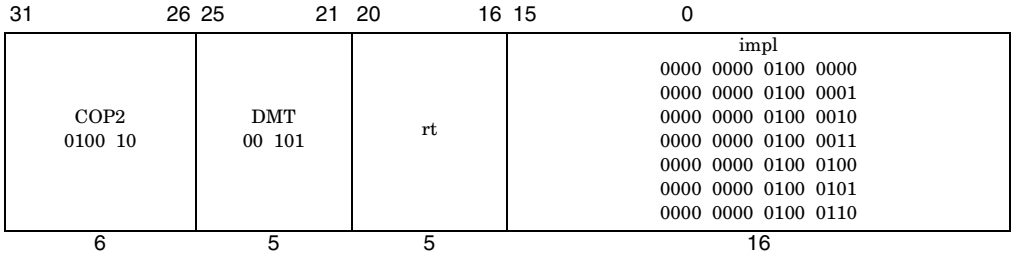
Reserved Instruction

Programming Notes:

CVM_MT_GFM_XORMUL1 should be used to initiate GF(2^{64}) multiplications, as well as GF(2^{128}). GPR[rt] is XORed with GFMRESINP[1] to create the actual 64-bit input for the GF(2^{64}) multiplication. Assuming that GFMRESINP[0] and GFMMUL[1] are zero, GFMRESINP[1] will contain the GF(2^{64}) multiplication result:

$$\text{GFMRESINP[1]} = (\text{GPR[rt]} \oplus \text{GFMRESINP[1]}) \times \text{GFMMUL[0]}$$

Load Data into HSH Unit (narrow mode) CVM_MT_HSH_DAT



Format: DMTC2 rt, 0x0040 **CVM**

DMTC2 rt, 0x0041

DMTC2 rt, 0x0042

DMTC2 rt, 0x0043

DMTC2 rt, 0x0044

DMTC2 rt, 0x0045

DMTC2 rt, 0x0046

Purpose:

To load values into HASHDAT via the narrow mechanism. The MD5, SHA-1, and SHA-256 algorithms use the narrow HASHDAT format. The most-significant and least-significant 32 bits of rt are written into the least-significant 32 bits of separate 64-bit double-words HASHDAT.

Description:

```
HASHDAT[0]<31:0>  || HASHDAT[1]<31:0>   = rt // DMTC2 rt,0x0040
HASHDAT[2]<31:0>  || HASHDAT[3]<31:0>   = rt // DMTC2 rt,0x0041
HASHDAT[4]<31:0>  || HASHDAT[5]<31:0>   = rt // DMTC2 rt,0x0042
HASHDAT[6]<31:0>  || HASHDAT[7]<31:0>   = rt // DMTC2 rt,0x0043
HASHDAT[8]<31:0>  || HASHDAT[9]<31:0>   = rt // DMTC2 rt,0x0044
HASHDAT[10]<31:0> || HASHDAT[11]<31:0>  = rt // DMTC2 rt,0x0045
HASHDAT[12]<31:0> || HASHDAT[13]<31:0>  = rt // DMTC2 rt,0x0046
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0040: HASHDAT[0]<31:0>       = GPR[rt]<63:32>
                   HASHDAT[1]<31:0>       = GPR[rt]<31:0>
                   HASHDAT[0..1]<63:32>   = unpredictable
      case 0x0041: HASHDAT[2]<31:0>       = GPR[rt]<63:32>
                   HASHDAT[3]<31:0>       = GPR[rt]<31:0>
                   HASHDAT[2..3]<63:32>   = unpredictable
```

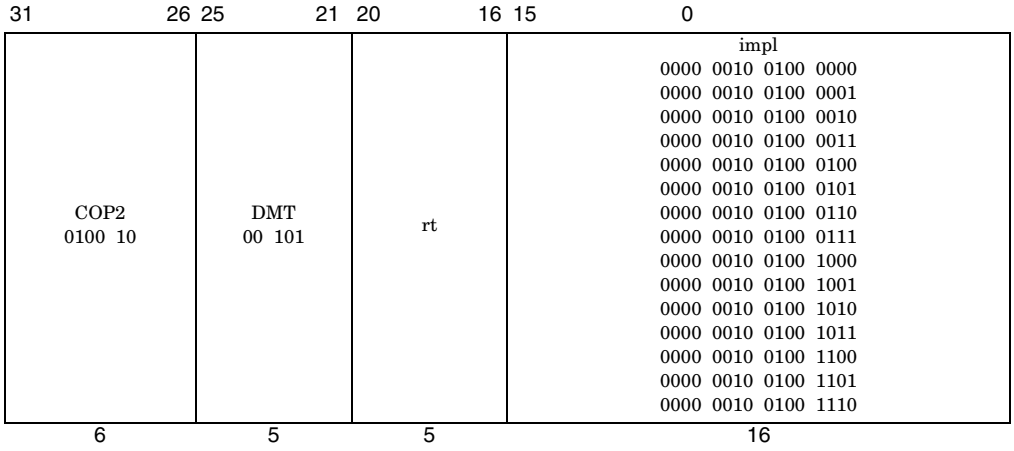
```
      case 0x0042: HASHDAT[4]<31:0>       = GPR[rt]<63:32>
                   HASHDAT[5]<31:0>       = GPR[rt]<31:0>
                   HASHDAT[4..5]<63:32>   = unpredictable
      case 0x0043: HASHDAT[6]<31:0>       = GPR[rt]<63:32>
                   HASHDAT[7]<31:0>       = GPR[rt]<31:0>
                   HASHDAT[6..7]<63:32>   = unpredictable
      case 0x0044: HASHDAT[8]<31:0>       = GPR[rt]<63:32>
                   HASHDAT[9]<31:0>       = GPR[rt]<31:0>
                   HASHDAT[8..9]<63:32>   = unpredictable
      case 0x0045: HASHDAT[10]<31:0>      = GPR[rt]<63:32>
                   HASHDAT[11]<31:0>      = GPR[rt]<31:0>
                   HASHDAT[10..11]<63:32> = unpredictable
      case 0x0046: HASHDAT[12]<31:0>      = GPR[rt]<63:32>
                   HASHDAT[13]<31:0>      = GPR[rt]<31:0>
                   HASHDAT[12..13]<63:32> = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load Data into HSH Unit (wide mode) CVM_MT_HSH_DATW



Format: DMTC2 rt, 0x0240 **CVM**

DMTC2 rt, 0x0241
DMTC2 rt, 0x0242
DMTC2 rt, 0x0243
DMTC2 rt, 0x0244
DMTC2 rt, 0x0245
DMTC2 rt, 0x0246
DMTC2 rt, 0x0247
DMTC2 rt, 0x0248
DMTC2 rt, 0x0249
DMTC2 rt, 0x024A
DMTC2 rt, 0x024B
DMTC2 rt, 0x024C
DMTC2 rt, 0x024D
DMTC2 rt, 0x024E

Purpose:

To load values into HASHDAT via the wide mechanism. The SHA-512 algorithm uses the wide HASHDAT format.

Description:

```
HASHDAT[0]  = rt // DMTC2 rt,0x0240
HASHDAT[1]  = rt // DMTC2 rt,0x0241
HASHDAT[2]  = rt // DMTC2 rt,0x0242
HASHDAT[3]  = rt // DMTC2 rt,0x0243
HASHDAT[4]  = rt // DMTC2 rt,0x0244
HASHDAT[5]  = rt // DMTC2 rt,0x0245
HASHDAT[6]  = rt // DMTC2 rt,0x0246
HASHDAT[7]  = rt // DMTC2 rt,0x0247
HASHDAT[8]  = rt // DMTC2 rt,0x0248
HASHDAT[9]  = rt // DMTC2 rt,0x0249
HASHDAT[10] = rt // DMTC2 rt,0x024A
HASHDAT[11] = rt // DMTC2 rt,0x024B
HASHDAT[12] = rt // DMTC2 rt,0x024C
HASHDAT[13] = rt // DMTC2 rt,0x024D
HASHDAT[14] = rt // DMTC2 rt,0x024E
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

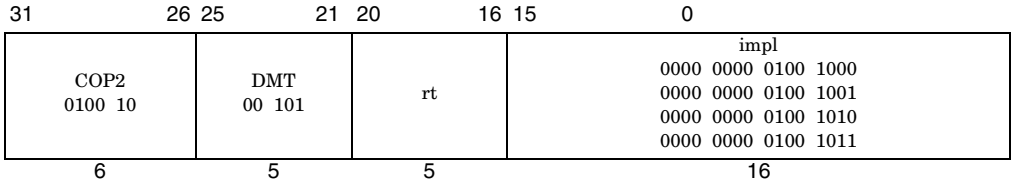
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0240: HASHDAT[ 0] = GPR[rt]
      case 0x0241: HASHDAT[ 1] = GPR[rt]
      case 0x0242: HASHDAT[ 2] = GPR[rt]
      case 0x0243: HASHDAT[ 3] = GPR[rt]
      case 0x0243: HASHDAT[ 4] = GPR[rt]
      case 0x0245: HASHDAT[ 5] = GPR[rt]
      case 0x0246: HASHDAT[ 6] = GPR[rt]
      case 0x0247: HASHDAT[ 7] = GPR[rt]
      case 0x0248: HASHDAT[ 8] = GPR[rt]
      case 0x0249: HASHDAT[ 9] = GPR[rt]
      case 0x024a: HASHDAT[10] = GPR[rt]
      case 0x024b: HASHDAT[11] = GPR[rt]
      case 0x024c: HASHDAT[12] = GPR[rt]
      case 0x024d: HASHDAT[13] = GPR[rt]
      case 0x024e: HASHDAT[14] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load IV into HSH Unit (narrow mode) CVM_MT_HSH_IV



Format: DMTC2 rt, 0x0048 **CVM**

DMTC2 rt, 0x0049

DMTC2 rt, 0x004A

DMTC2 rt, 0x004B

Purpose:

To load values into HASHIV in the narrow mode. MD5, SHA-1, and SHA-256 use the narrow mode. The least-significant 32 bits of separate 64-bit HASHDAT double-words are written with the most-significant and least-significant 32 bits of rt.

Description:

```
HASHDAT[0]<31:0>  || HASHDAT[1]<31:0>  = rt // DMTC2 rt,0x0048
HASHDAT[2]<31:0>  || HASHDAT[3]<31:0>  = rt // DMTC2 rt,0x0049
HASHDAT[4]<31:0>  || HASHDAT[5]<31:0>  = rt // DMTC2 rt,0x004A
HASHDAT[6]<31:0>  || HASHDAT[7]<31:0>  = rt // DMTC2 rt,0x004B
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

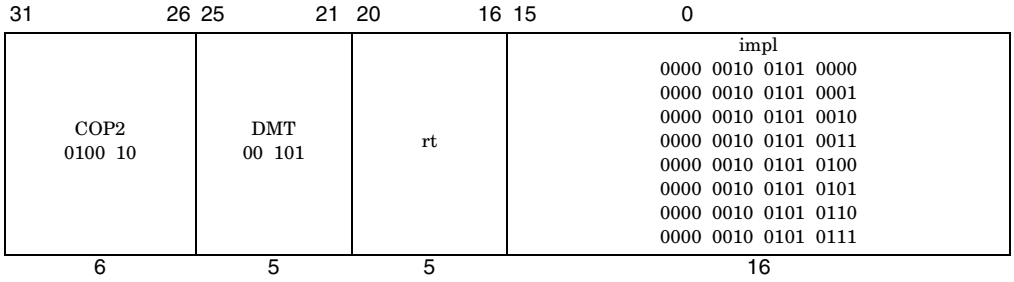
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0048: HASHIV[0]<31:0>     = GPR[rt]<63:32>
                   HASHIV[1]<31:0>     = GPR[rt]<31:0>
                   HASHIV[1..0]<63:32> = unpredictable
      case 0x0049: HASHIV[2]<31:0>     = GPR[rt]<63:32>
                   HASHIV[3]<31:0>     = GPR[rt]<31:0>
                   HASHIV[3..2]<63:32> = unpredictable
      case 0x004a: HASHIV[4]<31:0>     = GPR[rt]<63:32>
                   HASHIV[5]<31:0>     = GPR[rt]<31:0>
                   HASHIV[5..4]<63:32> = unpredictable
      case 0x004a: HASHIV[6]<31:0>     = GPR[rt]<63:32>
                   HASHIV[7]<31:0>     = GPR[rt]<31:0>
                   HASHIV[7..6]<63:32> = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Load IV into HSH Unit (wide mode) CVM_MT_HSH_IVW



Format: DMTC2 rt, 0x0250 **CVM**

DMTC2 rt, 0x0251

DMTC2 rt, 0x0252

DMTC2 rt, 0x0253

DMTC2 rt, 0x0254

DMTC2 rt, 0x0255

DMTC2 rt, 0x0256

DMTC2 rt, 0x0257

Purpose:

To load values into HASHIV in the wide mode. SHA-512 uses the wide mode.

Description:

```
HASHIV[0] = rt // DMTC2 rt,0x0250
HASHIV[1] = rt // DMTC2 rt,0x0251
HASHIV[2] = rt // DMTC2 rt,0x0252
HASHIV[3] = rt // DMTC2 rt,0x0253
HASHIV[4] = rt // DMTC2 rt,0x0254
HASHIV[5] = rt // DMTC2 rt,0x0255
HASHIV[6] = rt // DMTC2 rt,0x0256
HASHIV[7] = rt // DMTC2 rt,0x0257
```

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      case 0x0250: HASHIV[0] = GPR[rt]
      case 0x0251: HASHIV[1] = GPR[rt]
      case 0x0252: HASHIV[2] = GPR[rt]
      case 0x0253: HASHIV[3] = GPR[rt]
```

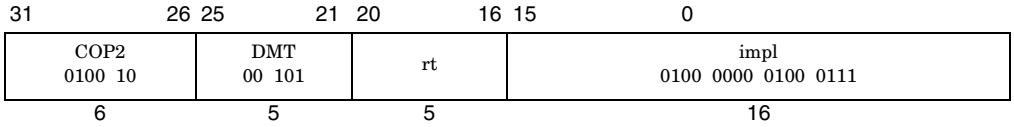
```
      case 0x0254: HASHIV[4] = GPR[rt]
      case 0x0255: HASHIV[5] = GPR[rt]
      case 0x0256: HASHIV[6] = GPR[rt]
      case 0x0257: HASHIV[7] = GPR[rt]
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

MD5 Hash **CVM_MT_HSH_STARTMD5**



Format: DMTC2 rt, 0x4047 **CVM**

Purpose:

To start an MD5 hash. The least-significant 32 bits of the first 14 HASHDAT words together with rt is hashed using the IV from the least-significant 32 bits of the first four HASHIV words. The least-significant 32 bits of the first four HASHIV words are updated with the result (in narrow mode).

Description:
```
HASHIV[3:0]<31:0> = MD5(rt || HASHDAT[13:0]<31:0>,
                        HASHIV[3:0]<31:0>)
```

Restrictions:

HASHDAT[0..13]<31:0> must be predictable. This means that HASHDAT[0..13]<31:0> must be written between execution of this CVM_MT_HSH_STARTMD5 and the execution of the immediately preceding CVM_MT_HSH_STARTMD5, CVM_MT_HSH_STARTSHA, CVM_MT_HSH_STARTSHA256, or CVM_MT_HSH_STARTSHA512.

All of HASHIV[3:0]<31:0> must be predictable prior to this CVM_MT_HSH_STARTMD5.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      for(i = 0; i < 2; i++)
         TIV[i]<63:0> = HASHIV[2*i]<31:0> || HASHIV[2*i+1]<31:0>
      for(i = 0; i < 7; i++)
         TDAT[i]<63:0> = HASHDAT[2*i]<31:0> || HASHDAT[2*i+1]<31:0>
      TDAT[7]<63:0> = GPR[rt]
      TIV = MD5(TDAT, TIV)
      for(i = 0; i < 2; i++) {
         HASHIV[2*i]<31:0>   = TIV[i]<63:32>
         HASHIV[2*i+1]<31:0> = TIV[i]<31:0>
      }
      HASHIV[3..0]<63:32> = unpredictable
      HASHDAT[14..0]<63:0> = unpredictable
   endif
```

```
  else
     SignalException(CoprocessorUnusable, 2)
  endif
```

Exceptions:

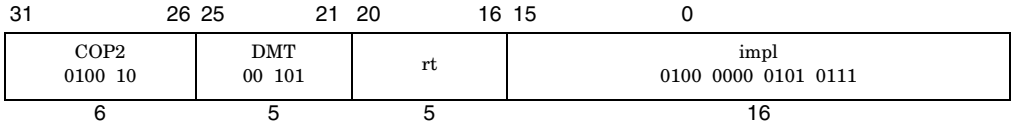
Coprocessor Unusable

Reserved Instruction

Notes:

The MD5 algorithm is little-endian and the HASHIV and HASHDAT data are big-endian, so both HASHIV and HASHDAT are byte-swapped in and out of the MD5() function above.

SHA-1 Hash **CVM_MT_HSH_STARTSHA**



Format: DMTC2 rt, 0x4057 **CVM**

Purpose:

To start a SHA-1 hash. The least-significant 32 bits of the first 14 HASHDAT words together with rt is hashed using the IV from the least-significant 32 bits of the first five HASHIV words. The least-significant 32 bits of the first five HASHIV words are updated with the result (in narrow mode).

Description:
```
HASHIV[4:0]<31:0> = SHA-1(rt || HASHDAT[13:0]<31:0>,
                          HASHIV[4:0]<31:0>)
```

Restrictions:

HASHDAT[13:0]<31:0> must be predictable. This means that HASHDAT[13:0]<31:0> must be written between execution of this CVM_MT_HSH_STARTSHA and the execution of the immediately preceding CVM_MT_HSH_STARTMD5, CVM_MT_HSH_STARTSHA, CVM_MT_HSH_STARTSHA256, or CVM_MT_HSH_STARTSHA512.

All of HASHIV[4:0]<31:0> must be predictable prior to this CVM_MT_HSH_STARTSHA.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      for(i = 0; i < 2; i++)
         TIV[i]<63:0> = HASHIV[2*i]<31:0> || HASHIV[2*i+1]<31:0>
      TIV[2]<63:0> = TIV[4]<31:0> || 0^32
      for(i = 0; i < 7; i++)
         TDAT[i]<63:0> = HASHDAT[2*i]<31:0> || HASHDAT[2*i+1]<31:0>
      TDAT[7]<63:0> = GPR[rt]
      TIV = SHA-1(TDAT, TIV)
      for(i = 0; i < 2; i++) {
         HASHIV[2*i]<31:0>   = TIV[i]<63:32>
         HASHIV[2*i+1]<31:0> = TIV[i]<31:0>
      }
      HASHIV[4]<31:0> = TIV[2]<63:32>
      HASHIV[4..0]<63:32> = unpredictable
      HASHDAT[14..0]<63:0> = unpredictable
```

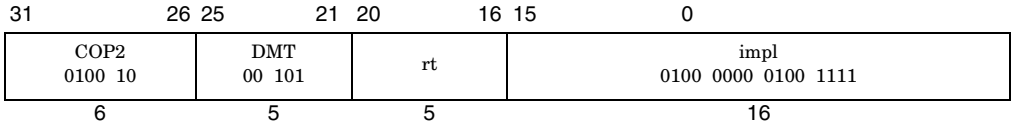
```
    endif
  else
    SignalException(CoprocessorUnusable, 2)
  endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

SHA-256 Hash **CVM_MT_HSH_STARTSHA256**



Format: DMTC2 rt, 0x404F **CVM**

Purpose:

To start a SHA-256 hash. The least-significant 32 bits of the first 14 HASHDAT words together with rt is hashed using the IV from the least-significant 32 bits of the first five HASHIV words. The least-significant 32 bits of the first five HASHIV words are updated with the result (in narrow mode).

Description:
```
HASHIV[7:0]<31:0> = SHA-256(rt || HASHDAT[13:0]<31:0>,
                            HASHIV[7:0]<31:0>)
```

Restrictions:

HASHDAT[13:0]<31:0> must be predictable. This means that HASHDAT[13:0]<31:0> must be written between execution of this CVM_MT_HSH_STARTSHA256 and the execution of the immediately preceding CVM_MT_HSH_STARTMD5, CVM_MT_HSH_STARTSHA, CVM_MT_HSH_STARTSHA256, or CVM_MT_HSH_STARTSHA512.

All of HASHIV[7:0]<31:0> must be predictable prior to this CVM_MT_HSH_STARTSHA256.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      for(i = 0; i < 4; i++)
         TIV[i]<63:0> = HASHIV[2*i]<31:0> || HASHIV[2*i+1]<31:0>
      for(i = 0; i < 7; i++)
         TDAT[i]<63:0> = HASHDAT[2*i]<31:0> || HASHDAT[2*i+1]<31:0>
      TDAT[7]<63:0> = GPR[rt]
      TIV = SHA-256(TDAT, TIV)
      for(i = 0; i < 4; i++) {
         HASHIV[2*i]<31:0>   = TIV[i]<63:32>
         HASHIV[2*i+1]<31:0> = TIV[i]<31:0>
      }
      HASHIV[7..0]<63:32> = unpredictable
      HASHDAT[14..0]<63:0> = unpredictable
   endif
else
```

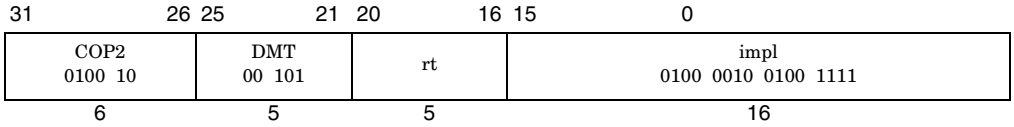
```
    SignalException(CoprocessorUnusable, 2)
  endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

SHA-512 Hash **CVM_MT_HSH_STARTSHA512**



Format: DMTC2 rt, 0x424F **CVM**

Purpose:

To start a SHA-512 hash. The HASHDAT words together with rt are hashed using HASHIV. HASHIV is updated with the result in wide mode.

Description: `HASHIV[7:0] = SHA-512(rt || HASHDAT[14:0], HASHIV[7:0])`

Restrictions:

HASHDAT[14:0]<63:0> must be predictable. This means that HASHDAT[14:0] must be written between execution of this CVM_MT_HSH_STARTSHA512 and the execution of the immediately preceding CVM_MT_HSH_STARTMD5, CVM_MT_HSH_STARTSHA, CVM_MT_HSH_STARTSHA256, or CVM_MT_HSH_STARTSHA512.

All of HASHIV[7..0]<31:0> must be predictable prior to this CVM_MT_HSH_STARTSHA512. This means that HASHIV[7:0] must have been written since the last CVM_MT_HSH_STARTMD5, CVM_MT_HSH_STARTSHA, or CVM_MT_HSH_STARTSHA256.

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

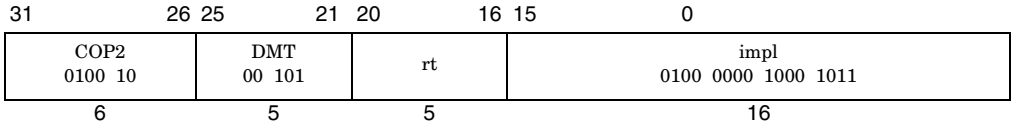
```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or CvmCtl[NOCRYPTO] then
      SignalException(ReservedInstruction)
   else
      HASHIV[7:0] = SHA-512(rt || HASHDAT[14:0], HASHIV[7:0])
      HASHDAT[14:0]<63:0> = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

KASUMI Encrypt — CVM_MT_KAS_ENC



Format: DMTC2 rt, 0x408B **CVM**

Purpose:

To do a KASUMI encrypt. GPR[rt] is the input to a KASUMI encrypt (using 3DESKEY[1:0]). 3DESRESULT is set to the result of the encrypt.

Description: `KASUMIRESULT<63:0> = 3DESRESULT<63:0> = KASUMI(rt, 3DESKEY[1:0])`

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or (not CvmCtl[KASUMI]) then
      SignalException(ReservedInstruction)
   else
      KASUMIRESULT<63:0> = KASUMI(GPR[rt]<63:0>, 3DESKEY[1:0])
      3DESRESULT<63:0> = KASUMIRESULT<63:0>
      3DESKEY[2]<63:0> = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

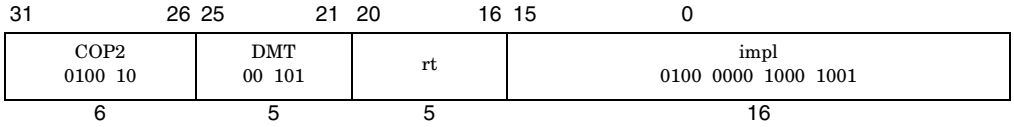
Coprocessor Unusable

Reserved Instruction

Notes:

The CVM_MT_KAS_ENC instruction first appeared on the OCTEON Plus CN5XXX series parts. The behavior of the CVM_MT_KAS_ENC instruction is not predictable on the earlier OCTEON CN3XXX series parts. CvmCtl[KASUMI] is always 0 on CN3XXX parts, so accurately portrays the lack of a KASUMI unit. However, there is no guarantee of a Reserved Instruction exception when CVM_MT_KAS_ENC is executed. KASUMI code that is portable to both CN5XXX (and successors) and CN3XXX must not execute CVM_MT_KAS_ENC when CvmCtl[KASUMI] is clear.

KASUMI CBC Encrypt — CVM_MT_KAS_ENC_CBC



Format: DMTC2 rt, 0x4089 **CVM**

Purpose:

To do a KASUMI CBC encrypt. GPR[rt] XOR KASUMIRESULT is the input to a KASUMI encrypt (using 3DESKEY[1:0]). 3DESRESULT is set to the result of the encrypt.

Description: KASUMIRESULT<63:0> = 3DESRESULT<63:0> = KASUMI(rt ⊕ KASUMIRESULT, 3DESKEY[1:0])

Restrictions:

If access to Coprocessor 2 is not enabled, a Coprocessor Unusable Exception is signaled. If access to 64-bit operations is not enabled, a Reserved Instruction Exception is signaled.

This instruction is not available for OCTEON family parts that do not provide encryption functionality, such as **OCTEON EXP** and **OCTEON CP** family members. Such parts signal a Reserved Instruction exception when issued this instruction.

Operation:

```
if IsCoprocessorEnabled(2) then
   if (not Are64bitOperationsEnabled()) or (not CvmCtl[KASUMI]) then
      SignalException(ReservedInstruction)
   else
      KASUMIRESULT<63:0> = KASUMI(GPR[rt]<63:0> ⊕ KASUMIRESULT<63:0>, 3DESKEY[1:0])
       3DESRESULT<63:0> = KASUMIRESULT<63:0>
       3DESKEY[2]<63:0> = unpredictable
   endif
else
   SignalException(CoprocessorUnusable, 2)
endif
```

Exceptions:

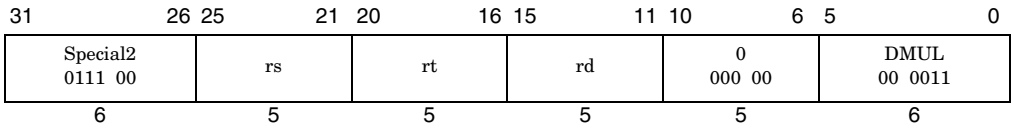
Coprocessor Unusable

Reserved Instruction

Notes:

The CVM_MT_KAS_ENC_CBC instruction first appeared on the OCTEON Plus CN5XXX series parts. The behavior of the CVM_MT_KAS_ENC_CBC instruction is not predictable on the earlier OCTEON CN3XXX series parts. CvmCtl[KASUMI] is always 0 on CN3XXX parts, so accurately portrays the lack of a KASUMI unit. However, there is no guarantee of a Reserved Instruction exception when CVM_MT_KAS_ENC_CBC is executed. KASUMI code that is portable to both CN5XXX (and successors) and CN3XXX must not execute CVM_MT_KAS_ENC_CBC when CvmCtl[KASUMI] is clear.

Multiply Doubleword to GPR — DMUL



Format: DMUL rd, rs, rt **CVM**

Purpose:

To multiply 64-bit signed integers and write the result to a GPR.

Description: `rd = rs × rt`

The 64-bit double-word value in GPR rt is multiplied by the 64-bit value in GPR rs, treating both operands as signed values, to produce a 128-bit result. The low-order 64-bit double word of the result is written to GPR rd. The contents of HI, LO, P0, P1, and P2 are unpredictable after the operation.

No arithmetic exception occurs under any circumstances.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

Operation:

```
if Are64bitOperationsEnabled() then
   prod = GPR[rs] * GPR[rt]
   GPR[rd] = prod<63:0>
else
   SignalException(ReservedInstruction)
endif
```

Exceptions:

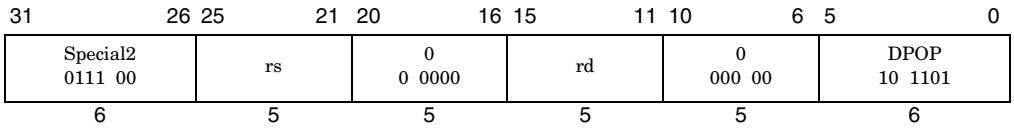
Reserved Instruction

Programming Notes:

The integer multiply operation proceeds asynchronously and other CPU instructions can execute before it is complete. An attempt to read GPR rd before the results are written interlocks until the results are ready. Asynchronous execution does not affect the program result, but offers an opportunity for performance improvement by scheduling the multiply so that other instructions can execute in parallel.

Programs that require overflow detection must check for it explicitly.

Count Ones in a Doubleword — DPOP



Format: DPOP rd, rs **CVM**

Purpose

Count the number of ones in a double word

Description: `rd = count_ones(rs)`

Bits 63..0 of GPR rs are scanned. The number of ones is counted and the result is written to GPR rd.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

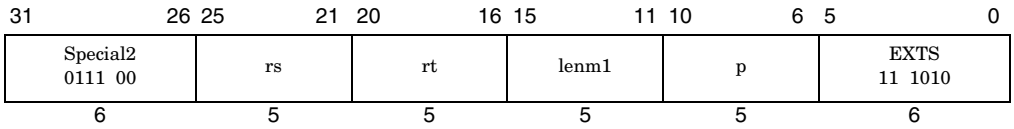
Operation:

```
if Are64bitOperationsEnabled() then
   temp = 0
   for i in 63 .. 0
      if GPR[rs]<i> = 1 then
         temp++
      endif
   endfor
   GPR[rd] = temp
else
   SignalException(ReservedInstruction)
endif
```

Exceptions:

Reserved Instruction.

Extract a Signed Bit Field — EXTS



Format: EXTS rt, rs, p, lenm1 **CVM**

Purpose:

To extract and sign-extend a bit field that starts from the lower 32 bits of a register.

Description: `rt = sign-extend(rs<p+lenm1:p>, lenm1)`

Bit locations p + lenm1 to p are extracted from rs and the result is written into the lowest bits of destination register rt. The remaining bits in rt are a sign-extension of the most-significant bit of the bit field (i.e. rt<63:lenm1> are all duplicates of the source-register bit rs<p+lenm1>).

Restrictions:

The p and lenm1 fields are only 5-bits, so the widest allowed bit field is 32-bits.

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

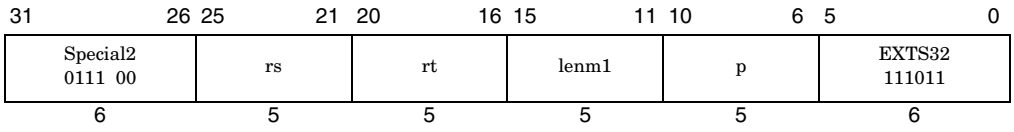
Operation:

```
if Are64bitOperationsEnabled() then
   GPR[rt] <- sign-extend(GPR[rs]<p+lenm1:p>, lenm1)
else
   SignalException(ReservedInstruction)
endif
```

Exceptions:

Reserved Instruction.

Extract a Signed Bit Field Plus 32 EXTS32



Format: EXTS32 rt, rs, p, lenm1 **CVM**

Purpose:

To extract and sign-extend a bit field that starts from the upper 32 bits of a register.

Description: `rt = sign-extend(rs<p+32+lenm1:p+32>, lenm1)`

Bit locations p + 32 + lenm1 to p + 32 are extracted from rs and the result is written into the lowest bits of destination register rt. The remaining bits in rt are a sign-extension of the most-significant bit of the bit field (i.e. rt<63:lenm1> are all duplicates of the source-register bit rs<p+32+lenm1>).

Restrictions:

The p and lenm1 fields are only 5-bits, so the widest allowed bit field is 32-bits.

The result register GPR[rt] is unpredictable when p+32+lenm1 is larger than 63.

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

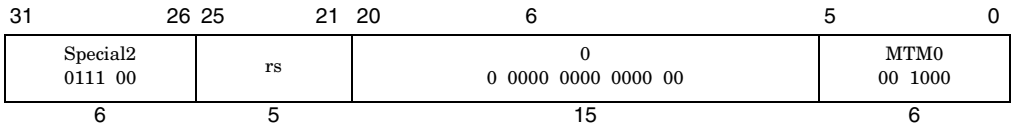
Operation:

```
if Are64bitOperationsEnabled() then
   GPR[rt] <- sign-extend(GPR[rs]<p+32+lenm1:p+32>, lenm1)
else
   SignalException(ReservedInstruction)
endif
```

Exceptions:

Reserved Instruction.

Load Multiplier Register MPL0 — MTM0



Format: MTM0 rs **CVM**

Purpose:

To load the multiplier register MPL0 (and zero P0, P1, P2).

Description: `MPL0 = rs; P0, P1, P2 = 0`

The 64-bit double-word value in GPR rs is loaded into the multiplier register MPL0 and P0-P2 are zeroed.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

Operation:

```
if Are64bitOperationsEnabled() and !CvmCtl[NOMUL] then
   MPL0<63:0> = GPR[rs]
   P0 = 0
   P1 = 0
   P2 = 0
else
   SignalException(ReservedInstruction)
endif
```

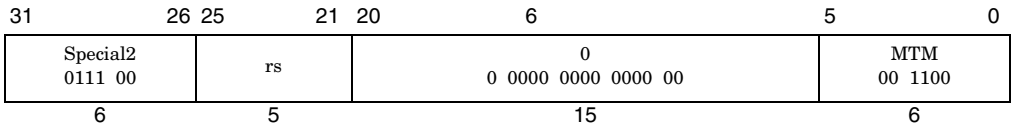
Exceptions:

Reserved Instruction

Programming Notes:

See “Special MUL Topics” on page 194 for a method to read out the MPL0 value.

Load Multiplier Register MPL1 — MTM1



Format: MTM1 rs **CVM**

Purpose:

To load the multiplier register MPL1 (and zero P0, P1, P2).

Description: `MPL1 = rs; P0, P1, P2 = 0`

The 64-bit doubleword value in GPR rs is loaded into the multiplier register MPL1 and P0-P2 are zeroed.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

Operation:

```
if Are64bitOperationsEnabled() and !CvmCtl[NOMUL] then
   MPL1<63:0> = GPR[rs]
   P0 = 0
   P1 = 0
   P2 = 0
else
   SignalException(ReservedInstruction)
endif
```

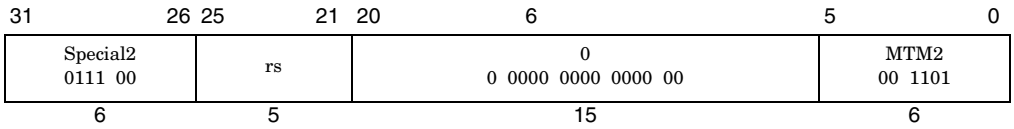
Exceptions:

Reserved Instruction

Programming Notes:

See “Special MUL Topics” on page 194 for a method to read out the MPL1 value.

Load Multiplier Register MPL2 — MTM2



Format: MTM2 rs **CVM**

Purpose:

To load the multiplier register MPL2 (and zero P0, P1, P2).

Description: `MPL2 = rs; P0, P1, P2 = 0`

The 64-bit doubleword value in GPR rs is loaded into the multiplier register MPL2 and P0-P2 are zeroed.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

Operation:

```
if Are64bitOperationsEnabled() and !CvmCtl[NOMUL] then
   MPL2<63:0> = GPR[rs]
   P0 = 0
   P1 = 0
   P2 = 0
else
   SignalException(ReservedInstruction)
endif
```

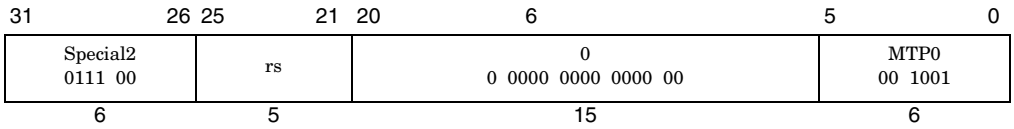
Exceptions:

Reserved Instruction

Programming Notes:

See “Special MUL Topics” on page 194 for a method to read out the MPL2 value.

Load Multiplier Register P0 — MTP0



Format: MTP0 rs **CVM**

Purpose:

To load the product register P0.

Description: `P0 = rs`

The 64-bit doubleword value in GPR rs is loaded into the product register P0.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

Operation:

```
if Are64bitOperationsEnabled() and !CvmCtl[NOMUL] then
   P0 = GPR[rs]
else
   SignalException(ReservedInstruction)
endif
```

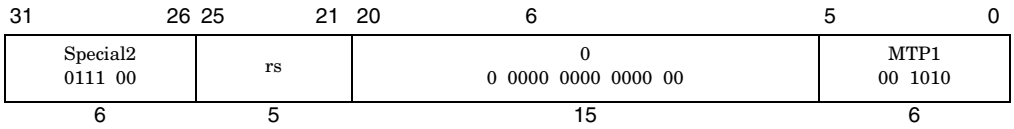
Exceptions:

Reserved Instruction

Programming Notes:

See “Special MUL Topics” on page 194 for a method to read out the P0 value.

Load Multiplier Register P1 — MTP1



Format: MTP1 rs **CVM**

Purpose:

To load the product register P1.

Description: `P1 = rs`

The 64-bit doubleword value in GPR rs is loaded into the product register P1.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

Operation:

```
if Are64bitOperationsEnabled() and !CvmCtl[NOMUL] then
   P1 = GPR[rs]
else
   SignalException(ReservedInstruction)
endif
```

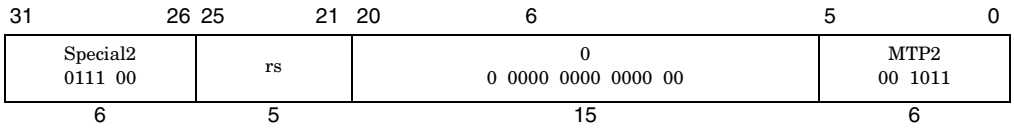
Exceptions:

Reserved Instruction

Programming Notes:

See “Special MUL Topics” on page 194 for a method to read out the P1 value.

Load Multiplier Register P2 — MTP2



Format: MTP2 rs **CVM**

Purpose:

To load the product register P2.

Description: `P2 = rs`

The 64-bit doubleword value in GPR rs is loaded into the product register P2.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

Operation:

```
if Are64bitOperationsEnabled() and !CvmCtl[NOMUL] then
   P2 = GPR[rs]
else
   SignalException(ReservedInstruction)
endif
```

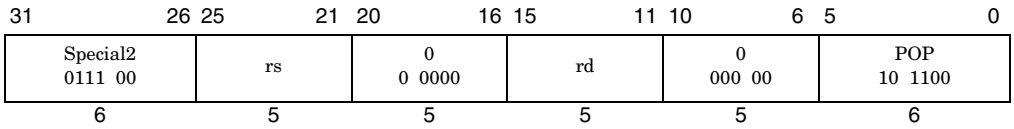
Exceptions:

Reserved Instruction

Programming Notes:

See “Special MUL Topics” on page 194 for a method to read out the P2 value.

Count Ones in a Word — POP



Format: POP rd, rs **CVM**

Purpose

Count the number of ones in a word

Description: `rd = count_ones(rs)`

Bits 31..0 of GPR rs are scanned. The number of ones is counted and the result is written to GPR rd.

Restrictions:

None.

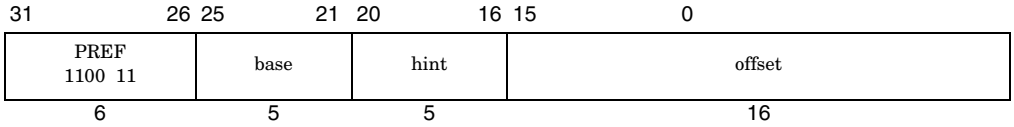
Operation:

```
temp = 0
for i in 31 .. 0
   if GPR[rs]<i> = 1 then
      temp++
   endif
endfor
GPR[rd] = temp
```

Exceptions:

None.

Prefetch PREF



Format: PREF hint, offset(base) **MIPS32/CVM**

Purpose:

To move data between memory and cache.

Description: `prefetch_memory(base+offset)`

PREF adds the 16-bit signed offset to the contents of GPR base to form an effective byte address. PREF operate on the surrounding naturally-aligned cache block, typically prefetching it into a cache. The cache block size is 128 bytes on CN50XX. With the exception of the Prepare-For-Store and Don't-Write-Back hints, PREF instruction execution does not change architectural state. The hint field selects the particular operation.

PREF does not cause addressing-related exceptions. If the address specified would cause an addressing exception, the exception condition is ignored and no data movement occurs.

PREF never generates a memory operation for I/O memory locations, DSEG locations, and CVMSEG locations.

Possible values of the hint field for the PREF Instruction are listed in the table below.

NOTE: This is a standard MIPS32 instruction for which OCTEON chips implement all the functionality defined in the MIPS standard instruction set, as well as additional, cnMIPS-specific operations. The hint field is cnMIPS-specific.

Hint Value	Name	Data Use and Desired Prefetch Action
0-1, 6-7, 25, 31	Normal	The block will be prefetched into the L1 cache and the L2 cache.
2-3, 8-24, 26-27	Reserved	Results are unpredictable.
4, 5	L1 only	The block will be prefetched into the L1 cache but will not be put into the L2 cache.
28	L2 only	The block will be prefetched into the L2 cache without putting it into the L1 cache.
29	Don't-Write-Back	The block will not be written back to memory (i.e. the cache dirty bit is cleared). The value of the bytes in the cache block are unpredictable and may change value unpredictably until they are later stored to. The Don't-Write-Back operation can be used to avoid unnecessary write backs from the L2 cache (to DRAM) for memory locations that are not being used.
30	Prepare-For-Store	A write buffer entry is created. If the write buffer entry misses in the L2 cache, OCTEON will not read the prior block from DRAM into the L2 cache. The value of the bytes in the cache block are unpredictable and may change value unpredictably until they are later stored to. The Prepare-For-Store operations can be used to avoid unnecessary DRAM reads for memory locations whose current value does not matter.

Restrictions:

None.

Operation:

```
vAddr , GPR[base] + sign_extend(offset)
(pAddr, CCA) , AddressTranslation(vAddr, DATA, LOAD)
Prefetch(CCA, pAddr, vAddr, DATA, hint)
```

Exceptions:

Cache Error, Watch, Breakpoint

Prefetch does not take any TLB-related or address-related exceptions under any circumstances.

OCTEON Core watchpoints and (EJTAG) breakpoints match against the PREF 29 and PREF 30 instructions as if they were stores to every byte in the surrounding cache block. These PREFs always match a value compare.

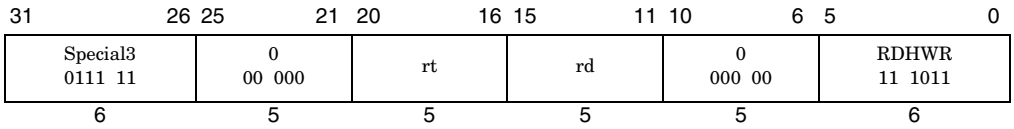
None of the PREF 0-28, 31 instruction ever match against watchpoints/breakpoints.

Programming Notes:

Prefetch cannot prefetch data from a mapped location unless the translation for that location is present in the TLB. Locations in memory pages that have not been accessed recently may not have translations in the TLB, so prefetch may not be effective for such locations.

Prefetch does not cause addressing exceptions, so a legal PREF instruction can have any effective address.

Read Hardware Register — RDHWR



Format: RDHWR rt, rd **MIPS32 Release 2/CVM**

Purpose:

To move the contents of a hardware register to a general purpose register (GPR) if that operation is enabled by privileged software.

Description: `rt= HWR[rd]`

If access is allowed to the specified hardware register, the contents of the register specified by rd is sign-extended and loaded into general register rt. Access control for each register is selected by the bits in the coprocessor 0 HWREna register.

The available hardware registers, and the encoding of the rd field for each, is as shown in the table below.

NOTE: This is a standard MIPS32 instruction for which OCTEON chips implement all the functionality defined in the MIPS standard instruction set, as well as additional, cnMIPS-specific operations. The register numbers 30 and 31 are cnMIPS-specific. <31:30> are present in the HWREna register on OCTEON chips.

Register Number (rd Value)	Register Name	Contents
0	CPUNum	Number of the Core on which the program is currently running. This comes directly from the coprocessor 0 EBaseCPUNum field.
1	SYNCI_Step	Address step size to be used with the SYNCI instruction. See that instruction's description for the use of this value. A SYNCI flushes the entire instruction cache on OCTEON, so OCTEON's SYNCI_Step is 4096.
2	CC	High-resolution cycle counter. This comes directly from the coprocessor 0 Count register. This is one for OCTEON.
3	CCRes	Resolution of the CC register. This value denotes the number of cycles between update of the register.
30	ChOrd	The current value of the pending tag switch bit local to this core: 1 = the local bit indicates that there is no pending POW tag switch 0 = the local bit indicates that there is a pending POW tag switch See “Core and Fetch-and-Add Pending Switch Bits” on page 241 for a discussion of the pending tag switch bit that is local to each core. Note that the local pending switch bit is NOT the actual POW state, but should match in normal circumstances.
31	CvmCount	The current value of the 64-bit Cavium-specific coprocessor 0 CvmCount register. Note that the full 64-bit CvmCount value is provided when 64-bit operations are enabled, and only the lower 32 bits of CvmCount are provided (sign-extended) when 64-bit operations are not enabled.
All others		Access results in a Reserved Instruction Exception

Restrictions:

In implementations of Release 1 of the Architecture, this instruction resulted in a Reserved Instruction Exception.

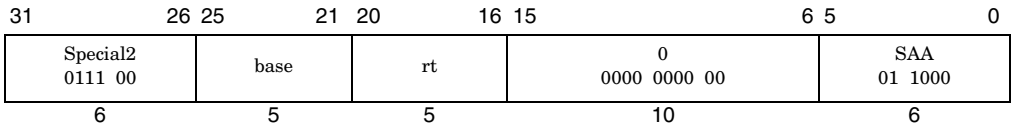
Operation:

```
if ((HWREna(rd) = 1) or IsCoprocessorEnabled(0)) then
     case rd
        16#00: temp = EBase[CPUNum]
        16#01: temp = 4096
        16#02: temp = sign_extend(Count)
        16#03: temp = 1
        16#1e: temp = local pending switch bit
        16#1f: if Are64bitOperationsEnabled() then
                  temp = CvmCount<63:0>
               else
                  temp = sign_extend(CvmCount<31:0>)
        otherwise: SignalException(ReservedInstruction)
     endcase
     GPR[rt] = temp
  else
     SignalException(ReservedInstruction)
  endif
```

Exceptions:

Reserved Instruction

Store Atomic Add Word — SAA



Format: SAA rt, (base) **CVM**

Purpose:

To atomically add a word to a memory location.

Description: `memory[base] = memory[base] + rt`

The least-significant 32-bit word of register rt is added to the memory at the aligned address specified by register base.

The memory read, add, and memory store are not interrupted by any other instructions on this or any other processor. No Integer Overflow exception occurs under any circumstance.

Restrictions:

The address in the base register must be naturally-aligned. If either of the two least-significant bits of the address are set, an Address Error Exception occurs. SAA only works on cacheable memory locations. When the resultant physical address is a non-cacheable / IO address, the behavior is generally unpredictable.

Operation:

```
vAddr = GPR[base]
if (vAddr<1:0> != 0) or
    (vAddr is CVMSEG IO or CVMSEG LM and CVMSEG is enabled) then
  SignalException(AddressError)
else
  pAddr = AddressTranslation(vAddr, DATA, STORE)
  if (pAddr is IO) or (vAddr is DSEG and DSEG is enabled) then
   unpredictable
  else
   memdoubleword = LoadMemory (CCA, WORD, pAddr, vAddr, DATA)
   byte = vAddr<2:0> XOR (BigEndianCPU || 0<1:0>)
   resultdoubleword = memdoubleword + (GPR[rt]<63:0> << (byte * 8))
   StoreMemory (CCA, WORD, resultdoubleword, pAddr, vAddr, DATA)
  endif
endif
```

Exceptions:

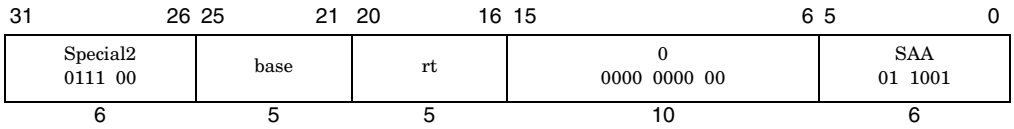
TLB Refill, TLB Invalid, TLB Modified, Bus Error, Address Error, Watch, Breakpoint

Programming Notes:

The SAA instruction is performance-optimized for memory locations that are present in L2/DRAM, not the level-one data cache. The cache block that contains the memory location is removed from the level-one data cache (if present) during execution of the SAA instruction on the CN5XXX series.

TLB accesses, watchess, and breakpoints treat SAA instructions as an SW instruction. The value comparison of value breakpoints compares against the contents of the least-significant word of GPR[rt], not the old contents of the memory location, nor the contents of the memory location after the SAA.

Store Atomic Add Double Word — SAAD



Format: SAAD rt, (base) **CVM**

Purpose:

To atomically add a double word to a memory location.

Description: `memory[base] = memory[base] + rt`

The 64-bit register rt is added to the memory at the aligned address specified by register base.

The memory read, add, and memory store are not interrupted by any other instructions on this or any other processor. No Integer Overflow exception occurs under any circumstance.

Restrictions:

The address in the base register must be naturally-aligned. If either of the three least-significant bits of the address are set, an Address Error Exception occurs. SAAD only works on cacheable memory locations. When the resultant physical address is a noncacheable / IO address, the behavior is generally unpredictable.

A Reserved Instruction Exception is signalled if access to 64-bit operations is not enabled.

Operation:

```
if Are64bitOperationsEnabled() then
  vAddr = GPR[base]
  if (vAddr<2:0> != 0) or
     (vAddr is CVMSEG IO or CVMSEG LM and CVMSEG is enabled) then
    SignalException(AddressError)
  else
    pAddr = AddressTranslation(vAddr, DATA, STORE)
    if (pAddr is IO) or (vAddr is DSEG and DSEG is enabled) then
     unpredictable
    else
     memdoubleword = LoadMemory (CCA, DOUBLEWORD, pAddr, vAddr, DATA)
     StoreMemory (CCA, DOUBLEWORD, memdoubleword + GPR[rt]<63:0>, pAddr, vAddr, DATA)
    endif
  endif
else
  SignalException(ReservedInstruction)
endif
```

Exceptions:

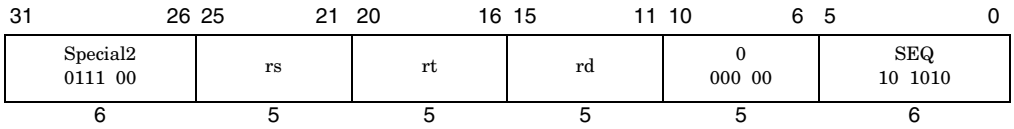
Reserved Instruction, TLB Refill, TLB Invalid, TLB Modified, Bus Error, Address Error, Watch, Breakpoint

Programming Notes:

The SAAD instruction is performance-optimized for memory locations that are present in L2/DRAM, not the level-one data cache. The cache block that contains the memory location is removed from the level-one data cache (if present) during execution of the SAAD instruction on the 5XXX series.

TLB accesses, watches, and breakpoints treat SAAD instructions as an SD instruction. The value comparison of value breakpoints compares against the GPR[rt] contents, not the old contents of the memory location, nor the contents of the memory location after the SAAD.

Set on Equal — SEQ



Format: SEQ rd, rs, rt **CVM**

Purpose:

To record the result of an equals comparison

Description: `rd = (rs == rt)`

Compare the contents of GPR rs and GPR rt and record the Boolean result of the comparison in GPR rd. If GPR rs equals GPR rt, the result is 1 (true); otherwise, it is 0 (false).

The arithmetic comparison does not cause an Integer Overflow exception.

Restrictions:

None.

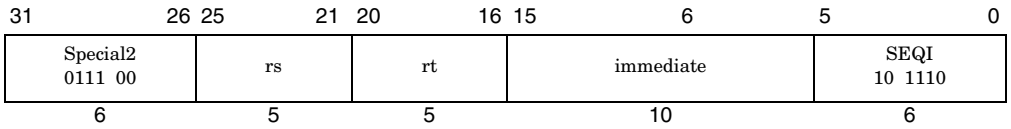
Operation:

```
if GPR[rs] == GPR[rt] then
   GPR[rd] = 0^63 || 1
else
   GPR[rd] = 0^64
endif
```

Exceptions:

None.

Set on Equal Immediate — SEQI



Format: SEQI rt, rs, immediate **CVM**

The immediate is a 10-bit sign-extended value contained in <15:6> of the instruction.

Purpose: To record the result of an equals comparison with a constant

Description: `rt = (rs == immediate)`

Compare the contents of GPR rs and the 10-bit signed immediate and record the Boolean result of the comparison in GPR rt. If GPR rs equals immediate, the result is 1 (true); otherwise, it is 0 (false).

The arithmetic comparison does not cause an Integer Overflow exception.

Restrictions:

None.

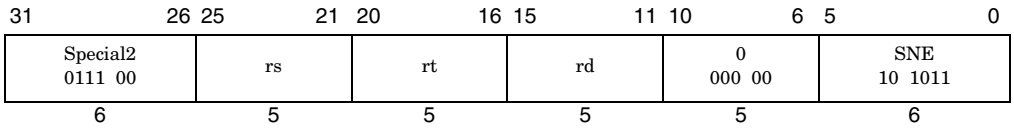
Operation:

```
if GPR[rs] == sign_extend(immediate) then
   GPR[rt] = 0^63 || 1
else
   GPR[rt] = 0^64
endif
```

Exceptions:

None.

Set on Not Equal — SNE



Format: SNE rd, rs, rt **CVM**

Purpose:

To record the result of a not equals comparison

Description: `rd = (rs != rt)`

Compare the contents of GPR rs and GPR rt and record the Boolean result of the comparison in GPR rd. If GPR rs equals GPR rt, the result is 0 (false); otherwise, it is 1 (true).

The arithmetic comparison does not cause an Integer Overflow exception.

Restrictions:

None.

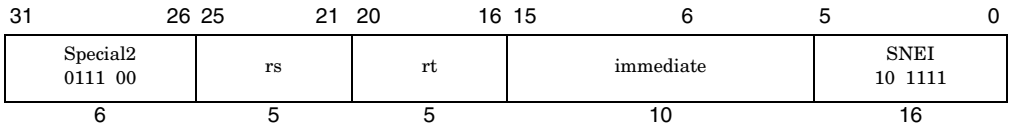
Operation:

```
if GPR[rs] != GPR[rt] then
   GPR[rd] = 0^63 || 1
else
   GPR[rd] = 0^64
endif
```

Exceptions:

None.

Set on Not Equal Immediate SNEI



Format: SNEI rt, rs, immediate **CVM**

The immediate is a 10-bit sign-extended value contained in <15:6> of the instruction.

Purpose:

To record the result of a not equals comparison with a constant

Description: `rt = (rs != immediate)`

Compare the contents of GPR rs and the 10-bit signed immediate and record the Boolean result of the comparison in GPR rt. If GPR rs equals immediate, the result is 0 (false); otherwise, it is 1 (true).

The arithmetic comparison does not cause an Integer Overflow exception.

Restrictions:

None.

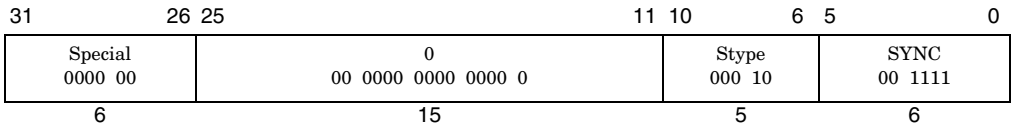
Operation:

```
if GPR[rs] != sign_extend(immediate) then
   GPR[rt] = 0^63 || 1
else
   GPR[rt] = 0^64
endif
```

Exceptions:

None.

Synchronize IOBDMAs — SYNCIOBDMA



Format: SYNCIOBDMA **CVM**

Purpose:

To complete IOBDMA operations

Description:

SYNCIOBDMA execution guarantees that there are no in-flight IOBDMA operations; a subsequent CVMSEG LM load will get results updated for all outstanding IOBDMAs.

Restrictions:

None.

Operation:

```
SyncOperation(stype)
```

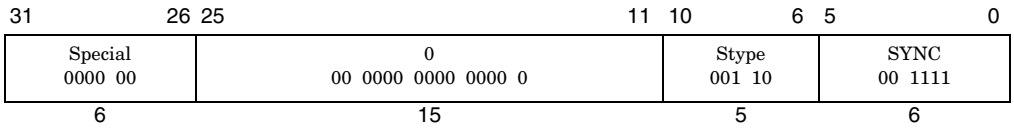
Exceptions:

None.

Programming Notes:

IOBDMA operations (see Section 4.7 on page 160) can be initiated without waiting for their CVMSEG LM result. SYNCIOBDMA operations are necessary to complete them. (SYNC/SYNCS instructions also guarantee that in-flight IOBDMA operations complete, but are more costly than SYNCIOBDMA operations.)

Synchronize Special — SYNCS



Format: SYNCS **CVM**

Purpose:

To order unmarked L2/DRAM and I/O load/store operations and IOBDMAs. SYNCS is identical to SYNC, except that SYNCS does not order marked L2/DRAM load/store operations.

Description:

- SYNCS affects the ordering of all unmarked L2/DRAM load/store operations, I/O load/store operations, and IOBDMA operations. The unmarked L2/DRAM load/store operations, I/O load/store operations, and IOBDMA operations that occur before the SYNCS are completed before any unmarked L2/DRAM load/store operations, I/O load/store operations, or IOBDMA operations after the SYNCS are allowed to start.
- SYNCS does not affect the order of marked L2/DRAM load/store operations. Marked L2/DRAM load/store operations are those L2/DRAM load/store operations whose cache coherency attribute equals 7.
- Loads are complete when the destination register is written.
- L2/DRAM stores are complete when the stored value is visible to every other core and all OCTEON IO units.
- I/O stores are posted on OCTEON, and are “complete” when they reach the coherent memory bus.
- SYNCS execution guarantees that there are no in-flight IOBDMA operations; a subsequent CVMSEG LM load gets results updated for all outstanding IOBDMA operations.
- SYNCS does not guarantee the order in which instruction fetches are performed.
- SYNCS is identical to SYNC when CvmMemCtl[DISSYNCWS] is set.
- Refer to the cnMIPS™ Cores chapter (“cnMIPS™ Cores” on page 143) for more discussion of ordering.

Restrictions:

None.

Operation:

```
SyncOperation(stype)
```

Exceptions:

None.

Programming Notes:

On OCTEON, SYNCW and SYNCWS are much faster than SYNC or SYNCS, so should be used when possible.

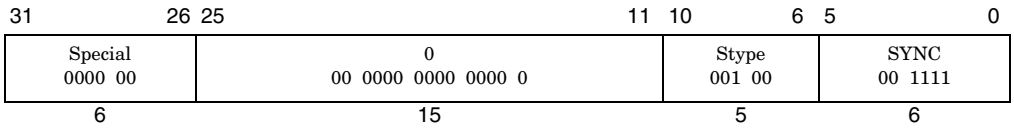
A processor executing load and store instructions observes the order in which loads and stores occur in the instruction stream; this is known as program order.

A parallel program has multiple instruction streams that can execute simultaneously on different processors. In OCTEON, the order in which the effects of loads and stores are observed by other processors - the global order of the loads and stores - determines the actions necessary to reliably share data in parallel programs.

In OCTEON, the effects of store instructions executed by one processor may be observed out of program order by other processors, so parallel programs must take explicit actions to reliably share data. At critical points in the program, the effects of stores from an instruction stream must occur in the same order for all processors. SYNCS separates the unmarked L2/DRAM and I/O load/store operations executed on the processor into two groups, and the effect of all references in one group is seen by all processors before the effect of any reference in the subsequent group.

IOBDMA operations (see Section 4.7 on page 160) can be initiated without waiting for their CVMSEG LM result. SYNCS operations complete them. (SYNCIOBDMA instructions also guarantee that in-flight IOBDMA operations complete, and are less costly than SYNC/SYNCS operations.)

Synchronize Stores — SYNCW



Format: SYNCW **CVM**

Purpose:

To order stores. SYNCW is similar to SYNCWS, but SYNCW orders more stores.

Description:

- SYNCW affects only stores. All the stores that occur before the SYNCW are completed before any of the stores after the SYNC are allowed to start.
- L2/DRAM stores are complete when the stored value is visible to every other core and all OCTEON units.
- I/O stores are posted on OCTEON, and are “complete” when they reach the coherent memory bus.
- SYNCW does not affect any ordering between load/IOBDMA operations and stores. Furthermore, SYNCW operations are queued and complete out of order with respect to load/IOBDMA’s.
- Refer to the cnMIPS™ Cores chapter (“cnMIPS™ Cores” on page 143) for more discussion of ordering.

Restrictions:

None.

Operation:

```
SyncOperation(stype)
```

Exceptions:

None.

Programming Notes:

On OCTEON, SYNCW and SYNCWS are much faster than SYNC or SYNCS, so should be used whenever possible. SYNCWS can produce substantially better system performance than SYNCW, as it orders fewer stores.

A processor executing load and store instructions observes the order in which loads and stores occur in the instruction stream; this is known as program order.

A parallel program has multiple instruction streams that can execute simultaneously on different processors. In OCTEON, the order in which the effects of loads and stores are observed by other processors - the global order of the loads and stores - determines the actions necessary to reliably share data in parallel programs.

In OCTEON, the effects of store instructions executed by one processor may be observed out of program order by other processors, so parallel programs must take explicit actions to reliably share data. At critical points in the program, the effects of stores from an instruction stream must occur in the same order for all processors. SYNCW separates the load operations and store operations executed on the processor into two groups, and the effect of all stores in one group is seen by all processors before the effect of any store operation in the subsequent group. In effect, SYNCW causes OCTEON to be strongly ordered for the Core at the instant that the SYNCW is executed.

Conditions at entry: The value 0 has been stored in FLAG and that value is observable by core B

Core A (writer)

```
SW R1, DATA   #              change shared DATA value
LI R2, 1
SYNCW         # (or SYNCWS) Perform DATA store before performing FLAG store
SW R2, FLAG   #              say that the shared DATA value is valid
SYNCW         # (or SYNCWS) Force the FLAG store soon ( OCTEON-specific)
```

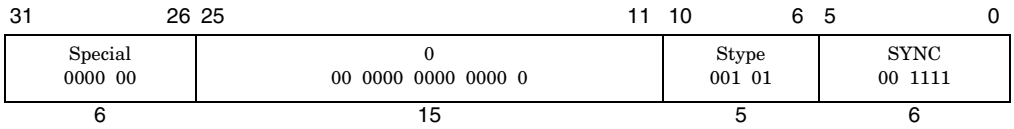
Core B (reader)

```
   LI R2, 1
1: LW R1, FLAG     # Get FLAG
   BNE R2, R1, 1B  # if it says that DATA is not valid, poll again
   NOP
   LW R1, DATA     # Read (valid) shared DATA value
```

The code fragments above show how SYNCW can be used to coordinate the use of shared data between separate writer and reader instruction streams in OCTEON. The FLAG location is used by the instruction streams to determine whether the shared data item DATA is valid. The first SYNCW instruction executed by core A forces the store of DATA to be performed globally before the store to FLAG is performed. This is necessary for correct behavior on OCTEON.

The second SYNCW instruction executed by core A is not necessary for correctness, but has very important performance effects on OCTEON. Without it, the store to FLAG may linger in core A's write buffer before it becomes visible to other cores. (If core A is not performing many stores, this may add hundreds of thousands of cycles to the flag release time since the OCTEON core nominally retains stores to attempt to merge them before sending the store on the CMB.) Applications should include this second SYNCW instruction after flag or lock releases.

Synchronize Stores Special — SYNCWS



Format: SYNCWS **CVM**

Purpose:

To order unmarked L2/DRAM and I/O stores. SYNCWS is identical to SYNCW, except that SYNCWS does not order marked L2/DRAM stores.

Description:

- SYNCWS affects the ordering of unmarked L2/DRAM store operations and all I/O store operations. The unmarked L2/DRAM and I/O store operations that occur before the SYNCWS are completed before the unmarked L2/DRAM and I/O store operations after the SYNC are allowed to start.
- SYNCWS does not affect the order of marked L2/DRAM store operations. Marked L2/DRAM store operations are those L2/DRAM store operations whose cache coherency attribute equals 7.
- L2/DRAM store operations are completed when the stored value is visible to every other core and all OCTEON I/O units.
- I/O store operations are posted on OCTEON, and are “complete” when they reach the coherent memory bus.
- SYNCWS does not affect any ordering between load operations and store operations, nor between IOBDMA operations and store operations. Furthermore, SYNCWS operations may execute out of order with respect to load operations and IOBDMA operations.
- SYNCWS is identical to SYNCW when CvmMemCtl[DISSYNCWS] is set.
- Refer to the cnMIPS™ Cores chapter (“cnMIPS™ Cores” on page 143) for more discussion of ordering.

Restrictions:

None.

Operation:

```
SyncOperation(stype)
```

Exceptions:

None.

Programming Notes:

On OCTEON, SYNCW and SYNCWS are much faster than SYNC or SYNCS, so should be used whenever possible. SYNCWS can produce substantially better system performance than SYNCW, as it orders fewer stores.

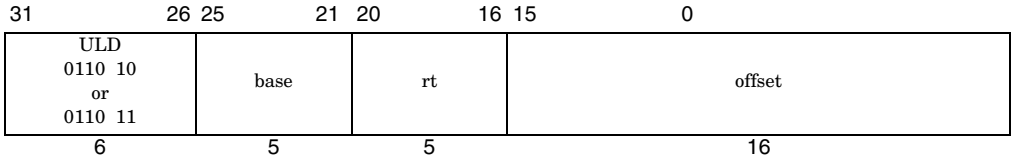
A processor executing load and store instructions observes the order in which loads and stores occur in the instruction stream; this is known as program order.

A parallel program has multiple instruction streams that can execute simultaneously on different processors. In OCTEON, the order in which the effects of loads and stores are observed by other processors - the global order of the loads and stores - determines the actions necessary to reliably share data in parallel programs.

In OCTEON, the effects of store instructions executed by one processor may be observed out of program order by other processors, so parallel programs must take explicit actions to reliably share data. At critical points in the program, the effects of stores from an instruction stream must occur in the same order for all processors. SYNCWS separates the unmarked memory and I/O stores executed on the processor into two groups, and the effect of all store operations in one group is seen by all processors before the effect of any store in the subsequent group.

Refer to the SYNCW Programming Notes: section on page 942 for a code fragment of a writer/reader application and optimization on OCTEON.

Unaligned Load Doubleword ULD



Format: ULD rt, offset(base) **CVM**

The ULD instruction does not exist when CvmCtl[USEUN] is clear. (The MIPS LDL and LDR instructions do not exist when CvmCtl[USEUN] is set.)

The ULD instruction (when enabled by CvmCtl[USEUN]) consumes either the MIPS LDL or LDR major opcodes according to the following table:

CvmCtl[USELONLY]	BigEndianCPU (!CvmCtl[LE])	MIPS LDL opcode is:	MIPS LDR opcode is:
0	0	NOP	ULD
0	1	ULD	NOP
1	X	ULD	NOP

When CvmCtl[USEUN] is set, one of MIPS LDL and LDR major opcodes execute as NOPs, as specified in the table above.

Purpose:

To load a doubleword from a (potentially-unaligned) memory location

Description: `rt = memory[base+offset]`

The contents of the 64-bit doubleword at the memory location specified by the effective address are fetched and placed in GPR rt. The 16-bit signed offset is added to the contents of GPR base to form the effective address.

Restrictions:

Whenever any of the bytes required to service the unaligned load reside in either I/O space or DSEG or CVMSEG IO (note NOT normal/cacheable memory space and CVMSEG LM), the effective address must be naturally-aligned. If any of the three least-significant bits of the address is non-zero in this case, an Address Error exception occurs. Note that these alignment address error exceptions are lower-priority than TLB refill and invalid exceptions, though.

CVMSEG I/O references cause address errors.

CVMSEG LM references by these loads cause an address error when outside the legal range allowed by CvmMemCtl[LMEMSZ] (assuming CVMSEG LM is enabled by CvmMemCtl[CVMSEGENA*]).

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

Operation:

```
if (Are64bitOperationsEnabled() then
     vAddr = sign_extend(offset) + GPR[base]
     if (vAddr<2:0> != 0) then
```

```
      if (vAddr is CVMSEG IO and CVMSEG is enabled)
           or (vAddr is CVMSEG LM and CVMSEG is enabled and address is out of range)then
         SignalException(AddressError)
      else
         pAddr = AddressTranslation(vAddr, DATA, LOAD)
         if (pAddr is IO) or (pAddr is DSEG)
             or (vAddr+8 is CVMSEG IO and CVMSEG is enabled)
             or (vAddr+8 is CVMSEG LM and CVMSEG is enabled and address is out of range) then
            SignalException(AddressError)
         else
            pAddr2 = AddressTranslation(vAddr+8, DATA, LOAD)
            if (paddr2 is IO) or (paddr2 is DSEG) then
               SignalException(AddressError)
            else
               memdoubleword0 = LoadMemory (CCA, DOUBLEWORD, pAddr, vAddr, DATA)
               memdoubleword1 = LoadMemory (CCA, DOUBLEWORD, pAddr2, vAddr+8, DATA)
               if (BigEndianCPU) then
                  memdoubleword = (memdoubleword0 << (vAddr<2:0> * 8)) |
                                  (memdoubleword1 >> ((8-vAddr<2:0>) * 8));
               else
                  memdoubleword = (memdoubleword0 >> (vAddr<2:0> * 8)) |
                                  (memdoubleword1 << ((8-vAddr<2:0>) * 8));
               endif
               GPR[rt] = memdoubleword
            endif
         endif
      endif
   else
      pAddr = AddressTranslation(vAddr, DATA, LOAD)
      memdoubleword = LoadMemory (CCA, DOUBLEWORD, pAddr, vAddr, DATA)
      GPR[rt] = memdoubleword
   endif
else
   SignalException(ReservedInstruction)
endif
```

Exceptions:

TLB Refill, TLB Invalid, Bus Error, Address Error, Reserved Instruction, Watch, Breakpoint

Notes:

OCTEON Cores execute naturally-aligned LDs one cycle faster than naturally-aligned ULDs, so the LD instructions should be used rather than ULD when an address is known to be naturally-aligned.

ULD is an assembler macro that converts to MIPS LDL/LDR sequences on most MIPS assemblers.

The following table indicates the byte lanes for 64-bit references in big-endian mode:

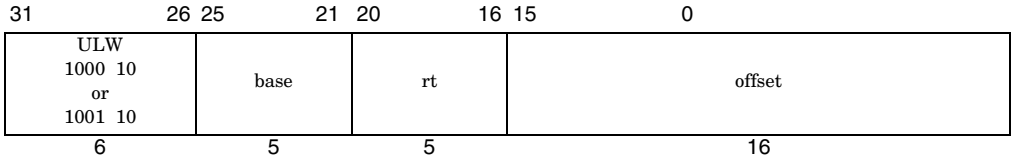
VA<2:0>	Register File Byte Positions in the two memory double words							
0	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>	<15: 8>	< 7: 0>
	X	X	X	X	X	X	X	X
1	X	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>	<15: 8>
	< 7: 0>		X	X	X	X	X	X
2	X	X	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>
	<15: 8>	< 7: 0>		X	X	X	X	X
3	X	X	X	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>
	<23:16>	<15: 8>	< 7: 0>		X	X	X	X
4	X	X	X	X	<63:56>	<55:48>	<47:40>	<39:32>
	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X	X

VA<2:0>	Register File Byte Positions in the two memory double words							
5	X <39:32>	X <31:24>	X <23:16>	X <15: 8>	X < 7: 0>	<63:56> X	<55:48> X	<47:40> X
6	X <47:40>	X <39:32>	X <31:24>	X <23:16>	X <15: 8>	X < 7: 0>	<63:56> X	<55:48> X
7	X <55:48>	X <47:40>	X <39:32>	X <31:24>	X <23:16>	X <15: 8>	X < 7: 0>	<63:56> X

The following table indicates the byte lanes for 64-bit references in little-endian mode:

VA<2:0>	Register File Byte Positions in the two memory double words							
0	<63:56> X	<55:48> X	<47:40> X	<39:32> X	<31:24> X	<23:16> X	<15: 8> X	< 7: 0> X
1	<55:48> X	<47:40> X	<39:32> X	<31:24> X	<23:16> X	<15: 8> X	< 7: 0> X	X <63:56>
2	<47:40> X	<39:32> X	<31:24> X	<23:16> X	<15: 8> X	< 7: 0> X	X <63:56>	X <55:48>
3	<39:32> X	<31:24> X	<23:16> X	<15: 8> X	< 7: 0> X	X <63:56>	X <55:48>	X <47:40>
4	<31:24> X	<23:16> X	<15: 8> X	< 7: 0> X	X <63:56>	X <55:48>	X <47:40>	X <39:32>
5	<23:16> X	<15: 8> X	< 7: 0> X	X <63:56>	X <55:48>	X <47:40>	X <39:32>	X <31:24>
6	<15: 8> X	< 7: 0> X	X <63:56>	X <55:48>	X <47:40>	X <39:32>	X <31:24>	X <23:16>
7	< 7: 0> X	X <63:56>	X <55:48>	X <47:40>	X <39:32>	X <31:24>	X <23:16>	X <15: 8>

Unaligned Load Word ULW



Format: ULW rt, offset(base) **CVM**

The ULW instruction does not exist when CvmCtl[USEUN] is clear. (The MIPS LWL and LWR instructions do not exist when CvmCtl[USEUN] is set.)

The ULW instruction (when enabled by CvmCtl[USEUN]) consumes either the MIPS LWL or LWR major opcodes according to the following table:

CvmCtl[USEL ONLY]	BigEndianCPU (!CvmCtl[LE])	MIPS LWL opcode is:	MIPS LWR opcode is:
0	0	NOP	ULW
0	1	ULW	NOP
1	X	ULW	NOP

When CvmCtl[USEUN] is set, one of MIPS LWL and LWR major opcodes execute as NOPs, as specified in the table above.

Purpose:

To load a word from a (potentially-unaligned) memory location

Description: `rt = memory[base+offset]`

The contents of the 32-bit word at the memory location specified by the effective address are fetched, sign-extended to 64-bits, and placed in GPR rt. The 16-bit signed offset is added to the contents of GPR base to form the effective address.

Restrictions:

Whenever any of the bytes required to service the unaligned load reside in either I/O space or DSEG or CVMSEG IO (note NOT normal/cacheable memory space and CVMSEG LM), the effective address must be naturally-aligned. If any of the two least-significant bits of the address is non-zero in this case, an Address Error exception occurs. Note that these alignment address error exceptions are lower-priority than TLB refill and invalid exceptions, though.

CVMSEG IO references cause address errors.

CVMSEG LM references by these loads cause an address error when outside the legal range allowed by CvmMemCtl[LMEMSZ] (assuming CVMSEG LM is enabled by CvmMemCtl[CVMSEGENA*]).

Operation:

```
vAddr = sign_extend(offset) + GPR[base]
if (vAddr<1:0> != 0) then
   if (vAddr is CVMSEG IO and CVMSEG is enabled)
        or (vAddr is CVMSEG LM and CVMSEG is enabled and address is out of range) then
      SignalException(AddressError)
   else
      pAddr = AddressTranslation(vAddr, DATA, LOAD)
      if (pAddr is IO) or (pAddr is DSEG) then
         SignalException(AddressError)
      else if vAddr<2> then
         // requires two aligned 64 bit fetches
         if (vAddr+8 is CVMSEG IO and CVMSEG is enabled)
             or (vAddr+8 is CVMSEG LM and CVMSEG is enabled and address is out of range) then
            SignalException(AddressError)
         else
            pAddr2 = AddressTranslation(vAddr+8, DATA, LOAD)
            if (pAddr2 is IO) or (pAddr2 is DSEG) then
               SignalException(AddressError)
            else
               memdoubleword0 = LoadMemory (CCA, DOUBLEWORD, pAddr, vAddr, DATA)
               memdoubleword1 = LoadMemory (CCA, DOUBLEWORD, pAddr2, vAddr+8, DATA)
               if (BigEndianCPU) then
                  memword = (memdoubleword0 << (vAddr<1:0> * 8)) |
                            (memdoubleword1 >> ((8-vAddr<1:0>) * 8));
               else
                  memword = (memdoubleword0 >> ((4+vAddr<1:0>) * 8)) |
                            (memdoubleword1 << ((4-vAddr<1:0>) * 8));
               endif
               GPR[rt] = sign_extend(memword)
            endif
         endif
      else
         // all bytes in one aligned 64 bit fetch
         memdoubleword = LoadMemory (CCA, DOUBLEWORD, pAddr, vAddr, DATA)
         if (BigEndianCPU) then
            memword = memdoubleword >> ((4 - vAddr<1:0>) * 8)
         else
            memword = memdoubleword >> (vAddr<1:0> * 8)
         endif
         GPR[rt] = sign_extend(memword)
      endif
   endif
else
   pAddr = AddressTranslation(vAddr, DATA, LOAD)
   memword = LoadMemory (CCA, WORD, pAddr, vAddr, DATA)
   GPR[rt] = sign_extend(memword)
endif
```

Exceptions:

TLB Refill, TLB Invalid, Bus Error, Address Error, Watch, Breakpoint

Notes:

OCTEON Cores execute naturally-aligned LWs one cycle faster than naturally-aligned ULWs, so the LW instructions should be used rather than ULW when an address is known to be naturally-aligned.

ULW is an assembler macro that converts to MIPS LWL/LWR sequences on most MIPS assemblers.

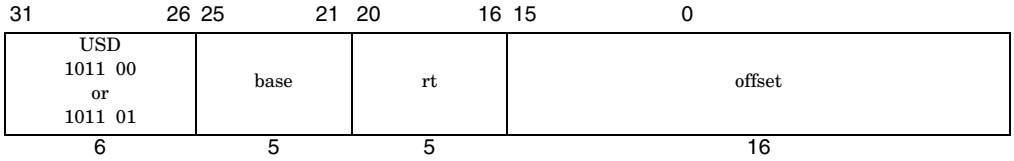
The following table indicates the byte lanes for 32-bit references in big-endian mode:

VA<2:0>	Register File Byte Positions in the two memory double words							
0	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X	X
	X	X	X	X	X	X	X	X
1	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X
	X	X	X	X	X	X	X	X
2	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X
	X	X	X	X	X	X	X	X
3	X	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X
	X	X	X	X	X	X	X	X
4	X	X	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>
	X	X	X	X	X	X	X	X
5	X	X	X	X	X	<31:24>	<23:16>	<15: 8>
	< 7: 0>	X	X	X	X	X	X	X
6	X	X	X	X	X	X	<31:24>	<23:16>
	<15: 8>	< 7: 0>	X	X	X	X	X	X
7	X	X	X	X	X	X	X	<31:24>
	<23:16>	<15: 8>	< 7: 0>	X	X	X	X	X

The following table indicates the byte lanes for 32-bit references in little-endian mode:

VA<2:0>	Register File Byte Positions in the two memory double words							
0	X	X	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>
	X	X	X	X	X	X	X	X
1	X	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X
	X	X	X	X	X	X	X	X
2	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X
	X	X	X	X	X	X	X	X
3	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X
	X	X	X	X	X	X	X	X
4	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X	X
	X	X	X	X	X	X	X	X
5	<23:16>	<15: 8>	< 7: 0>	X	X	X	X	X
	X	X	X	X	X	X	X	<31:24>
6	<15: 8>	< 7: 0>	X	X	X	X	X	X
	X	X	X	X	X	X	<31:24>	<23:16>
7	< 7: 0>	X	X	X	X	X	X	X
	X	X	X	X	X	<31:24>	<23:16>	<15: 8>

Unaligned Store Doubleword USD



Format: USD rt, offset(base) **CVM**

The USD instruction does not exist when CvmCtl[USEUN] is clear. (The MIPS SDL and SDR instructions do not exist when CvmCtl[USEUN] is set.)

The USD instruction (when enabled by CvmCtl[USEUN]) consumes either the MIPS SDL or SDR major opcodes according to the following table:

CvmCtl[USELY]	BigEndianCPU (!CvmCtl[LE])	MIPS SDL opcode is:	MIPS SDR opcode is:
0	0	NOP	USD
0	1	USD	NOP
1	X	USD	NOP

When CvmCtl[USEUN] is set, one of MIPS SDL and SDR major opcodes execute as NOPs, as specified in the table above.

Purpose:

To store a doubleword to a (potentially-unaligned) memory location

Description: `memory[base+offset] = rt`

The 64-bit doubleword in GPR rt is stored in memory at the location specified by the effective address. The 16-bit signed offset is added to the contents of GPR base to form the effective address.

If the effective address is not naturally-aligned and the second of the two memory stores required to complete the unaligned reference encounters an Address Error, TLB Refill, TLB Invalid, or TLB Modified exception, the first of the two memory references will still complete. Effectively, the USD may partially complete.

If the effective address is not naturally-aligned, the two memory stores required to complete the unaligned reference may complete in any order (as seen by other OCTEON Cores and I/O devices).

Restrictions:

Whenever any of the bytes required to service the unaligned store reside in either I/O space or DSEG or CVMSEG IO (note NOT normal/cacheable memory space and CVMSEG LM), the effective address must be naturally-aligned. If any of the three least-significant bits of the address is non-zero in this case, an Address Error exception occurs. Note that these alignment address error exceptions are lower-priority than TLB refill, invalid, and modified exceptions, though.

CVMSEG IO references cause address errors.

CVMSEG LM references by these stores cause an address error when outside the legal range allowed by CvmMemCtl[LMEMSZ] (assuming CVMSEG LM is enabled by CvmMemCtl[CVMSEGENA*]). Stores causing address errors solely due to this hardware range check may corrupt cache locations, however.

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

Operation:

```
if (Are64bitOperationsEnabled() then
   vAddr = sign_extend(offset) + GPR[base]
   datadoubleword = GPR[rt]
   if (vAddr<2:0> != 0) then
      if (vAddr is CVMSEG IO and CVMSEG is enabled)
          or (vAddr is CVMSEG LM and CVMSEG is enabled and address is out of range) then
         SignalException(AddressError)
      else
         pAddr = AddressTranslation(vAddr, DATA, STORE)
         if (pAddr is IO) or (pAddr is DSEG) then
            SignalException(AddressError)
         else
            if (BigEndianCPU) then
               StoreMemory (CCA, DOUBLEWORD-vAddr<2:0>, datadoubleword >> ((vAddr<2:0>) * 8), pAddr, vAddr, DATA)
            else
               StoreMemory (CCA, DOUBLEWORD-vAddr<2:0>, datadoubleword << ((vAddr<2:0>) * 8), pAddr, vAddr, DATA)
            endif
            if (vAddr+8 is CVMSEG IO and CVMSEG is enabled)
                or (vAddr+8 is CVMSEG LM and CVMSEG is enabled and address is out of range) then
               SignalException(AddressError)
            else
               pAddr2 = AddressTranslation(vAddr+8, DATA, STORE)
               if (paddr2 is IO) or (paddr2 is DSEG) then
                  SignalException(AddressError)
               else
                  if (BigEndianCPU) then
                    StoreMemory (CCA, DOUBLEWORD-vAddr<2:0>, datadoubleword << ((8-vAddr<2:0>) * 8), pAddr, vAddr, DATA)
                    else
                    StoreMemory (CCA, DOUBLEWORD-vAddr<2:0>, datadoubleword >> ((8-vAddr<2:0>) * 8), pAddr, vAddr, DATA)
                    endif
               endif
            endif
         endif
      endif
   else
      pAddr = AddressTranslation(vAddr, DATA, STORE)
      StoreMemory (CCA, DOUBLEWORD, datadoubleword, pAddr, vAddr, DATA)
   endif
else
   SignalException(ReservedInstruction)
endif
```

Exceptions:

TLB Refill, TLB Invalid, TLB Modified, Bus Error, Address Error, Reserved Instruction, Watch, Breakpoint

Notes:

OCTEON Cores execute naturally-aligned SDs one cycle faster than naturally-aligned USDs, so the SD instructions should be used rather than USD when an address is known to be naturally-aligned.

USD is an assembler macro that converts to MIPS SDL/SDR sequences on most MIPS assemblers.

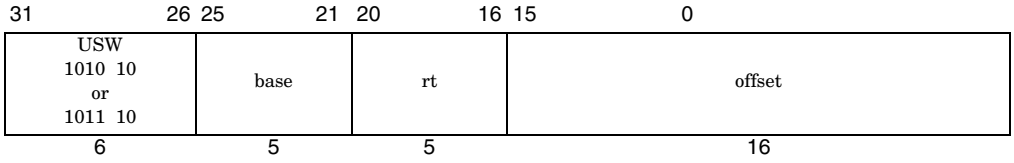
The following table indicates the byte lanes for 64-bit references in big-endian mode:

VA<2:0>	Register File Byte Positions in the two memory double words							
0	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>	<15: 8>	< 7: 0>
	X	X	X	X	X	X	X	X
1	X	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>	<15: 8>
	< 7: 0>	X	X	X	X	X	X	X
2	X	X	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>
	<15: 8>	< 7: 0>	X	X	X	X	X	X
3	X	X	X	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>
	<23:16>	<15: 8>	< 7: 0>	X	X	X	X	X
4	X	X	X	X	<63:56>	<55:48>	<47:40>	<39:32>
	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X	X
5	X	X	X	X	X	<63:56>	<55:48>	<47:40>
	<39:32>	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X
6	X	X	X	X	X	X	<63:56>	<55:48>
	<47:40>	<39:32>	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X
7	X	X	X	X	X	X	X	<63:56>
	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>	<15: 8>	< 7: 0>	X

The following table indicates the byte lanes for 64-bit references in little-endian mode:

VA<2:0>	Register File Byte Positions in the two memory double words							
0	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>	<15: 8>	< 7: 0>
	X	X	X	X	X	X	X	X
1	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>	<15: 8>	< 7: 0>	X
	X	X	X	X	X	X	X	<63:56>
2	<47:40>	<39:32>	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X
	X	X	X	X	X	X	<63:56>	<55:48>
3	<39:32>	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X
	X	X	X	X	X	<63:56>	<55:48>	<47:40>
4	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X	X
	X	X	X	X	<63:56>	<55:48>	<47:40>	<39:32>
5	<23:16>	<15: 8>	< 7: 0>	X	X	X	X	X
	X	X	X	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>
6	<15: 8>	< 7: 0>	X	X	X	X	X	X
	X	X	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>
7	< 7: 0>	X	X	X	X	X	X	X
	X	<63:56>	<55:48>	<47:40>	<39:32>	<31:24>	<23:16>	<15: 8>

Unaligned Store Word — USW



Format: USW rt, offset(base) **CVM**

The USW instruction does not exist when CvmCtl[USEUN] is clear. (The MIPS SWL and SWR instructions do not exist when CvmCtl[USEUN] is set.)

The USW instruction (when enabled by CvmCtl[USEUN]) consumes either the MIPS SWL or SWR major opcodes according to the following table:

CvmCtl[USELY]	BigEndianCPU (!CvmCtl[LE])	MIPS SWL opcode is:	MIPS SWR opcode is:
0	0	NOP	USW
0	1	USW	NOP
1	X	USW	NOP

When CvmCtl[USEUN] is set, one of MIPS SWL and SWR major opcodes execute as NOPs, as specified in the table above.

Purpose:

To store a word to a (potentially-unaligned) memory location

Description: `memory[base+offset] = rt`

The 64-bit word in GPR rt is stored in memory at the location specified by the effective address. The 16-bit signed offset is added to the contents of GPR base to form the effective address.

If the effective address is not naturally-aligned and two memory stores are required to complete the unaligned reference and the second of the two memory stores encounters an Address Error, TLB Refill, TLB Invalid, or TLB Modified exception, the first of the two memory references will still complete. Effectively, the USW may partially complete.

If the effective address is not naturally-aligned and two memory stores are required to complete the unaligned reference, the two memory stores may complete in any order (as seen by other OCTEON Cores and I/O devices).

Restrictions:

Whenever any of the bytes required to service the unaligned store reside in either I/O space or DSEG or CVMSEG IO (note NOT normal/cacheable memory space and CVMSEG LM), the effective address must be naturally-aligned. If any of the two least-significant bits of the address is non-zero in this case, an Address Error exception occurs. Note that these alignment address error exceptions are lower-priority than TLB refill, invalid, and modified exceptions, though.

CVMSEG IO references cause address errors.

CVMSEG LM references by these stores cause an address error when outside the legal range allowed by CvmMemCtl[LMEMSZ] (assuming CVMSEG LM is enabled by CvmMemCtl[CVMSEGENA*]). Stores causing address errors solely due to this hardware range check may corrupt cache locations, however.

Operation:

```
vAddr = sign_extend(offset) + GPR[base]
dataword = GPR[rt]<31:0>
if (vAddr<1:0> != 0) then
   if (vAddr is CVMSEG IO and CVMSEG is enabled)
       or (vAddr is CVMSEG LM and CVMSEG is enabled and address is out of range) then
      SignalException(AddressError)
   else
      pAddr = AddressTranslation(vAddr, DATA, STORE)
      if (pAddr is IO) or (pAddr is DSEG) then
         SignalException(AddressError)
      else if (vAddr<2>) then
         if (BigEndianCPU) then
            StoreMemory (CCA, WORD-vAddr<1:0>, dataword >> (vAddr<2:0> * 8), pAddr, vAddr, DATA)
         else
            StoreMemory (CCA, WORD-vAddr<1:0>, dataword << ((4+vAddr<2:0>) * 8), pAddr, vAddr, DATA)
         endif
         if (vAddr+8 is CVMSEG IO and CVMSEG is enabled)
             or (vAddr+8 is CVMSEG LM and CVMSEG is enabled and address is out of range) then
            SignalException(AddressError)
         else
            pAddr2 = AddressTranslation(vAddr+8, DATA, STORE)
            if (paddr2 is IO) or (paddr2 is DSEG) then
               SignalException(AddressError)
            else
               if (BigEndianCPU) then
                 StoreMemory (CCA, WORD-vAddr<1:0>, dataword << ((8-vAddr<1:0>) * 8), pAddr, vAddr, DATA)
               else
                 StoreMemory (CCA, WORD-vAddr<1:0>, dataword >> ((4-vAddr<1:0>) * 8), pAddr, vAddr, DATA)
               endif
            endif
         endif
      else
         if (BigEndianCPU) then
            StoreMemory (CCA, WORD, dataword << ((4-vAddr<1:0>) * 8), pAddr, vAddr, DATA)
         else
            StoreMemory (CCA, WORD, dataword << (vAddr<1:0> * 8), pAddr, vAddr, DATA)
         endif
      endif
   endif
else
   pAddr = AddressTranslation(vAddr, DATA, STORE)
   if (BigEndianCPU) then
      StoreMemory (CCA, WORD, dataword << ((1-vAddr<2>) * 8), pAddr, vAddr, DATA)
   else
      StoreMemory (CCA, WORD, dataword << (vAddr<2> * 8), pAddr, vAddr, DATA)
   endif
endif
```

Exceptions:

TLB Refill, TLB Invalid, TLB Modified, Bus Error, Address Error, Watch, Breakpoint

Notes:

OCTEON Cores execute naturally-aligned SWs one cycle faster than naturally-aligned USWs, so the SW instructions should be used rather than USW when an address is known to be naturally-aligned.

USW is an assembler macro that converts to MIPS SWL/SWR sequences on most MIPS assemblers.

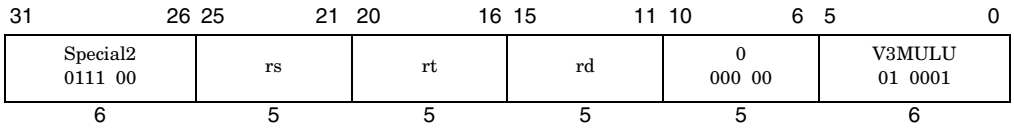
The following table indicates the byte lanes for 32-bit references in big-endian mode:

VA<2:0>	Register File Byte Positions in the two memory double words							
0	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X	X
	X	X	X	X	X	X	X	X
1	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X
	X	X	X	X	X	X	X	X
2	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X
	X	X	X	X	X	X	X	X
3	X	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X
	X	X	X	X	X	X	X	X
4	X	X	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>
	X	X	X	X	X	X	X	X
5	X	X	X	X	X	<31:24>	<23:16>	<15: 8>
	< 7: 0>	X	X	X	X	X	X	X
6	X	X	X	X	X	X	<31:24>	<23:16>
	<15: 8>	< 7: 0>	X	X	X	X	X	X
7	X	X	X	X	X	X	X	<31:24>
	<23:16>	<15: 8>	< 7: 0>	X	X	X	X	X

The following table indicates the byte lanes for 32-bit references in little-endian mode:

VA<2:0>	Register File Byte Positions in the two memory double words							
0	X	X	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>
	X	X	X	X	X	X	X	X
1	X	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X
	X	X	X	X	X	X	X	X
2	X	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X
	X	X	X	X	X	X	X	X
3	X	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X
	X	X	X	X	X	X	X	X
4	<31:24>	<23:16>	<15: 8>	< 7: 0>	X	X	X	X
	X	X	X	X	X	X	X	X
5	<23:16>	<15: 8>	< 7: 0>	X	X	X	X	X
	X	X	X	X	X	X	X	<31:24>
6	<15: 8>	< 7: 0>	X	X	X	X	X	X
	X	X	X	X	X	X	<31:24>	<23:16>
7	< 7: 0>	X	X	X	X	X	X	X
	X	X	X	X	X	<31:24>	<23:16>	<15: 8>

192-bit × 64-bit Unsigned Multiply and Add — V3MULU



Format: V3MULU rd, rs, rt **CVM**

Purpose:

To perform a 192-bit x 64-bit unsigned multiply and add yielding a 256-bit result.

Description: `(P2 || P1 || P0 || rd) = (0^64 || P2 || P1 || P0) + (0^192 || rt) + rs × (MPL2 || MPL1 || MPL0)`

The 64-bit doubleword value in GPR rs is multiplied by the 192-bit product registers MPL0-MPL2, treating both as unsigned values, producing a 256-bit result. The 64-bit doubleword value in GPR rt and the 64-bit product registers P0-P2 are zero-extended and added to the 256-bit result. The least-significant 64 bits of the result is placed in GPR rd. The most-significant 192 bits of the result is placed into the product registers P0-P2.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

If a V3MULU is preceded by an MTP* instruction (without an intervening MTM*/VMM0, VMULU/V3MULU instruction), the MTP* instruction must be preceded by an MTM* or VMM0 instruction (without any intervening VMULU/V3MULU instructions).

If a V3MULU precedes a VMULU/VMM0, there must be an intervening MTM*/VMM0 between the two.

No overflow or other arithmetic exception occurs under any circumstances.

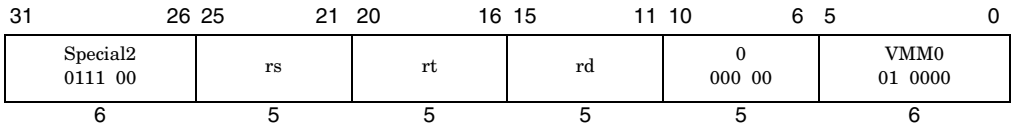
Operation:

```
if Are64bitOperationsEnabled() and !CvmCtl[NOMUL] then
   product_register<191:0> = MPL2<63:0> || MPL1<63:0> || MPL0<63:0>
   product<255:0> = GPR[rs]<63:0> * product_register<191:0>
   Pext<255:0> = 0^64 || P2<63:0> || P1<63:0> || P0<63:0>
   rtext<127:0> = 0^192 || GPR[rt]<63:0>
   sum<255:0> = product<255:0> + Pext<255:0> + rtext<255:0>
   GPR[rd] = sum<63:0>
   P2 = sum<255:192>
   P1 = sum<191:128>
   P0 = sum<127:64>
else
   SignalException(ReservedInstruction)
endif
```

Exceptions:

Reserved Instruction

64-bit Unsigned Multiply and Add Move — VMM0



Format: VMM0 rd, rs, rt **CVM**

Purpose:

To perform a 64-bit × 64-bit unsigned multiply and add yielding a 64-bit result.

Description:

```
rd = P0 + rt + rs × MPL0
MPL0 = rd
P0, P1, P2 = 0
```

The 64-bit doubleword value in GPR rs is multiplied by the 64-bit product register MPL0, treating both as unsigned values. The 64-bit doubleword value in GPR rt and the 64-bit product register P0 are added to the result. The 64-bit result is placed in GPR rd and the multiplier register MPL0. The product registers P0-P2 are zeroed.

MPL1 and MPL2 are unpredictable after this operation executes.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

If a VMM0 instruction is preceded by an MTP* instruction (without an intervening MTM*/VMM0, VMULU/V3MULU instruction), the MTP* instruction must be preceded by an MTM* or VMM0 instruction (without any intervening VMULU/V3MULU instructions).

If a VMM0 is preceded by a V3MULU, there must be an intervening MTM*/VMM0 between the two.

No overflow or other arithmetic exception occurs under any circumstances.

Operation:

```
if Are64bitOperationsEnabled() and !CvmCtl[NOMUL] then
   product<63:0> = GPR[rs]<63:0> * MPL0<63:0>
   sum<63:0> = product<63:0> + P0<63:0> + GPR[rt]<63:0>
   GPR[rd] = sum<63:0>
   MPL0 = sum<63:0>
   MPL1 = unpredictable
   MPL2 = unpredictable
   P0 = 0
   P1 = 0
   P2 = 0
else
   SignalException(ReservedInstruction)
endif
```

Exceptions:

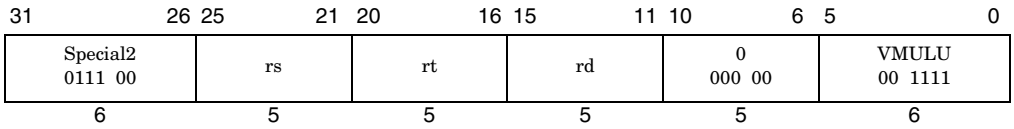
Reserved Instruction

Programming Notes:

VMM0 rd, rs, rt is functionally identical to the two-instruction sequence:

```
VMULU rd, rs, rt
MTM0 rd
```

64-bit Unsigned Multiply and Add — VMULU



Format: VMULU rd, rs, rt **CVM**

Purpose:

To perform a 64-bit x 64-bit unsigned multiply and add yielding a 128-bit result.

Description: `(P0 || rd) = (0^64 || P0) + (0^64 || rt) + rs× MPL0`

The 64-bit doubleword value in GPR rs is multiplied by the 64-bit product register MPL0, treating both as unsigned values, producing a 128-bit result. The 64-bit doubleword value in GPR rt and the 64-bit product register P0 are zero-extended and added to the 128-bit result. The least-significant 64 bits of the result is placed in GPR rd. The most-significant 64 bits of the result is placed into the product register P0.

P1, P2, MPL1, and MPL2 are unpredictable after this operation executes.

Restrictions:

A Reserved Instruction Exception is signaled if access to 64-bit operations is not enabled.

If a VMULU is preceded by an MTP* instruction (without an intervening MTM*/VMM0, VMULU/V3MULU instruction), the MTP* instruction must be preceded by an MTM* or VMM0 instruction (without any intervening VMULU/V3MULU instructions).

If a VMULU is preceded by a V3MULU, there must be an intervening MTM*/VMM0 between the two.

No overflow or other arithmetic exception occurs under any circumstances.

Operation:

```
if Are64bitOperationsEnabled() and !CvmCtl[NOMUL] then
   product<127:0> = GPR[rs]<63:0> * MPL0<63:0>
   Pext<127:0> = 0^64 || P0<63:0>
   rtext<127:0> = 0^64 || GPR[rt]<63:0>
   sum<127:0> = product<127:0> + Pext<127:0> + rtext<127:0>
   GPR[rd] = sum<63:0>
   P0 = sum<127:64>
   P1 = unpredictable
   P2 = unpredictable
   MPL1 = unpredictable
   MPL2 = unpredictable
else
   SignalException(ReservedInstruction)
endif
```

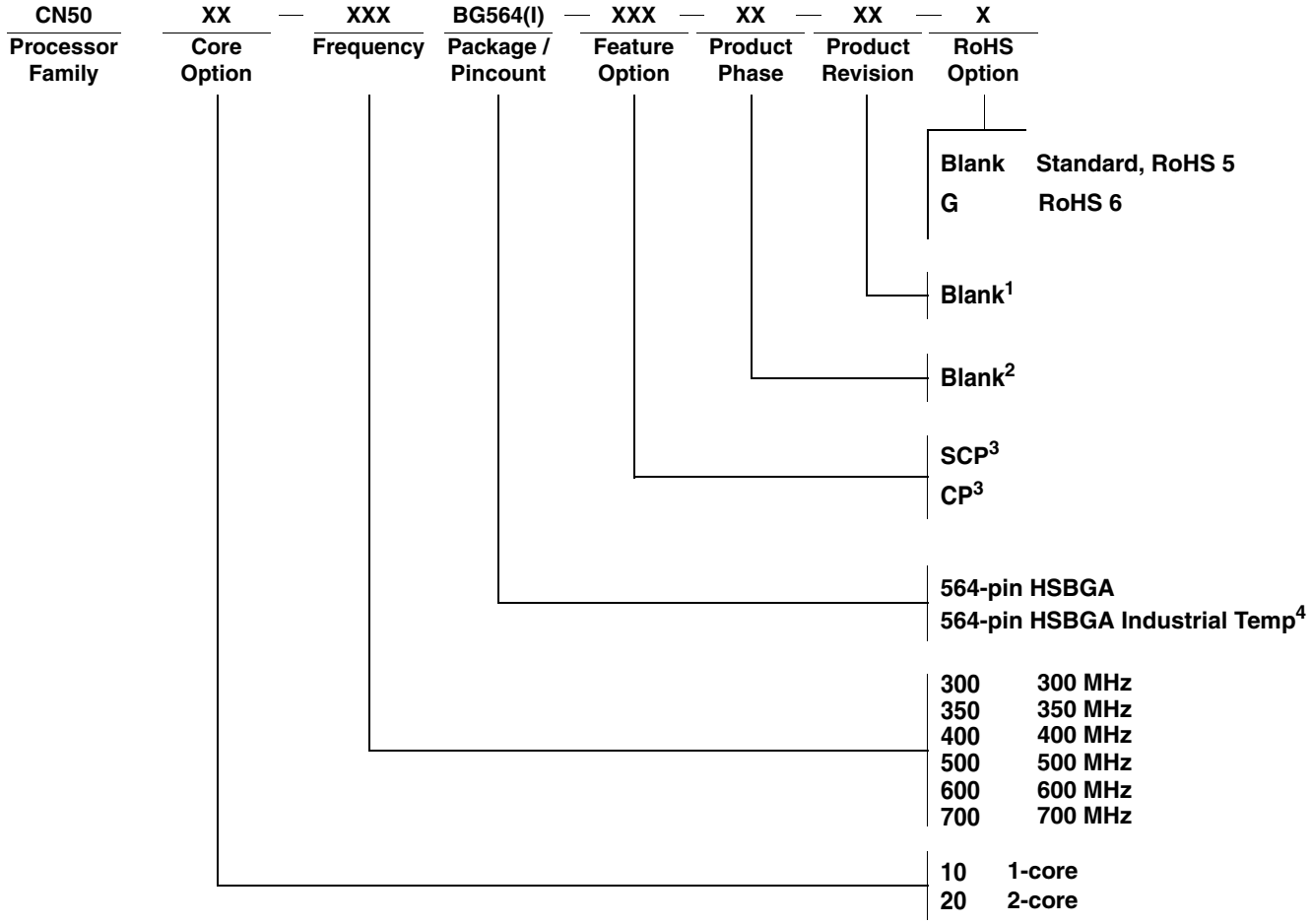
Exceptions:

Reserved Instruction

Appendix B

Ordering Information

The following shows the breakdown of the CN50XX family part numbers.



[1] Blank No revision suffix is required.

[2] Blank No phase suffix is required.

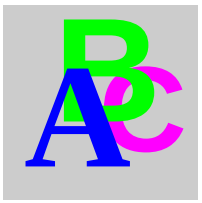
[3] SCP Secure Communications Processor includes networking, security, and TCP acceleration.
CP Communications Processor includes networking and TCP acceleration.

[4] I I Temp option is available only for 300MHz and 500 MHz parts with the -G RoHS option.

Glossary

CMB	Coherent Memory Bus - Includes the Address/Command Bus (ADD), the Store Data Bus (STORE), the Commit/Response Control Bus (COMMIT), and the Fill Data Bus (FILL).
Dcache	L1 Data Cache (16KB 32-way set associative, write-through/no-write allocate policy).
DWB	Don't-write-back
DuTag	Duplicate Tag Store of the Core L1 Cache used to maintain coherency within the L2 cache controller
FPA	Free Page Allocater
I/O Bus	I/O Bus: used to describe the bus on the far side of the I/O Bridge which contains multiple I/O agents that communicate to the Cores and memory.
IOB	I/O Bridge
IPD	Input Packet Data
KEY	Key Memory
L1	1st Level (or Level 1) Cache. For the OCTEON family, the L1 is not a subset of the L2 cache (non-inclusive).
L2	2nd Level (or Level 2) Cache.
LDL	Load Lock Instruction (used to support atomicity for semaphores in shared memory.
PWB	Core Write Buffer which holds BOTH memory and I/O writes.
Reflection	CN50XX reflection occurs in the Level 2 Controller (L2C). It is a technique programmed into the L2C whereby the L2C receives certain types of data streams and loops (reflects) them back out over the bus(es) to the sender(s). The effect is Tri-stating without having to physically Tristate.
RNG	Random Number Memory
RSL	Register Slave Logic
STC	Store Conditional Instruction (used to support atomicity for semaphores in shared memory.
STDN	Write completion
STIN	Write invalidate

Index



D

P

Q

R

Inside Back Cover

Cavium Networks
805 East Middlefield Road,
Mountain View, CA 94043
Telephone: +1-650-623-7000
Fax: +1-650-625-9751
Email: info@caviumnetworks.com