# Configuration

## TABLE OF CONTENTS

TABLE OF CONTENTS ..... 1
LIST OF TABLES ..... 3
LIST OF FIGURES ..... 3
1 Introduction ..... 4
1.1 Conventions Used In this Chapter ..... 5
2 Configuration Overview ..... 5
2.1 Getting Started ..... 5
2.2 Configuration Steps ..... 5
3 Linux ..... 6
3.1 Configuring the Linux Kernel (`make menuconfig`) ..... 6
3.1.1 SE-UM Applications: FPA Buffer Access ..... 6
3.1.2 SE-UM Applications: Configuration and Status (CSR) Register Access ..... 7
3.2 The Cavium Networks Ethernet Driver ..... 7
3.2.1.1 Ethernet Driver: Configuring the Number of Packet Data Buffers ..... 8
3.3 Configuring the Simple Executive (SE) Library for Linux ..... 9
3.3.1 SE Library Configuration for Linux: SDK 1.9 and Below ..... 9
3.3.2 SE Library Configuration for Linux: SDK 2.0 ..... 9
4 Simple Executive Library Configuration ..... 9
4.1 Simple Executive Configuration Overview ..... 11
4.1.1 Input Files to the Configuration Utility ..... 11
4.1.2 All Software Running Must Agree on the Configuration ..... 13
4.2 Configuration Utility ..... 13
4.2.1 Configuration Utility Grammar ..... 14
4.3 Required Include Files ..... 15
4.3.1 Essential Include Files ..... 15
4.3.2 Hardware Unit-Specific Include Files ..... 15
5 Application Configuration Steps ..... 16
5.1 Start with an Existing Example ..... 16
5.2 Edit the Configuration Files as Needed ..... 17
5.3 Edit the Makefile as Needed ..... 17
5.4 Configure and Build the Application ..... 17
6 Configuration Limitations ..... 17
7 Configuring `executive-config.h` ..... 18
8 Configuring `cvmx-resources.config` ..... 22
8.1 Enabling Creation of Default Pools and Scratchpads ..... 24

8.2 Configuring Buffers and Buffer Pool Definitions ..... 24
8.2.1 Unprotected pools ..... 26
8.3 Configuring Scratchpad Areas ..... 26
8.3.1 Permanent Scratchpads ..... 28
8.4 Configuring Fetch and Add (FAU) Register Resources ..... 28
9 Adding a Custom `*-config.h` File ..... 29
9.1 Example of Using the `define` Keyword ..... 30
10 Configuration Output File (`cvmx_config.h`) Contents ..... 31
11 Adding Custom Configuration While Using the SDK API ..... 31
12 "Unprotected" Buffer Pools ..... 34

# LIST OF TABLES

Table 1: Three Essential Include Files ..... 15
Table 2: Example Hardware Unit Include Files ..... 16
Table 3: Configuration Variables are Provided for Some Options ..... 18
Table 4: Other Options are Set Without a Configuration Variable ..... 18
Table 5: Definitions in `executive-config.h` ..... 19
Table 6: How to Enable Creation of Commonly-Used Pools and Scratchpads ..... 24
Table 7: Configuration Keywords for Buffer Pools ..... 25
Table 8: Configuration Keywords for Scratchpad Areas ..... 27
Table 9: Configuration Keywords for FAU Registers ..... 29
Table 10: Configuration Override Functions for PIP/IPD and PKO ..... 32

# LIST OF FIGURES

Figure 1: What Software Needs Simple Executive Configuration ..... 4
Figure 2: Configuring the Simple Executive ..... 11
Figure 3: Use Configuration Files with Identical Content ..... 13

# 1 Introduction

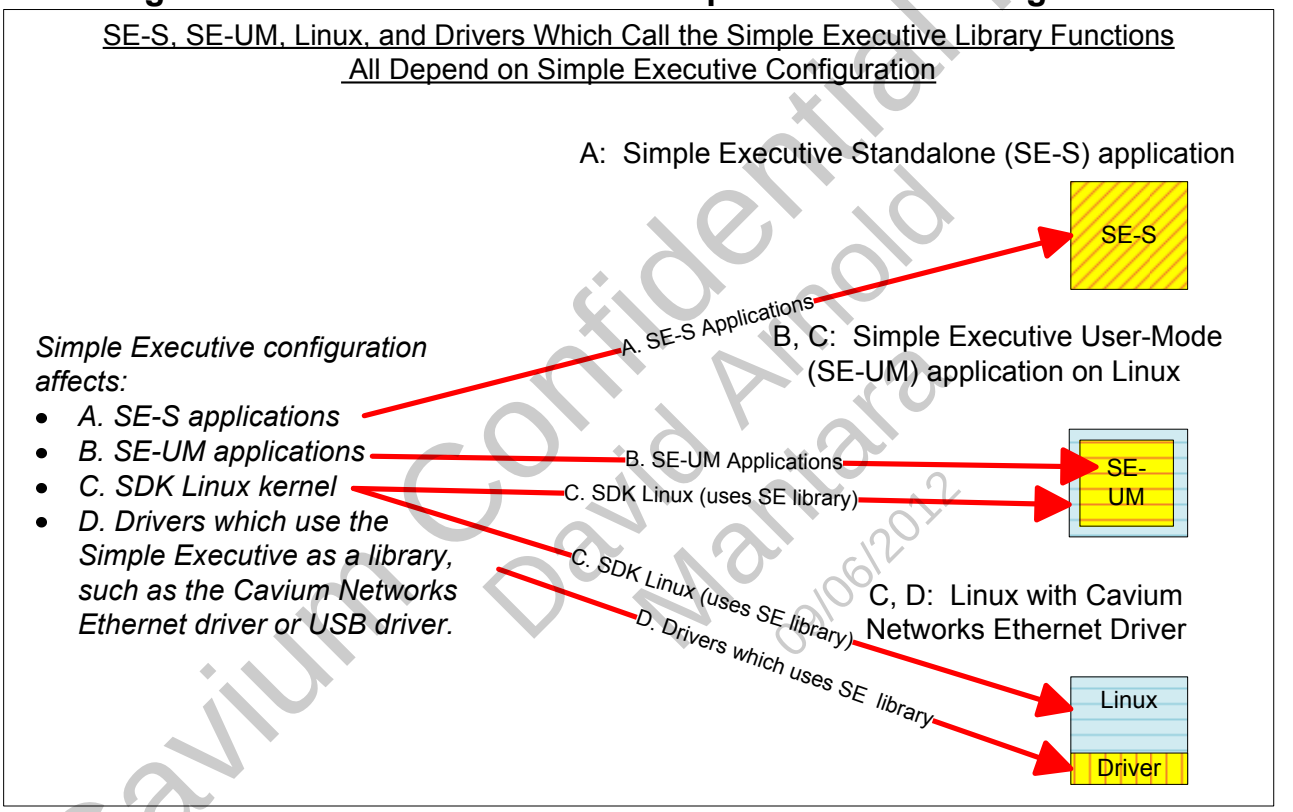
This chapter provides basic information on how to configure and customize Simple Executive. Since improper configuration can lead to runtime problems, everyone who is writing or debugging code running on OCTEON should read this chapter to become familiar with the issues.

Note that Simple Executive configuration affects:

- SE-S applications
- SE-UM applications
- SDK Linux (the Linux supplied with the SDK), which uses the Simple Executive library
- Drivers which use the Simple Executive library, such as the Cavium Networks Ethernet driver, or the USB driver

Incorrect Simple Executive configuration can result in difficult-to-debug runtime problems.

**Figure 1: What Software Needs Simple Executive Configuration**






Before reading this chapter, please read the *Essential Topics* chapter, which introduces the resources being configured.

Advanced readers who are adding code to the Simple Executive API, writing a custom API, or reading the SDK code must read the *Advanced Topics* chapter. This chapter contains information about register access, race conditions to avoid, etc.

### 1.1 Conventions Used In this Chapter

Note that in most cases the format `REGISTER[FIELD]` in this chapter refers to a hardware register and field combination, not a software `ARRAY[INDEX]`.

# 2 Configuration Overview

Ninety percent of users can use Simple Executive without further modification, and speed their application to market if they:

- Start with a suitable example program such as `linux-filter` or `passthrough`
- Make the minor configurations described in this chapter

### 2.1 Getting Started

Typically, users create a custom application by using the `passthough` or `linux-filter` example as a base: If the Cavium Networks Ethernet driver will be used with the custom application, then select `linux-filter` as a base, otherwise select the `passthough` example:

- The `passthrough` example is designed to run *without* the Cavium Networks Ethernet driver loaded. At startup, one core running a Simple Executive application configures the hardware units and populates the FPA pools.
- The `linux-filter` example is designed to run *with* the Cavium Networks Ethernet driver loaded. In this case, the driver configures the hardware units and populates only the three essential FPA pools (Packet Data buffers, WQE buffers, and PKO Command buffers). In this case, it is essential that the application not reinitialize the hardware units. For example if the application reinitializes the FPA, any buffers put in the FPA pools by the driver will become lost: neither allocated, nor free, and not a member of any pool.)

Copy the example to a new sub-directory in the `examples` directory, and then modify the code to create the custom application. (See the *Software Development Kit (SDK) Tutorial* chapter, in the section "Hands-on: Creating a Custom Application".)

### 2.2 Configuration Steps

The exact configuration steps depend on the software architecture:

- Simple Executive Library Configuration for SE-S and SE-UM Applications: see Section 4 – "Simple Executive Library Configuration").
- Linux Configuration (menuconfig): see Section 3.1 – "Configuring the Linux Kernel (`make menuconfig`)".
- Simple Executive Library Configuration for the Linux Kernel and Drivers: See Section 3.3 – "Configuring the Simple Executive (SE) Library for Linux".
- Cavium Networks Ethernet Driver Configuration (number of buffers): See Section 3.2 – "The Cavium Networks Ethernet Driver".

This chapter focuses on standard configuration options available via Simple Executive files which are processed by a configuration utility. The output of the configuration utility is a header file which can be included by the application. Customized hardware unit configuration is not covered in this chapter. Customization details are provided in the chapter about the specific hardware unit, such as the *PIP/IPD* chapter. Before attempting customizations beyond those provided by the

configuration utility, please read the *Advanced Topics* and *FPA* chapters. These chapters contain critical information necessary to avoid race conditions and difficult-to-debug runtime problems.

# 3 Linux

When running Linux:

- The kernel must be configured (via `make menuconfig`)
- The Simple Executive library which is used by the kernel must be configured
- If the Cavium Networks Ethernet driver is used, the number of Packet Data buffers must be configured via `make menuconfig`.

## *3.1 Configuring the Linux Kernel (`make menuconfig`)*

Linux configuration via `make menuconfig` is covered in the *Software Overview* chapter, and also in the documentation provided with the SDK.

### 3.1.1 SE-UM Applications: FPA Buffer Access

For SE-UM applications (SDK 1.9 and SDK 2.0), use `make menuconfig` to configure the kernel to enable/disable access to memory via *xkphys* addresses for SE-UM applications:

```
Allow User space to access memory directly (Allowed)  --->
                        (X) Allowed
                        ( ) Per process
                        ( ) Disabled
```

`CONFIG_CAVIUM_OCTEON_USER_MEM`:
Allows user applications to use *xkphys* addresses directly access to memory. This option allows user space direct access to shared memory not in use by Linux. This memory is suitable for use with the OCTEON hardware. Cavium Networks Simple Executive applications also share this memory. Since this bypass all of the Linux memory protection, only use this option on embedded devices where all user applications are strictly controlled.

By default, all SE-UM applications are allowed access.

If the `Allowed` or `Per Process` option is selected, a process running as *root* can enable or disable access via the `cvmx_linux_enable_xkphys_access()` Simple Executive library call. If the argument to this library call is 0, then a warning message is printed if there is an error. If the argument is 1, then no warning message will be printed if there is an error.

The `cvmx_linux_enable_xkphys_access()` library call enables access to the entire xkphys region, so both memory and CSR access are enabled.

Processes not running as root may only disable access using the system call.

Information on how 32-bit applications access these addresses is provided in the *Advanced Topics* chapter.

### 3.1.2 SE-UM Applications: Configuration and Status (CSR) Register Access

For SE-UM applications (SDK 1.9 and SDK 2.0), use `make menuconfig` to configure the kernel to enable/disable access to memory via *xkphys* addresses for SE-UM applications:

```
Allow User space to access hardware IO directly (Allowed)  --->
                        (X) Allowed
                        ( ) Per process
                        ( ) Disabled
```

`CONFIG_CAVIUM_OCTEON_USER_IO`: This option allows user applications to directly access the OCTEON hardware IO addresses (`0x1000000000000 - 0x1fffffffffff`). This allows high-performance networking applications to run in user space with minimal performance penalties. This also means a user application can bring down the entire system. Only use this option on embedded devices where all user applications are strictly controlled.

By default, all SE-UM applications are allowed access.

If the `Allowed` or `Per Process` option is selected, a process running as *root* can enable or disable access via the `cvmx_linux_enable_xkphys_access()` system call.

Processes not running as root may only disable access using the system call.

Information on how 32-bit applications access these addresses is provided in the *Advanced Topics* chapter.

## *3.2 The Cavium Networks Ethernet Driver*

When running Linux, if the Cavium Networks Ethernet driver will be run (as in `linux-filter`), then the Ethernet driver is responsible for initializing the following OCTEON hardware units: SSO, FPA, CIU, PIP, IPD, PKO, and the FAU. When the Ethernet driver is in use, applications must not reconfigure these hardware units.

The Ethernet driver populates the Packet Data buffer, WQE buffer, and PKO Command buffer FPA pools. In this case, the number of Packet Data buffers and an equal number of WQE buffers are configured by using `make menuconfig` ( `make menuconfig` steps are introduced in the *Software Overview* chapter (see the "Linux Memory Configuration Steps" section)). The kernel configuration tool only allows the user to control the number of Packet Data buffers (an identical number of WQE buffers are allocated), and the number of PKO Command buffers is fixed.

Note that the buffer size and buffer pool name and other attributes are configured via Section 3.3 "Configuring the Simple Executive (SE) Library for Linux". Since pool population is done at runtime, Simple Executive configuration does not include the number of buffers. The `make menuconfig` step is used to inform the Cavium Networks Ethernet driver the number of buffers desired.

For other buffer pools, such as Timer buffers, after the Ethernet driver is running, then the application must populate the pools. No coordination with the Ethernet driver is needed because it

doesn't use these pools. SE-UM applications follow the same steps as a SE-S application to allocate memory and populate the pools.

### 3.2.1.1 Ethernet Driver: Configuring the Number of Packet Data Buffers

Note that this discussion only applies to applications running the Cavium Networks Ethernet driver (such as `linux-filter`). The Cavium Networks Ethernet driver will populate the Packet Data buffer pool, WQE buffer pool, and PKO Command buffer pools. Only the number of Packet Data buffers is configurable. The number of WQE buffers equals the number of Packet Data buffers. The number of PKO Command buffers is not configurable: it is hard-coded. (This information applies to SDK 2.0 and lower.)

To configure the Cavium Networks-specific options, select "Machine selection". The next screen will look similar to this:

```
System type (Support for the Cavium Networks OCTEON reference board)  --->
 [*] Enable OCTEON specific options
 [ ]   Build the kernel to be used as a 2nd kernel on the same chip
 [*]   Enable support for Compact flash hooked to the OCTEON Boot Bus
 [*]   Enable hardware fixups of unaligned loads and stores
 [*]   Enable fast access to the thread pointer
 [*]   Support dynamically replacing emulated thread pointer accesses
 (2)   Number of L1 cache lines reserved for CVMSEG memory
 [*]   Lock often used kernel code in the L2
 [*]   Lock the TLB handler in L2
 [*]   Lock the exception handler in L2
 [*]   Lock the interrupt handler in L2
 [*]   Lock the 2nd level interrupt handler in L2
 [*]   Lock memcpy() in L2
 [*]   Allow User space to access hardware IO directly
 [*]   Allow User space to access memory directly
 (0)   Memory to reserve for user processes shared region (MB)
 [*]   Use wired TLB entries to access the reserved memory region
 (5000) Number of packet buffers (and work queue entries) for the Ethernet
       driver
 <M>   POW based internal only Ethernet driver
 <*>   OCTEON watchdog driver
 [ ]   Enable enhancements to the IPSec stack to allow protocol offload.
```

When using `make menuconfig`:

- Type "?" for help with a highlighted option.
- Items marked with [*] are "on". To turn them to off, change the star to a space ("*" becomes " ").

To see the configured values, look in the file `linux/kernel_2.6linux/include/linux/autoconf.h`. This file is created during the build.

Items marked with [*] are "on". To turn them to off, change the star to a space ("*" becomes " ").

## 3.3 Configuring the Simple Executive (SE) Library for Linux

The kernel will call the Simple Executive library, so Simple Executive must be configured. This configuration must be match the configuration for a SE-UM or SE-S application (shown in Section 4 – "Simple Executive Library Configuration", but different files may be involved (depending on which SDK is used, as shown below), so care must be taken when changing one configuration file to change the corresponding file.

### 3.3.1 SE Library Configuration for Linux: SDK 1.9 and Below

For SDK 1.9 and below, follow the directions for configuring Simple Executive. In addition to copying the `executive-config.h` file to the application's `config` directory, copy the file to the kernel's `config` directory:
`./kernel_2.6/linux/arch/mips/Cavium-octeon/gpl-executive/config/executive-config.h`.

The kernel uses the copy of `cvmx-resources.config` in the `$OCTEON_ROOT/executive` directory, which is in the kernel build's search path. If the user needs to customize `cvmx-resources.config`, then the copy in `$OCTEON_ROOT/executive` must be modified.

The file `cvmx-config.h`, is created as part of the kernel build (the kernel build calls the configuration utility). This file is included in the kernel and driver source files as needed.

### 3.3.2 SE Library Configuration for Linux: SDK 2.0

For SDK 2.0, the kernel configuration file is `cvmx-config.h`:
`./kernel_2.6/linux/arch/mips/include/asm/octeon/cvmx-config.h`.

This file needs to be edited directly with any changes needed to match the application's configuration. (The output of the Simple Executive application configuration is a `cvmx-config.h` file in the application's `config` directory. This file can be used as a guide when editing the kernel version of the file.)

For SDK 2.0, the `executive-config.h` file is no longer included in various Simple Executive `.c` files when building Simple Executive for the Linux kernel, as seen in `cvmx-pip.h`:

```
#ifdef CVMX_BUILD_FOR_LINUX_KERNEL
#include "cvmx-pip-defs.h"
#else
#ifndef CVMX_DONT_INCLUDE_CONFIG
#include "executive-config.h"
#endif // end CVMX_DONT_INCLUDE_CONFIG
#endif // end CVMX_BUILD_FOR_LINUX_KERNEL
```



# 4 Simple Executive Library Configuration

This section describes how to configure the Simple Executive library for use with a SE-S or SE-UM application. The same directions apply when configuring the Linux kernel, with minor exceptions such as which file to edit, or which directory contains the configuration files. Before

configuring Simple Executive for the Linux kernel, configure Simple Executive for the application. Then use that information to configure the Linux kernel:

- For SDK 1.9, the same directions are used to configure Simple Executive for the Linux kernel and drivers, but the configuration files may be located in different directories (See Section 3.3.1 – "SE Library Configuration for Linux: SDK 1.9 and Below".
- For SDK 2.0, see Section 3.3.2 – "SE Library Configuration for Linux: SDK 2.0".

***Note: It is important to understand Simple Executive configuration, and verify that the information in the configuration files is correct. Simple Executive configuration is not difficult, but incorrect configuration can lead to difficult-to-debug errors.***

In addition to configuration variables and keywords presented in this chapter, SE-S application virtual memory configuration information is located in the *Software Overview* chapter. See the *Software Overview* chapter, in the section labeled "Simple Executive Virtual Memory Configuration Options" for more information. (We strongly recommend that the 1-1 physical-to-virtual memory mapping (which is the default) be *disabled* using instructions in this section.)

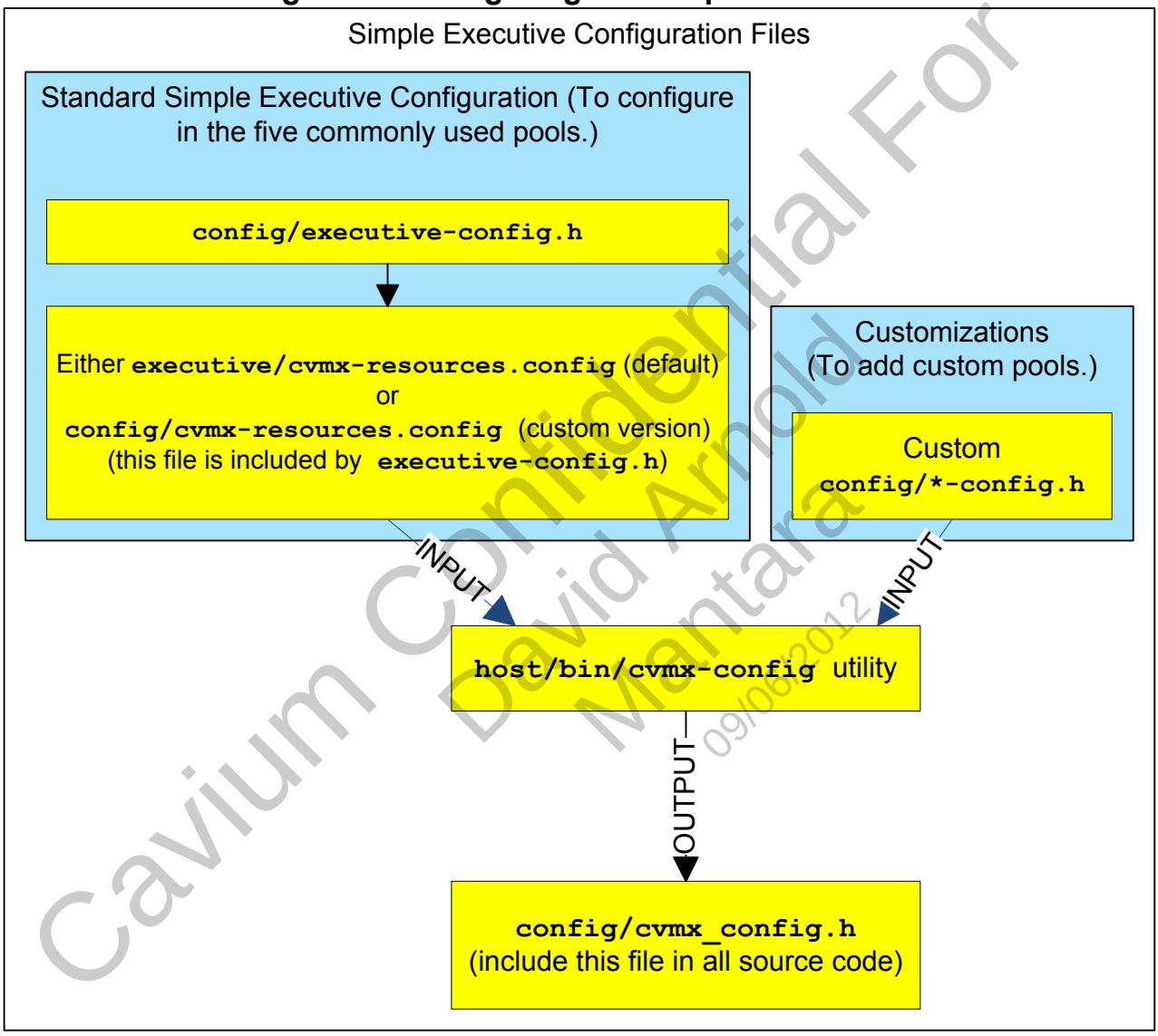
## 4.1 Simple Executive Configuration Overview

A configuration utility, `host/bin/cvmx-config` takes various files as input and creates a `cvmx-config.h` file.

### 4.1.1 Input Files to the Configuration Utility

The figure below shows the `host/bin/cvmx-config` utility processing the input configuration files to create a `cvmx-config.h` file which can then be included in all source code files:

**Figure 2: Configuring the Simple Executive**






The key input files for the `cvmx-config` utility are:

- `executive-config.h`: The template for this file is in the `executive` directory. Copy `executive-config.h.template` to the config directory in your specific

example directory, rename it `executive-config.h`, and then make modifications to the local copy. This file is used to configure hardware unit-specific configuration variables, global helper functions, and feature-specific configuration variables. Note that the `executive-config.h` file is also included by Simple Executive code, so using this name is essential.

- `cvmx-resources.config:` The file `cvmx-resources.config` is included at the end of `executive-config.h`. This file is located in the `executive` directory. If changes are to be made to the file, then copy the file to the local `config` directory, and then make modifications to the local copy. If no changes are to be made to this file, then a copy of the file located in the `config` directory is not necessary. This file is used to configure buffer, buffer pools, and scratchpads for any of the 5 commonly-used pools: Packet Data buffers, WQE buffers, PKO Command buffers, TIMER buffers, and DFA buffers. Resource configuration is done via keywords which are interpreted by the configuration utility. Note that the `cvmx-resources.config` file is included by `executive-config.h`, so using this name is essential.
- `*-config.h:` Any file in the local config directory named with the convention "`*-config.h`", such as "`my-config.h`" will be used as input to the `cvmx-config` utility. This file is typically used to create custom pools (other than the 5 standard pools). An example can be seen in the `fpa_simplified` example, which is located on support site in the same directory as the *OCTEON Programmer's Guide*. This code contains only FPA code, no code for other hardware units, and is an easy way to get acquainted with configuring FPA pools and scratchpad areas. More than one `*-config.h` file can be used; the configuration utility will process them automatically.

  ***Note: All applications running on the same OCTEON chip should share the same `config` directory to most easily and reliably use the same configuration files. For example, it is essential that they share the FPA name-to-pool-number relationship, and the same buffer sizes. Otherwise buffers can be freed to the wrong pool, and memory may be overwritten by writing off the end of a buffer.***

  ***Note: It is essential to read the FPA chapter before configuring FPA pools.***

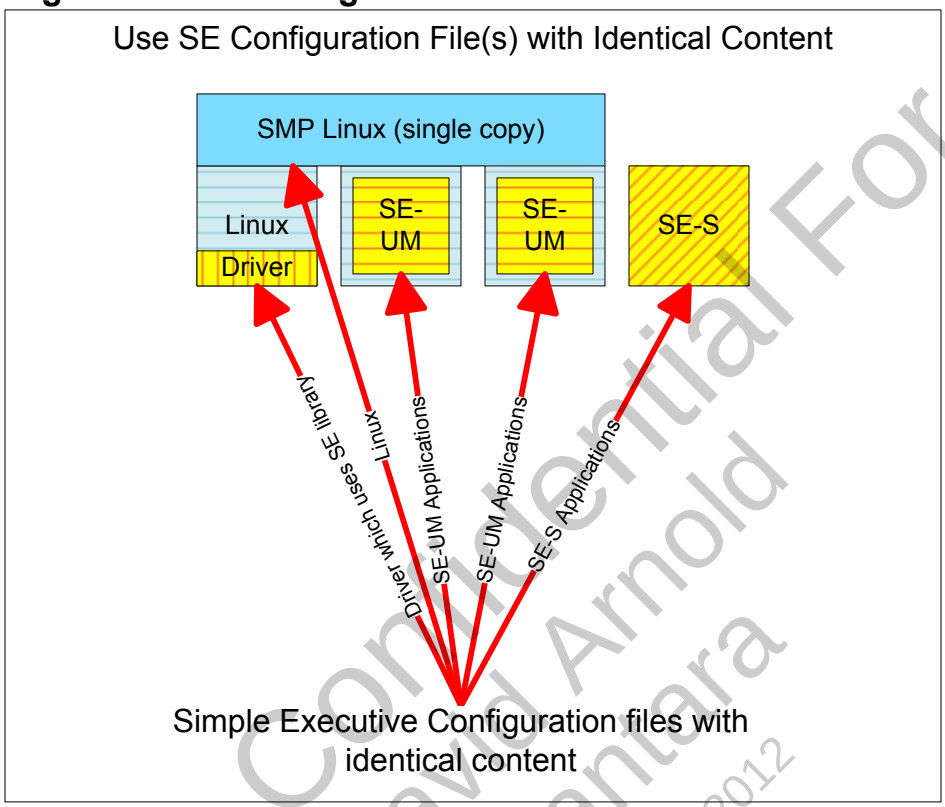
We recommend that customizations be, as much as possible, put in a custom `*-config.h` file (such as `my-config.h`). When new versions of the SDK are released, the old `executive-config.h` and `cvmx-resources.config` files will need to be compared with the new versions and updated when the SDK is updated. Minimizing the changes makes this comparison and update process simpler. (Multiple `*-config.h` files may be used if desired. The configuration utility reads all `*-config.h` files and combines them.)

***Note: The `executive-config.h` file is also directly included by many of the `.c` and `.h` source files in the executive directory, and `cvmx-resources.config` is included by `executive-config.h`, so the impact of the configuration in these files is not limited to the configuration utility and the included `cvmx-config.h` file.***

### 4.1.2 All Software Running Must Agree on the Configuration

When configuring multiple SE-UM applications to run on multi-core Linux, all SE-UM applications, the Linux kernel, and drivers which call Simple Executive functions must use configuration files with identical content. For example, if the configurations differ too greatly between `linux-filter` and the Cavium Networks Ethernet driver then runtime problems can occur. The driver and the `linux-filter` application need to exactly agree on the buffer sizes, queue to port mapping, and PIP/IPD configuration.

**Figure 3: Use Configuration Files with Identical Content**






## 4.2 Configuration Utility

The configuration utility reads the input files, ignoring all input until it sees the `cvmxconfig` keyword, so it is okay to have other code in the file (such as preprocessor directives). Once the configuration utility sees `cvmxconfig`, it expects a configuration section (enclosed in `{}`). After reading the section it ignores all input until it sees `cvmxconfig` again, then the process repeats.

Since a `cvmxconfig` block (configuration block) is not valid C, the `#ifdef CAVIUM_COMPONENT_REQUIREMENT` must precede the configuration block so that normal C files won't attempt to process it. This definition allows the C preprocessor and the configuration utility to share the same files:

```
#ifdef CAVIUM_COMPONENT_REQUIREMENT
cvmxconfig
{
    <configuration information for the tool>
}
#endif  // end CAVIUM_COMPONENT_REQUIREMENT
```

### 4.2.1 Configuration Utility Grammar

The following list is the simple grammar recognized by the configuration utility:

```
// <comment delimiter>
cvmxconfig {} // surrounds a configuration block

description "Just a string to name the config"

define NAME
        description="foo"
        value=<number>

define NAME value = <number>

#if
#else
#endif

fpa NAME
        description="foo"
        pool=<0-7>
        priority=<1-8>
        protected=<true|false, on|off, 1|0>
        size=<1-512> // in 128-byte cache lines

fau
        description="foo"
        size=<1,2,4,8>  // size is in bytes
        count=<number, no range checking>

scratch
        description="foo"
        size=<1,2,4,8>   // number of bytes per element in bytes
        count=<number, no range checking> // number of elements (default = 1)
        iobdma=<true|false, on|off, 1|0>
        permanent=<true|false, on|off, 1|0>
```



Notes:

- If the scratchpad area keyword `iobdma` is set to `true`, then `size` must be 8
- The two "define" descriptions shown above are the same: the configuration utility ignores whitespace.

- The limit of 512 cache lines (65,536 bytes) on FPA buffer size is arbitrary. Hardware units (other than the FPA) may set maximum size limits on buffers such as Packet Data buffers, but the FPA does not impose a maximum size limit. (The Packet Data buffer maximum size is 128 cache lines (16,384 bytes).)

An example of using the `#if`, `#else`, `#endif` inside the `cvmvconfig {}` block is:

```
#ifdef CAVIUM_COMPONENT_REQUIREMENT
cvmxconfig
{
    #if CVMX_LLM_CONFIG_NUM_PORTS == 2
        define VARIABLE_1 value = 2;
    #else
        define VARIABLE_1 value = 1;
    #endif // end CVMX_LLM_CONFIG_NUM_PORTS
}
#endif // end CAVIUM_COMPONENT_REQUIREMENT
```

## *4.3 Required Include Files*

User code needs to include three essential include files, and the include files for hardware unit-specific functions (such as `cvmx-fpa.h` for `cvmx_fpa_alloc()`).
This section presents the essential include files, some of the hardware unit-specific include files, and also provides an overview of the different categories of other include files in the `executive` directory to help the user sort the approximately 100 include files in the `executive` directory into those which can be ignored and those worthy of attention.

### 4.3.1 Essential Include Files

Application source files must include `cvmx_config.h`, `cvmx.h`, and any unit-specific include files. Unit-specific include files follow the first two required include files. Many headers depend on `cvmx-helper.h`, so it is best to include it before the unit-specific include files. Other than that, there is no required order for include files.

**Table 1: Three Essential Include Files**

| Include File | Purpose |
|---|---|
| `cvmx-config.h` | Created in the local config directory by the configuration utility. This file should be the first Simple Executive header included in the source code. |
| `cvmx.h` | Common macros and inline functions required by all programs running on OCTEON. This file should be the second Simple Executive header included in the source code, after `cvmx-config.h`. |
| `cvmx-helper.h` | Helper functions for common but complicated tasks. |

### 4.3.2 Hardware Unit-Specific Include Files

Hardware Unit-specific include files generally have names which match the hardware unit name. Many of these include files are included by `cvmx-helper.h`. The table below shows some of the hardware unit-specific include files.

**Table 2: Example Hardware Unit Include Files**

| Unit | Include File | Purpose |
|---|---|---|
| DFA | `cvmx-dfa.h` | DFA: Interface to the hardware DFA engine. |
| FAU | `cvmx-fau.h` | FAU: Interface to the hardware Fetch and Add Unit (FAU). |
| FPA | `cvmx-fpa.h` | FPA: Interface to the hardware Free Pool Allocator (FPA). |
| FPA | `cvmx-helper-fpa.h` | FPA: Helper functions for FPA setup. |
| IPD | `cvmx-ipd.h` | IPD: Interface to the hardware Input Packet Data (IPD) unit. |
| PIP | `cvmx-pip.h` | PIP: Interface to the hardware Packet Input Processor (PIP). |
| PKO | `cvmx-pko.h` | PKO: Interface to the hardware Packet Output (PKO) unit. |
| SSO (POW) | `cvmx-pow.h` | SSO (POW): Interface to the hardware SSO unit (also known as the Packet/Order/Work unit, or POW). |
| RNG | `cvmx-rng.h` | RNG: Functions and structure definitions for Random Number Generator (RNG) hardware. |
| TIM | `cvmx-tim.h` | TIM: Interface to the hardware TIMER unit. |
| ZIP | `cvmx-zip.h` | ZIP: Header file for the ZIP hardware unit. |

# 5 Application Configuration Steps

The next step is to create a private workspace. To create a private workspace, follow the directions in the *Software Development Kit (SDK) Tutorial* chapter to install the software and become familiar with the steps needed to configure, build, and run example code.

## *5.1 Start with an Existing Example*

Create a sub-directory in the examples directory for the custom application, for example: `my_application`. It is usually easiest to copy an existing example which has functionality similar to the desired application functionality, and then modify it. We recommend using either `passthrough` or `linux-filter` as a base application. Select `linux-filter` if the Cavium Networks Ethernet driver will be run under Linux. Other examples are not designed to do real work, so they are not sufficient to use as a base for a real-world application.

A simple example program (`fpa_simplified`) is also provided. This example can be used to test out the FPA configuration and API functions without involving other units. The `fpa_simplified` example can be downloaded from the Cavium Networks Technical Support Site in the same directory where an electronic copy of this chapter may be found. The downloadable file is a `tar` file. Untar it into the `examples` directory of the SDK. To build and run the example, follow the directions in the `README.txt` file provided with the example.

Note that the `fpa_simplified` example code only shows the FPA API, and is not useful for doing any actual work.

### 5.2 Edit the Configuration Files as Needed

The example code contains a `config` subdirectory (the *local* config directory). When the configuration utility is run, the resultant `cvmx-config.h` file is created in this directory.

The directory should contain `executive-config.h`. This file is copied from `executive/executive-config.h.template` to `config/executive-config.h`, and changes are made to the local copy. (See Section 7 – "Configuring `executive-config.h`".)

If the `cvmx-resources.config` file is not modified from the `executive/cvmx-resources.config` file, then there is no need for a local copy; otherwise, copy `executive/cvmx-resources.config` to `config/cvmx-resources.config` and make changes to the local copy. (See Section 8 – "Configuring `cvmx-resources.`".)

If custom pools are needed, create a file named `config/*-config.h` (such as the `my-config.h` file in the `fpa_simplified` directory), and put the customizations there. (See Section 9 – "Adding a Custom `*-config.h` File".)

### 5.3 Edit the Makefile as Needed

If the C source file is renamed, make any needed Makefile changes, such as changing the name of the C source file.

### 5.4 Configure and Build the Application

In the `my_application` directory, run `make` to build the application. This step will build a custom `cvmx-config.h` file in your local `config` directory. The Simple Executive source code already contains "`#include "cvmx-config.h"`", so executive source code will include the file from the local `config` directory.

The resultant target will appear in the `my_application` directory. The object files will be in `my_application/obj`.

## 6 Configuration Limitations

Simple Executive configuration supports the most commonly modified configuration items, but does not provide support for all possible configuration options. The other options are set to either hardware default values, or are set to values which are optimal for most applications. The PIP/IPD is an example of a unit which provides a rich selection of options. Details on all options supported by each hardware unit can be obtained from the HRM. Chapters in the *OCTEON Programmer's Guide*, such as the *PIP/IPD* chapter provide more information about the options from a software perspective.

At the hardware level, every register field corresponds to a variable controlling the hardware unit's behavior. These register fields are initialized to a hardware default value on power-on/reset. In

some cases, Simple Executive code will change the hardware default to either a user-configured value, or a preferred SDK default value.

Both the hardware default and SDK default values are shown in the table describing the option, for example the *PIP/IPD* chapter contains the following entry for the Parse Mode. Parse Mode can be changed via the Simple Executive configuration variable `CVMX_HELPER_INPUT_PORT_SKIP_MODE`.

**Table 3: Configuration Variables are Provided for Some Options**

| Brief Description | Register | Fields | H/W Default Value | SDK Default Value |
|---|---|---|---|---|
| Parse Mode: Parse mode (Skip-to-L2 (1), Skip-to-IP (2), or Uninterpreted (0)) | `PIP_PRT_CFGn` (one per port) | `MODE` | 0 | 1 (See Note1) |
| **Notes** | | | | |
| Note1: Configured via `executive-config.h`: `CVMX_HELPER_INPUT_PORT_SKIP_MODE = CVMX_PIP_PORT_CFG_MODE_SKIPL2` | | | | |

The SDK code often sets the register fields to reasonable values without using a configuration variable. For example, by default the size of each MBUF is set to the size of the Packet Data Buffers as shown in the following table. The divide by 8 seen below is needed to adjust the units to 8-byte words.

**Table 4: Other Options are Set Without a Configuration Variable**

| MBUF Configuration | | | | |
|---|---|---|---|---|
| MBUF size. The number of 8-byte words in an mbuf. Legal values are in the range 32-2048. (Pool 0 buffers must be a minimum of 256 bytes.) | `IPD_PACKET_MBUFF_SIZE` | `MB_SIZE` | 0x20 (256 bytes) | See Note1 |
| **Notes** | | | | |
| Note1: `IPD_PACKET_MBUFF_SIZE[MB_SIZE]` is initialized to `(CMVX_FPA_PACKET_POOL_SIZE / 8)` when `cvmx_helper_initialize_packet_io_global()` is called. | | | | |

Note: In most cases, Simple Executive doesn't change the hardware default register field value.

# 7 Configuring `executive-config.h`

To configure features into Simple Executive, modify the local copy of `executive-config.h`. Edit the file to remove the comment delimiters ("`//`") before the `#define` to enable needed functionality (see the table below). Note that enabling PKO, DFA, or TIM functions also enables the supporting FPA functions. It is recommended that the helper functions also be enabled.

For example, change:

```
/* Define to enable the use of simple executive packet output functions.
** For packet I/O setup enable the helper functions below.
*/
//#define CVMX_ENABLE_PKO_FUNCTIONS
```

to the following, by removing the comment delimiters "//":

```
/* Define to enable the use of simple executive packet output functions.
** For packet I/O setup enable the helper functions below.
*/
#define CVMX_ENABLE_PKO_FUNCTIONS
```

## Table 5: Definitions in `executive-config.h`

| Define (alphabetical order) | Default Value | Recommended Value |
|---|---|---|
| **CVMX_CONFIG_ENABLE_DEBUG_PRINTS* (See Note1)** | | |
| CVMX_CONFIG_ENABLE_DEBUG_PRINTS enables all prints in the executive. This macro should be set to 1 (TRUE). None of the debugging prints are in the fast path, so performance affect is negligible. If this macro is set to FALSE, there is no notification of failure because the prints are disables and this is the primary error-reporting mechanism. If this variable is defined in executive-config.h, cvmx-resources.config will define CVMX_ENABLE_DEBUG_PRINTS to the value of CVMX_CONFIG_ENABLE_DEBUG_PRINTS otherwise the value of CVMX_ENABLE_DEBUG_PRINTS is set to 1 (TRUE). If this variable is not defined, cvmx.h will define it to 1 (TRUE). Default=1 (TRUE). | 1 (TRUE) | 1 (TRUE) |
| **CVMX_CONFIG_NULL_POINTER_PROTECT* (See Note1)** | | |
| If CVMX_CONFIG_NULL_POINTER_PROTECT is defined to 1 (TRUE), then accesses to virtual address 0 will result in an access error (TLB trap). The low 1 MByte is reserved for bootloader and exception vectors, so protecting this space is wise. If this variable is defined in executive-config.h, cvmx-resources.config will define the variable **CVMX_NULL_POINTER_PROTECT** to the value of this variable, otherwise CVMX_NULL_POINTER_PROTECT is set to 1 (TRUE). Default=1 (TRUE). | 1 (TRUE) | 1 (TRUE) |
| **CVMX_ENABLE_DFA_FUNCTIONS** | | |
| This macro controls needed function definitions and creates FPA pool definitions for the DFA buffer pool, and a FAU register. Only DFA functions and the DFA buffer pool are affected. Default=not defined. | not defined | define if DFA will be used |
| **CVMX_ENABLE_HELPER_FUNCTIONS** | | |
| CVMX_ENABLE_HELPER_FUNCTIONS should always be defined. The only time it would not be defined is if the system was not used for Packet I/O. This macro will enable some of the cvmx_helper* functions and will also create the WQE buffer pool definitions. By default, this option is not defined; we recommend it be defined for all applications doing packet I/O. | not defined | **define** |
| **CVMX_ENABLE_LEN_M8_FIX** | | |
| See the *PIP/IPD* chapter. If this variable is not defined, cvmx-ipd.h will define it to 0 (FALSE). When CVMX_ENABLE_LEN_M8_FIX is set to 0, it causes IPD_CTL_STATUS[LEN_M8] to be cleared and the buffer next pointer includes the size of that next pointer. For performance reasons, this should always be set to 1. | 1 (TRUE) | 1 (TRUE) |
| **CVMX_ENABLE_PKO_FUNCTIONS (See Note2)** | | |
| This macro controls needed function definitions, FPA pool definitions, and other PKO-related definitions. For packet I/O, always define CVMX_ENABLE_PKO_FUNCTIONS. If CVMX_ENABLE_HELPER_FUNCTIONS is defined, then a test in executive-resources.h will force CVMX_ENABLE_PKO_FUNCTIONS to be defined. By default, this option is not defined; we recommend it be defined for all applications doing packet I/O. | not defined | **define** |
| **CVMX_ENABLE_TIMER_FUNCTIONS (See Note3)** | | |
| This macro controls needed function definitions and creates FPA pool definitions for the TIMER buffer pool. Only TIMER functions and the TIMER buffer pool are affected. Default=not defined. | not defined | define if TIMERS will be used |

| Define (alphabetical order) | Default Value | Recommended Value |
|---|---|---|
| **CVMX_HELPER_DISABLE_RGMII_BACKPRESSURE** | | |
| See Note4. If this macro is set to 1 (TRUE), then GMX[0,1]_TX_OVR_BP[EN] is set to 0xF and GMX[0,1]_TX_OVR_BP[IGN_FULL] is set to 0xF, causing the GMX RX FIFO to be ignored when computing backpressure for all 4 GMX ports. See the PIP/IPD chapter for a system-level view of backpressure. GMX has three sources of BP: 1) IPD, 2) GMX RX FIFO, 3) software via CSRs. GMX0/1/0_TX_OVR_BP IGN_FULL will only ever affect #2. Default=1 (TRUE), disabling RGMII backpressure based on RX FIFO full; we recommend the value be set to 0 (FALSE). | 1 (TRUE) | **0 (FALSE)** |
| **CVMX_HELPER_DISABLE_SPI4000_BACKPRESSURE** | | |
| See Note4. This should always be set to 0 (FALSE) so that backpressure is always enabled. Turning off backpressure on the SPI interface is disastrous and can result in protocol errors such as losing the end of packet markers. If this macro is set to 1 (TRUE) the SPI4000 will not stop sending packets when receiving backpressure; it will also not generate backpressure packets when its internal FIFOs are full. The default=1 (TRUE), disabling SPI 4000 backpressure; we recommend the value be set to 0 (FALSE). | 1 (TRUE) | **0 (FALSE)** |
| **CVMX_HELPER_ENABLE_BACK_PRESSURE** | | |
| See the PIP/IPD chapter. If this variable is not defined cvmx-helper-check-defines.h will define it to 1 (TRUE), and print a warning (see Note5). Default=1 (TRUE). (Note this macro is not present in SDK 2.0.) | 1 (TRUE) | define to 1 (TRUE) |
| **CVMX_HELPER_ENABLE_IPD** | | |
| See the *PIP/IPD* chapter. If this variable is not defined `cvmx-helper-check-defines.h` will define it to 1 (TRUE), and print a warning (see Note5). Default=1 (TRUE). | 1 (TRUE) | depends on architecture |
| **CVMX_HELPER_FIRST_MBUFF_SKIP** | | |
| See the *PIP/IPD* chapter. If this variable is not defined `cvmx-helper-check-defines.h` will define it to 184, and print a warning (see Note5). The file `cvmx-resources.config` will issue an error if the value is > 256. Default=184 bytes. | 184 bytes | depends on architecture |
| **CVMX_HELPER_INPUT_PORT_SKIP_MODE** | | |
| See the *PIP/IPD* chapter. If this variable is not defined `cvmx-helper-check-defines.h` will define it to `CVMX_PIP_PORT_CFG_MODE_SKIPL2`, and print a warning (see Note5). Default=`CVMX_PIP_PORT_CFG_MODE_SKIPL2` (Skip to L2) | Skip to L2 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_INPUT_PORT** | | |
| See the *PIP/IPD* chapter. If this variable is not defined `cvmx-helper-check-defines.h` will print an error (see Note5). Default=1. | 1 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_IPV4_DST_IP** | | |
| See the *PIP/IPD* chapter. Default=0. | 0 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_IPV4_DST_PORT** | | |
| See the *PIP/IPD* chapter. Default=0. | 0 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_IPV4_PROTOCOL** | | |
| See the *PIP/IPD* chapter. Default=0. | 0 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_IPV4_SRC_IP** | | |
| See the *PIP/IPD* chapter. Default=0. | 0 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_IPV4_SRC_PORT** | | |
| See the *PIP/IPD* chapter. Default=0. | 0 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_IPV6_DST_IP** | | |
| See the *PIP/IPD* chapter. Default=0. | 0 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_IPV6_DST_PORT** | | |
| See the *PIP/IPD* chapter. Default=0. | 0 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_IPV6_NEXT_HEADER** | | |
| See the *PIP/IPD* chapter. Default=0. | 0 | depends on architecture |

| Define (alphabetical order) | Default Value | Recommended Value |
|---|---|---|
| **CVMX_HELPER_INPUT_TAG_IPV6_SRC_IP** | | |
| See the *PIP/IPD* chapter. Default=0. | 0 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_IPV6_SRC_PORT** | | |
| See the *PIP/IPD* chapter. Default=0. | 0 | depends on architecture |
| **CVMX_HELPER_INPUT_TAG_TYPE** | | |
| See the *PIP/IPD* chapter. If this variable is not defined `cvmx-helper-check-defines.h` will define it to `CVMX_POW_TAG_TYPE_ORDERED`, and print a warning (see Note5). Default=`CVMX_POW_TAG_TYPE_ORDERED` (ORDERED) | ORDERED | depends on architecture |
| **CVMX_HELPER_IPD_DRAM_MODE** | | |
| See the *PIP/IPD* chapter. This configuration variable is new with SDK 2.0. Default=`CVMX_IPD_OPC_MODE_STT` (all blocks stored to DRAM). | All blocks stored to DRAM | depends on architecture |
| **CVMX_HELPER_NOT_FIRST_MBUFF_SKIP** | | |
| See the *PIP/IPD* chapter. If this variable is not defined `cvmx-helper-check-defines.h` will define it to 0, and print a warning (see Note5). The file `cvmx-resources.config` will issue an error if the value is > 256. Default=0. | 0 bytes | depends on architecture |
| **CVMX_HELPER_PKO_MAX_PORTS_INTERFACE0* (See Note1)** | | |
| The number of hardware output ports to support on interface 0. A typical value is 4 for RGMII/SGMII. In particular, when implementing lockless PKO queues, limit the number of ports to those actually used to ensure enough queues are available. If this variable is not defined `cvmx-pko.h` defines it to 16, but then the variable is never used. The variable which is used in Simple Executive code is **CVMX_PKO_MAX_PORTS_INTERFACE0**, which is defined in `cvmx-resources.config` (if `CVMX_ENABLE_PKO_FUNCTIONS` is defined) to be the value of `CVMX_HELPER_PKO_MAX_PORTS_INTERFACE0`. Default=not defined. | not defined | depends on architecture |
| **CVMX_HELPER_PKO_MAX_PORTS_INTERFACE1* (See Note1)** | | |
| The number of hardware output ports to support on interface 1. A typical value is 4 for RGMII/SGMII. In particular, when implementing lockless PKO queues, limit the number of ports to those actually used to ensure enough queues are available. If this variable is not defined `cvmx-pko.h` defines it to 16, but then the variable is never used. The variable which is used in Simple Executive code is **CVMX_PKO_MAX_PORTS_INTERFACE1**, which is defined in `cvmx-resources.config` (if `CVMX_ENABLE_PKO_FUNCTIONS` is defined) to be the value of `CVMX_HELPER_PKO_MAX_PORTS_INTERFACE0`. Default=not defined. | not defined | depends on architecture |
| **CVMX_HELPER_PKO_QUEUES_PER_PORT_INTERFACE0* (See Note1)** | | |
| The number of queues per port for hardware interface 0. If `CVMX_ENABLE_PKO_FUNCTIONS` is defined, the file `cvmx-resources.config` will create the variable **CVMX_PKO_QUEUES_PER_PORT_INTERFACE0** (if `CVMX_ENABLE_PKO_FUNCTIONS` is defined) and set the value to the value of this variable. Default=1. | 1 queue per port | depends on architecture |
| **CVMX_HELPER_PKO_QUEUES_PER_PORT_INTERFACE1* (See Note1)** | | |
| The number of queues per port for hardware interface 1. If `CVMX_ENABLE_PKO_FUNCTIONS` is defined, the file `cvmx-resources.config` will create the variable **CVMX_PKO_QUEUES_PER_PORT_INTERFACE1** (if `CVMX_ENABLE_PKO_FUNCTIONS` is defined) and set the value to the value of this variable. Default=1. | 1 queue per port | depends on architecture |
| **CVMX_HELPER_SPI_TIMEOUT** | | |
| `CVMX_HELPER_SPI_TIMEOUT` is used to determine how many seconds the SPI initialization routines wait for SPI training before timing out. If this variable is not defined, then in `cvmx-helper-spi.c`, it is defined and set to 10 seconds. Default=10 seconds. | 10 seconds | 10 seconds |
| **CVMX_LLM_CONFIG_NUM_PORTS* (See Note1)** | | |
| CVMX_LLM_CONFIG_NUM_PORTS can be set to 1 or 2. This macro is used only for CN38XX and CN58XX, which have two independent LLM memory ports (interfaces) used with the DFA unit. This macro is used to tell software how many ports are present on the board. The file cvmx-resources.config will create the variable CVMX_LLM_NUM_PORTS, limiting the legal values to 1 or 2 without warning or error. Default=2. | 2 LLM ports | depends on architecture (used only for CN38XX and CN58XX) |

| Define (alphabetical order) | Default Value | Recommended Value |
|---|---|---|
| **NOTES** | | |
| Note1: * is used to mark defines which are used to create similarly-named defines. These similarly-named defines are shown in blue in the Notes column, and are referenced in the `.c` and `.h` files. | | |
| Note2: A test for the `CVMX_ENABLE_PKO_FUNCTIONS` define appears in many `.c` files. We recommend this value be defined for all applications doing packet I/O. | | |
| Note3: The CVMX_ENABLE_TIMER_FUNCTIONS define is a perfect example of how the resources defines are used. This macro creates the definitions for the timer pool. Without these definitions, the functions accessing the timer pool will not compile, so the presence of these functions is controlled by the define. In the case of CVMX_ENABLE_PKO_FUNCTIONS, the test for the define is in the many different .c and .h files which have I/O-related functions. | | |
| Note4: In `cvmx-spi4000.c`, if neither `CVMX_HELPER_DISABLE_SPI4000_BACKPRESSURE` nor `CVMX_HELPER_DISABLE_RGMII_BACKPRESSURE` are defined, then `CVMX_HELPER_DISABLE_RGMII_BACKPRESSURE` is defined to `0 (FALSE)`<br><br>If `CVMX_HELPER_DISABLE_SPI4000_BACKPRESSURE` is not defined, then `CVMX_HELPER_DISABLE_SPI4000_BACKPRESSURE` is defined to `CVMX_HELPER_DISABLE_RGMII_BACKPRESSURE` | | |
| Note5: `cvmx-helper-check-defines.h` is included by `cvmx-helper.c`. | | |

See Section 10 – "Configuration Output File (`cvmx_config.h`) Contents" for the result of running the `config` utility on this configuration file.

# 8 Configuring `cvmx-resources.config`

Edit a local copy of `cvmx-resources.config` to modify:

- Any of the five standard pools (Packet Data buffers, WQE buffers, PKO Command buffers, TIMER buffers, or DFA buffers)
- Default scratchpads
- Default FAU register variables

To add custom resources, use a `*-config.h` file in the local `config` directory. This will simplify comparing the `cvmx-resources.config` file against newer versions in newer SDK releases.

All FPA-managed buffers must be aligned on a 128-byte boundary (the cache line size), and must be a minimum of 1 cache line size (128 bytes). The Simple Executive APIs ensure these requirements are met.

The following information contains *requirements* for specific types of FPA-managed buffers. Recommendations for Packet Data buffers, PKO Command buffers, and Work Queue Entry buffers can be found in the *FPA* chapter.

Buffer Requirements:

- Packet Data Buffers:
  - *Size is Multiple of Cache-Line Size:* Buffer size is *required* to be a multiple of the cache line size (128 bytes) (see the *FPA* chapter for details.)
  - *Minimum Size:* PIP/IPD configuration rules impose minimum buffer size of 256 bytes (2 * cache line size). *Maximum Size:* PIP/IPD configuration rules impose a maximum buffer size of 16 Kbytes (128 * cache line size). The default configured size is 2048 bytes (16 * cache line size).
  - *Pool Number:* This pool must *always* be FPA Pool 0.
  - *Buffer Count:* This pool must never run out of buffers (see "buffer exhaustion and critical backpressure" in the *PIP/IPD* chapter).
- Work Queue Entry Buffers:
  - *Size is Multiple of Cache-Line Size:* Buffer size is *strongly recommended* to be a multiple of the cache line size (128 bytes).
  - *Minimum Size:* 128 bytes (1 * cache line size)
  - *Pool Number:* The pool number can be any unused pool number from [1-7]. The pool number by default is "1".
  - *Buffer Count:* This pool must never run out of buffers (see "buffer exhaustion and critical backpressure" in the *PIP/IPD* chapter).
- Packet Output (PKO) Command Buffers:
  - *Size is Multiple of Cache-Line Size:* Buffer size is *strongly recommended* to be a multiple of the cache line size (128 bytes).
  - *Minimum Size:* 8 * cache line size
  - *Maximum Size:* 511 * cache line size (`PKO_REG_CMD_BUF[SIZE]` is in units of 64-bit words (8 bytes), and the field is 13 bits wide, so the maximum number of 64-bit words is 8,191. (8,191 * 8 bytes/word) / (128 bytes/cache line) = 511 cache lines.)
  - *Pool Number:* The pool number can be any unused pool number from [1-7]. The pool number by default is "2".
  - *Buffer Count:* It is okay for this pool to run out of buffers. See the *FPA* chapter for more information.
- Other Buffers:
  - *Size is Multiple of Cache-Line Size:* Buffer size is *strongly recommended* to be a multiple of the cache line size (128 bytes). This size restriction provides support for "Don't Write Back" (DWB) functionality (see the *Advanced Topics* chapter for details).
  - *Minimum Size:* 128 bytes (1 * cache line size) (Required by FPA internals – see the *FPA* chapter for more information.)
  - *Pool Number:* The pool number can be any unused pool number from [1-7].

The size of the buffer pool is not limited by the hardware; it is only limited by the amount of physical memory available.

Note: Although more than one size of buffer can be in the same pool, this configuration is incredibly difficult to manage and creates an impossibly complex debugging environment. For example, if the IPD requests a Packet Data Buffer and receives a buffer which is smaller than the expected mbuf size (`IPD_PACKET_MBUFF_SIZE[MB_SIZE]`), when it writes the packet data

to the buffer, it may overwrite adjacent memory. See the *FPA* chapter for more information on this type of error.

## 8.1 Enabling Creation of Default Pools and Scratchpads

To enable the creation of the default pools, simply define the needed variable in the local `executive-config.h` file, as shown in the table below. Later, call the appropriate API function to allocate memory for and populate the pools.

**Table 6: How to Enable Creation of Commonly-Used Pools and Scratchpads**

| **Create Pools** | | |
|---|---|---|
| **Pool Created** | **Configuration Option** | **Default Pool Name** |
| Input Packet Pool | `CVMX_ENABLE_HELPER_FUNCTIONS` | `CVMX_FPA_PACKET_POOL` |
| Work Queue Entry Pool | `CVMX_ENABLE_HELPER_FUNCTIONS` | `CVMX_FPA_WQE_POOL` |
| PKO Command Buffer Pool | `CVMX_ENABLE_PKO_FUNCTIONS` | `CVMX_FPA_OUTPUT_BUFFER_POOL` |
| DFA Pool | `CVMX_ENABLE_DFA_FUNCTIONS` | `CVMX_FPA_DFA_POOL` |
| Timer Pool | `CVMX_ENABLE_TIMER_FUNCTIONS` | `CVMX_FPA_TIMER_POOL` |
| ZIP Pool | No config option to create pool automatically. | `N/A` |
| **Create Scratchpads** | | |
| **Scratchpad Created** | **Configuration Option** | **Default Scratchpad Name** |
| Scratchpad for IOBDMA operations such as `cvmx_fpa_async_alloc()`. | `CVMX_ENABLE_HELPER_FUNCTIONS` | `CVMX_SCR_SCRATCH` |
| **Notes** | | |
| *Note: If `CVMX_ENABLE_HELPER_FUNCTIONS` is defined, then `CVMX_ENABLE_PKO_FUNCTIONS` will be automatically defined in `executive-config.h`* | | |

## 8.2 Configuring Buffers and Buffer Pool Definitions

The cvmx-resources.config file is in a format understood by the configuration utility. In the example below, the Packet Data buffer pool (`CVMX_FPA_PACKET_POOL`) is configured.

```
#ifdef CAVIUM_COMPONENT_REQUIREMENT
cvmxconfig
{
      fpa CVMX_FPA_PACKET_POOL
            pool        = 0
            size        = 16
            priority    = 1
            protected   = 1
            description = "Packet buffers";
}
#endif // end CAVIUM_COMPONENT_REQUIREMENT
```

**Table 7: Configuration Keywords for Buffer Pools**

| Configuration Keywords | Description |
|---|---|
| `pool = <integer>` | Pool number (0-7). Pool 0 is reserved for CVMX_FPA_PACKET_POOL. If no number is specified, the configuration system will select a pool number based on priority (this strongly recommended). (Range = 0-7.) |
| `size = <integer>` | Specifies the size of the pool's buffers in number of cache lines (128 bytes each). (Range = 1-512.) |
| `priority = <integer>` | The priority ranges from 1 through 8, specifies the desired priority. (If a pool number is not specified in the data structure, the system will assign a number: the highest priority gets the lowest available pool number.) Note that "priority" is only used to assign the pool numbers: there is no performance advantage to lower-numbered pools. (1 = highest priority; if no priority is specified, the default value = 8) (Range = 1-8.) |
| `protected = <boolean>` | Recommended=1 (TRUE) (Legal values are: true\|false, on\|off, 1\|0.) |
| `description = <string>` | A doubly quoted string, which will appear as comment in the generated `cvmx-config.h`, and the pool name in the `pool_info` data structure. |

The `priority` keyword is only used to assign a lower number to higher-priority pools: there is no performance advantage to lower-numbered pools.

The `protected` keyword is only used by the configuration utility. This keyword does not provide any runtime memory protection.

We recommend that the pool number should be unspecified, allowing the system to select the pool number. There is no inconvenience to software since pools are referenced by their symbolic names, not their pool number.

The WQE and PKO Command buffer pools definition shows that not all configuration keywords are required:

```
#ifdef CAVIUM_COMPONENT_REQUIREMENT
cvmxconfig
{
      fpa CVMX_FPA_OUTPUT_BUFFER_POOL
            size        = 8 // 128-byte cache lines
            protected   = 1
            description = "PKO queue command buffers";

      fpa CVMX_FPA_WQE_POOL
            size        = 1 // 128-byte cache lines
            priority    = 1
```

```
            protected   = 1
            description = "Work queue entries";
}
#endif // end CAVIUM_COMPONENT_REQUIREMENT
```

Note that because `CVMX_FPA_WQE_POOL` has priority 1, and `CVMX_FPA_OUTPUT_BUFFER_POOL` has priority 8 (default), in the resultant `cvmx-config.h` file, `CVMX_FPA_WQE_POOL` is assigned a lower pool number than `CVMX_FPA_OUTPUT_BUFFER_POOL` (2 instead of 3).

See Section 10 – "Configuration Output File (`cvmx_config.h`) Contents" for the result of running the `config` utility on this configuration file.

### 8.2.1 Unprotected pools

We recommend that protected *always* be set to 1 (TRUE). The "unprotected pools" feature should not be used because it can result in difficult to debug problems. See Section 12 – " "Unprotected" Buffer Pools".

## *8.3 Configuring Scratchpad Areas*

The `cvmx_fpa_async_alloc()` function uses both the buffer pools and a scratchpad area. The scratchpad area is defined using configuration keywords which are similar to those used for the buffer pools.

Note the configuration keyword "`iobdma`" which specifies the scratchpad area will be a target for an IOBDMA operation. This keyword tells the configuration utility to assign IOBDMA scratchpad areas the lowest addresses. Note that the scratchpad area must be 64-bit aligned, so the low 3 bits of the scratchpad address are zeroes. If all scratchpad areas are configured to be 8 bytes, a total of [(16 cache lines) * (128 bytes per cache line / 8 bytes per scratchpad area)] = 256 scratchpad areas are available. Usually only one 8-byte scratchpad area is used (`CVMX_SCR_SCRATCH`), so this restriction is not a problem. (The `CvmMemCtl` register field `IOBDMASCRMSB` can be used to allow IOBDMAs to access scratchpad areas above the first 16 cache lines. See the *Advanced Topics* chapter and the *HRM* for details.)

Note: The scratchpad address provided to the IOBDMA instruction must be 64-bit aligned, and IOBDMA operations always return a 64-bit result. Units (such as the FAU) that support 8-bit, 16-bit, and 32-bit operations store the result in bits <7:0> (8-bit), <15:0> (16-bit), or <31:0> (32bit) of the 64-bit register.

> ***Warning: If the user configures more scratchpad areas than the number of cache lines allocated for scratchpad, no error will occur, and the Dcache will become corrupted as accesses to those scratchpad locations access Dcache.***

```
#ifdef CAVIUM_COMPONENT_REQUIREMENT
cvmxconfig
{
      scratch CVMX_SCR_SCRATCH
            size        = 8  // number of bytes per element in bytes
            count       = 1  // number of elements (default = 1)
            iobdma      = true
            permanent   = false
            description = "Generic scratch iobdma area";
}
#endif  // end CAVIUM_COMPONENT_REQUIREMENT
```

Note that the above definition sets permanent to false. This is an exception. We generally recommend this value be set to true. See Section 8.3.1 – "Permanent Scratchpads".

**Table 8: Configuration Keywords for Scratchpad Areas**

| Configuration Keywords | Default | Description |
|---|---|---|
| `size = <integer>` | 8 bytes | Specifies the number of bytes per element. The default is 64 bits (8 bytes). (Legal values = 1, 2, 4, 8. If scratchpad area will be used for IOBDMA, the only legal value is 8.) |
| `count = <integer>` | 1 | Specifies the number of elements. (If `count` is not specified, number of elements =1.) (Legal values = number, no range checking.) |
| `iobdma = <boolean>` | 1 | If true, this scratch location can be used as an IOBDMA destination. This keyword tells the configuration tool to select a low address for the scratchpad area. (Legal values are true\|false, on\|off, 1\|0.) |
| `permanent = <boolean>` | 1 | Used to refer to the same scratchpad location by different names. If false, defines for this scratchpad can be shared with another scratchpad (recommended = true). The config utility will only create shared scratchpads if the sizes for the scratchpads are identical. (Legal values are true\|false, on\|off, 1\|0.) |
| `description = <string>` | none | A doubly quoted string, which will appear as comment in the generated `cvmx-config.h`. |

For example, the following entry is present in the default `executive-config.h` file (found in the `examples/traffic-gen` directory). (Note that the ideal place for this entry would have been in a custom `*=config.h` file.) (The `CAVIUM_COMPONENT_REQUIREMENT` define keeps the C preprocessor from trying to interpret the `cvmxconfig` section. The `cvmxconfig` section directs the configuration utility to process the contents of the section.)

```
#ifdef CAVIUM_COMPONENT_REQUIREMENT
cvmxconfig
{
    scratch TRAFFICGEN_SCR_WORK
        size        = 8 // number of bytes per element in bytes
        count       = 1  // number of elements (default = 1)
        iobdma      = true
        permanent   = true
        description = "Async get work";
}
#endif // end CAVIUM_COMPONENT_REQUIREMENT
```

### 8.3.1 Permanent Scratchpads

We recommend that permanent *always* be set to true. If permanent == false, and there is more than one scratchpad area is defined with the same size, two different names will reference the same scratchpad area. If this is not planned, then difficult-to-debug errors may occur (one part of the application over-writes the scratchpad area in use by another part of the application).

See Section 12 - "Unprotected" Buffer Pools" for a description of the matching problem for buffer pool definitions.

If two parts of the application *must* share the same scratchpad area, but use two *different* names for it, then use a simple #define instead of this feature. This is easier to debug.

(The term "permanent" was chosen to indicate to the user that the scratchpad area has a specific, long-term purpose. In contrast, a function can use a scratchpad area temporarily by initiating an IOBDMA request using the scratchpad area address, and retrieving the reply within the same function. Other functions which also need temporary scratchpad can use the same scratchpad area. The "permanent" designation is an indication to the user that the scratchpad area is dedicated for one use, and is not a temporary workspace. Other than not creating multiple names to refer to the same scratchpad, this keyword does not make any functional difference.)

## *8.4 Configuring Fetch and Add (FAU) Register Resources*

The DFA unit configuration uses Fetch and Add Registers. The example below shows how to instruct the configuration utility to configure in FAU registers.

```
#ifdef CAVIUM_COMPONENT_REQUIREMENT
cvmxconfig
{
    fau CVMX_FAU_DFA_STATE
        size        = 8  // size is in bytes
        count       = 1
        description = "FAU registers for the state of the DFA command queue";
}
#endif // end CAVIUM_COMPONENT_REQUIREMENT
```

These resources are configured using the configuration keywords shown in the following table:

**Table 9: Configuration Keywords for FAU Registers**

| Configuration Keywords | Default | Description |
|---|---|---|
| `size = <integer>` | 8 bytes | Specifies the number of bytes per element. The default is 64 bits (8 bytes). (Legal values are 1, 2, 4, or 8.) |
| `count = <integer>` | 1 | Specifies the number of elements. (If `count` is not specified, number of elements =1.) (Legal values are numbers, no range checking.) |
| `description = <string>` | none | A doubly quoted string, which will appear as comment in the generated `cvmx-config.h`. |

# 9 Adding a Custom `*-config.h` File

To add custom resources, use a `*-config.h` file in the local `config` directory. This will simplify comparing the `cvmx-resources.config` file against newer versions in newer SDK releases. For pools, scratchpads, and FAU registers, follow the same configuration keyword requirements shown in Section 8 – "Configuring `cvmx-resources.config`".

The `fpa_simplified` example contains a custom config file, `my-config.h`, with the contents:

```
#ifdef CAVIUM_COMPONENT_REQUIREMENT
cvmxconfig
{
    fpa CVMX_MY_FIRST_POOL
        size        = 2 // 128-byte cache lines
        protected   = 1
        description = "MY FIRST CUSTOM POOL";
    fpa CVMX_MY_SECOND_POOL
        size        = 2 // 128-byte cache lines
        protected   = 1
        description = "MY SECOND CUSTOM POOL";
    fpa CVMX_MY_THIRD_POOL
        size        = 2 // 128-byte cache lines
        protected   = 1
        description = "MY THIRD CUSTOM POOL";
}
#endif // end CAVIUM_COMPONENT_REQUIREMENT
```

After calling `make` to configure and build the example, the `cvmx-config.h` file contains the new definitions:

```
/************************* FPA allocation *********************************/
/* Pool sizes in bytes, must be multiple of a cache line */
#define CVMX_FPA_POOL_0_SIZE (16 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_1_SIZE (1 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_2_SIZE (8 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_3_SIZE (2 * CVMX_CACHE_LINE_SIZE)
```

```
#define CVMX_FPA_POOL_4_SIZE (2 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_5_SIZE (2 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_6_SIZE (0 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_7_SIZE (0 * CVMX_CACHE_LINE_SIZE)

/* Pools in use */
#define CVMX_FPA_PACKET_POOL              (0) /**< Packet buffers */
#define CVMX_FPA_PACKET_POOL_SIZE         CVMX_FPA_POOL_0_SIZE
#define CVMX_FPA_WQE_POOL                 (1) /**< Work queue entrys */
#define CVMX_FPA_WQE_POOL_SIZE            CVMX_FPA_POOL_1_SIZE
#define CVMX_FPA_OUTPUT_BUFFER_POOL       (2) /**< PKO queue command buffers */
#define CVMX_FPA_OUTPUT_BUFFER_POOL_SIZE  CVMX_FPA_POOL_2_SIZE
#define CVMX_MY_FIRST_POOL                (3) /**< MY FIRST CUSTOM POOL */
#define CVMX_MY_FIRST_POOL_SIZE           CVMX_FPA_POOL_3_SIZE
#define CVMX_MY_SECOND_POOL               (4) /**< MY SECOND CUSTOM POOL */
#define CVMX_MY_SECOND_POOL_SIZE          CVMX_FPA_POOL_4_SIZE
#define CVMX_MY_THIRD_POOL                (5) /**< MY THIRD CUSTOM POOL */
#define CVMX_MY_THIRD_POOL_SIZE             CVMX_FPA_POOL_5_SIZE
```

Note that the pool numbers (3, 4, and 5) were chosen automatically. Allowing the configuration utility to choose the pool number the best choice to set up the pools, except for pool 0: there is less chance of an error in numbering the pools.

## 9.1 Example of Using the `define` Keyword

Given the input:

```
#ifdef CAVIUM_COMPONENT_REQUIREMENT
cvmxconfig
{
        // the following item (description) is used only for
        // documentation:  there is no corresponding output in the
        // cvmx-config.h file.
        description "This is an example configuration file"

        #if CVMX_LLM_CONFIG_NUM_PORTS == 2
                define VARIABLE_1 value = 2;
        #else
                define VARIABLE_1 value = 1;
        #endif  // end CVMX_LLM_CONFIG_NUM_PORTS

        define VARIABLE_2 // a comment can be on the same line
        description="define the variable VARIABLE_2"
        value=9;

        define VARIABLE_2_STRING
        // Note that the description keyword is optional
        value="this will not work"; // note that this macro won't
                                    // work: the resultant value is
                                    // not quoted, so it is not a
                                    // string

        // a comment can also be on a separate line
}
#endif // CAVIUM_COMPONENT_REQUIREMENTS
```

The output in `cvmx-config.h` is:

```
#define FRED 1
#define VARIABLE_2 9     /**< define the variable VARIABLE_2*/
#define VARIABLE_2_STRING this will not work
```

# 10 Configuration Output File (`cvmx_config.h`) Contents

To run the configuration utility and build the application, execute the `make clean` command in the example directory, then the `make` command. See the *Software Development Kit (SDK) Tutorial* chapter for more information.

After `make` has run, the file `cvmx_config.h` will have been created The exact contents will depend on your configuration. The pool numbers are set in the order the pools were configured into the Simple Executive, with the custom pools at the end. In this example, no FAU registers were created.

```
/************************ FPA allocation *********************************/
/* Pool sizes in bytes, must be multiple of a cache line */
#define CVMX_FPA_POOL_0_SIZE (16 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_1_SIZE (1 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_2_SIZE (8 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_3_SIZE (0 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_4_SIZE (0 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_5_SIZE (0 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_6_SIZE (0 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_7_SIZE (0 * CVMX_CACHE_LINE_SIZE)

/* Pools in use */
#define CVMX_FPA_PACKET_POOL          (0)  /* Packet Data buffers        */
#define CVMX_FPA_PACKET_POOL_SIZE     CVMX_FPA_POOL_0_SIZE
#define CVMX_FPA_WQE_POOL             (1)  /* Work queue entries         */
#define CVMX_FPA_WQE_POOL_SIZE        CVMX_FPA_POOL_1_SIZE
#define CVMX_FPA_OUTPUT_BUFFER_POOL   (2)  /* PKO queue command buffers  */
#define CVMX_FPA_OUTPUT_BUFFER_POOL_SIZE  CVMX_FPA_POOL_2_SIZE

#define CVMX_SCR_SCRATCH              (8) /* Generic scratch iobdma area */
```

# 11 Adding Custom Configuration While Using the SDK API

Two functions pointers are provided with SDK 1.9+ which help the user easily change the default values used by Simple Executive in configuring selected hardware units:

- PIP/IPD: `cvmx_override_ipd_port_setup`
- PKO: `cvmx_override_pko_queue_priority`

If the user has set this pointer to a function before calling any `cvmx-helper*` functions, then the `cvmx-helper*` functions will call the override function at the appropriate time.

**Table 10: Configuration Override Functions for PIP/IPD and PKO**

| Configuration Override Functions |
|---|
| `CVMX_SHARED void (*cvmx_override_ipd_port_setup)(int ipd_port)` |
| This function pointer is initialized to NULL:<br>`CVMX_SHARED void (*cvmx_override_ipd_port_setup)(int ipd_port) = NULL;`<br>The user may initialize the function pointer to the address of a user-defined function.<br>This function is called by `cvmx_helper_initialize_packet_io_global()`. The calling program will provide the ipd port. This function is called after the default PIP/IPD configuration, but before IPD is enabled. Typically, the function will modify CSR values by reading the prior CSR value into the CSR data structure defined in the SDK, changing fields in the data structure as needed, then writing the new value.<br>`ipd_port`: which port to configure<br>Returns void. |
| `CVMX_SHARED void (*cvmx_override_pko_queue_priority)(int pko_port, uint64_t priorities[16])` |
| This function pointer is initialized to NULL:<br>`CVMX_SHARED void (*cvmx_override_pko_queue_priority)(int pko_port, uint64_t priorities[16]) = NULL;`<br>The user may initialize the function pointer to the address of a user-defined function.<br>This function is called by `cvmx_helper_initialize_packet_io_global()`, and is used to allow customization of the PKO queue priorities based on the PKO port number.<br>`pko_port`: which port to configure<br>`priorities[16]`: Each packet output queue has an associated priority. The higher the priority, the more often it can send a packet.<br>Returns void. |

For Example:

The default PKO queues for the port configuration (as of SDK 2.0) is:

```
uint64_t priorities[16] = {8,7,6,5,4,3,2,1,8,7,6,5,4,3,2,1};

The following custom function can be used to override the PKO port priorities.

// this will modify the contents of the priorities array, the calling
// function can then use the modified values to configure the PKO
// The calling function is responsible for writing the appropriate PKO
// registers.
void custom_PKO(int pko_port, uint64_t priorities[16])
{
    int j; // loop control variable
    // The three arrays below show different examples of how the up-to-16 PKO
    // queues per port can be configured for different QoS scheduling.

    // prio_strict is using strict priorities (9) for all 4 queues
    uint64_t prio_strict[16] = {9,9,9,9,0,0,0,0,0,0,0,0,0,0,0,0};

    // prio_wrr (weighted round robin) uses one strict priority (9) for the
    // first queue, and weighted round robin for the other 3 queues, with
    // relative weights (8, 4, and 1)
```

```
    uint64_t prio_wrr[16] = {9,8,4,1,0,0,0,0,0,0,0,0,0,0,0,0};

    // prio_default is a set of default priorities
    int prio_default[16] = {8,7,6,5,4,3,2,1,8,7,6,5,4,3,2,1};

    switch (pko_port)
    {
        case 0:
        {
            cvmx_dprintf("configuring port 0 for strict priority\n");
            for (j = 0; j < 16; j++)
            {
               priorities[j] = prio_strict[j];
            }
            break;
        } // end case 0
        case 1:
        {
            cvmx_dprintf("configuring port 1 for SP0,WRR1,WRR2,WRR3:\n");
            for (j = 0; j < 16; j++)
            {
                priorities[j] = prio_wrr[j];
            }
            break;
        } // end case 1
        default:
        {
            cvmx_dprintf("configuring port %d for default priorities:\n",
                         pko_port);
            for (j = 0; j < 16; j  ++)
            {
                priorities[j] = prio_default[j];
            }
            break;
        } // end default case
     } // end switch on pko_port
} // end custom_PKO()

CVMX_SHARED void (*cvmx_override_pko_queue_priority)(int pko_port, uint64_t
priorities[16]) = custom_PKO;
```

This function is then called from a helper function:

```
while (num_ports--)
    {
        /* Give the user a chance to override the per queue priorities */
        if (cvmx_override_pko_queue_priority)
            cvmx_override_pko_queue_priority(ipd_port, priorities);

        // the following function writes the priorities to the PKO register
        cvmx_pko_config_port(ipd_port,
                             cvmx_pko_get_base_queue_per_core(ipd_port, 0),
                             cvmx_pko_get_num_queues(ipd_port), priorities);
        ipd_port++;
    }
```

# 12 "Unprotected" Buffer Pools

Readers who simply follow the buffer pool configuration instructions in Section 8.2 – "Configuring Buffers and Buffer Pool Definitions" do not need to read this section. This section is provided to explain the dangers of using unprotected pools, which can be configured via the Simple Executive configuration.

> ***Note that the keyword `protected` does not mean that there is any runtime protection for the memory: it is only used in the limited way described in this section.***

We recommend that the keyword `protected` always be set to 1 (TRUE). The "unprotected pools" feature should not be used. Information about unprotected pools is provided in this section to illustrate the type of difficult-to-debug problems which can occur if this configuration keyword is not set appropriately.

If the "protected" field is set to "false" (0), AND two or more pools have the *same* buffer size, then multiple macros can refer to the same pool. This is sometimes used if there are more than 8 different uses for FPA pools. In the example below, both `CVMX_MY_POOL` and `CVMX_MY_THIRD_POOL` will refer to pool number three.

For example, given `my-config.h with` the following contents:

```
cvmxconfig
        {
                fpa CVMX_MY_POOL      // shared with CVMX_MY_THIRD_POOL
                    size        = 2   // must be identical with shared pool
                    protected   = 0   // not protected
                    description = "MY CUSTOM POOL";
                fpa CVMX_MY_SECOND_POOL
                    size        = 4
                    protected   = 1   // protected
                    description = "MY SECOND CUSTOM POOL";
                fpa CVMX_MY_THIRD_POOL   // shared with CVMX_MY_POOL
                    size        = 2      // must be identical with shared pool
                    protected   = 0      // not protected
                    description = "MY THIRD CUSTOM POOL";
        }
```

The `cvmx-config.h` file will contain:

```
/************************* FPA allocation *********************************/
/* Pool sizes in bytes, must be multiple of a cache line */
#define CVMX_FPA_POOL_0_SIZE (16 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_1_SIZE (1 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_2_SIZE (8 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_3_SIZE (2 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_4_SIZE (4 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_5_SIZE (0 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_6_SIZE (0 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_7_SIZE (0 * CVMX_CACHE_LINE_SIZE)

/* Pools in use */
#define CVMX_FPA_PACKET_POOL                (0) /**< Packet buffers*/
```

```
#define CVMX_FPA_PACKET_POOL_SIZE           CVMX_FPA_POOL_0_SIZE
#define CVMX_FPA_WQE_POOL                   (1) /**< Work queue entrys */
#define CVMX_FPA_WQE_POOL_SIZE              CVMX_FPA_POOL_1_SIZE
#define CVMX_FPA_OUTPUT_BUFFER_POOL         (2) /**< PKO queue command
buffers*/
#define CVMX_FPA_OUTPUT_BUFFER_POOL_SIZE    CVMX_FPA_POOL_2_SIZE
#define CVMX_MY_POOL                        (3) /**< MY CUSTOM POOL*/
#define CVMX_MY_POOL_SIZE                   CVMX_FPA_POOL_3_SIZE
#define CVMX_MY_THIRD_POOL                  (3) /**< MY THIRD CUSTOM POOL */
#define CVMX_MY_THIRD_POOL_SIZE             CVMX_FPA_POOL_3_SIZE
#define CVMX_MY_SECOND_POOL                 (4) /**< MY SECOND CUSTOM POOL */
#define CVMX_MY_SECOND_POOL_SIZE            CVMX_FPA_POOL_4_SIZE
```

***Note: If protected = false, and there more than one pool is defined with the same buffer size, these pools will be shared. If this is not planned, then difficult-to-debug errors may occur.***

***Another error can occur if it is intended that the pools be shared, but different buffer sizes were specified during configuration. An example of this type of error is shown in the next section.***

**Example of Configuration Error:**

Note: The pool will not be shared unless the buffer sizes are identical. If they are not identical, the config utility (`host/bin/cvmx-config`) will quietly create different #defines for the pools: they will not be shared.

```
cvmxconfig
        {
                fpa CVMX_MY_POOL        // shared with CVMX_MY_THIRD_POOL
                    size        = 6     // ERROR: size does not match!
                    protected   = 0     // not protected
                    description = "MY CUSTOM POOL";
                fpa CVMX_MY_SECOND_POOL
                    size        = 4
                    protected   = 1     // protected
                    description = "MY SECOND CUSTOM POOL";
                fpa CVMX_MY_THIRD_POOL  // shared with CVMX_MY_POOL
                    size        = 2     // ERROR: size does not match!
                    protected   = 0     // not protected
                    description = "MY THIRD CUSTOM POOL";
        }
```

The `cvmx-config.h` file will contain:

```
/************************ FPA allocation *********************************/
/* Pool sizes in bytes, must be multiple of a cache line */
#define CVMX_FPA_POOL_0_SIZE (16 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_1_SIZE (1 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_2_SIZE (8 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_3_SIZE (6 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_4_SIZE (4 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_5_SIZE (2 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_6_SIZE (0 * CVMX_CACHE_LINE_SIZE)
#define CVMX_FPA_POOL_7_SIZE (0 * CVMX_CACHE_LINE_SIZE)
```

```
/* Pools in use */
#define CVMX_FPA_PACKET_POOL                (0) /**< Packet buffers*/
#define CVMX_FPA_PACKET_POOL_SIZE           CVMX_FPA_POOL_0_SIZE
#define CVMX_FPA_WQE_POOL                   (1) /**< Work queue entrys */
#define CVMX_FPA_WQE_POOL_SIZE              CVMX_FPA_POOL_1_SIZE
#define CVMX_FPA_OUTPUT_BUFFER_POOL         (2) /**< PKO queue command
buffers*/
#define CVMX_FPA_OUTPUT_BUFFER_POOL_SIZE    CVMX_FPA_POOL_2_SIZE
#define CVMX_MY_POOL                        (3) /**< MY CUSTOM POOL*/
#define CVMX_MY_POOL_SIZE                   CVMX_FPA_POOL_3_SIZE
#define CVMX_MY_SECOND_POOL                 (4) /**< MY SECOND CUSTOM POOL */
#define CVMX_MY_SECOND_POOL_SIZE            CVMX_FPA_POOL_4_SIZE
#define CVMX_MY_THIRD_POOL                  (5) /**< MY THIRD CUSTOM POOL */
#define CVMX_MY_THIRD_POOL_SIZE             CVMX_FPA_POOL_5_SIZE
```

If it is absolutely needed to refer to the same pool by different names, for ease in debugging it is better to simply use a simple #define instead of configured-in sharing:

```
#define NAME_1   NAME2
```