OpenCPI
On Chip Control Plane (OCCP)
Memory Guide

OpenCPI Release: v2.4.8

Revision History

Revision	Description of Change	Date
1.0	Initial content	2024-06-24
1.1	Remove reference to deprecated platform	2025-02-20
1.2		
1.3		

Table of Contents

1 Introduction....4

2 References....5

3 OCCP Memory....6

3.1 Memory Allocation....7

3.2 Memory Addresses....8

3.3 Configurations....9

3.4 Configurations Synchronization....10

4 Maximum Number of Workers....11

4.1 63 Worker Maximum....12

4.2 127 Worker Maximum....13

4.3 255 Worker Maximum....14

4.4 511 Worker Maximum....15

1 Introduction

The OpenCPI On Chip Control Plane (OCCP) memory allocation handles memory in a specific way when deploying on a platform. This document is written for developers explaining how memory is allocated, the memory addressing is done, how configurations can be altered, and the limits imposed when executing an OpenCPI application that uses a hardware description language (HDL) assembly.

2 References

The document currently has no references.

Table 1: Table of Reference Documents

Title	Link

3 OCCP Memory

This section describes the OCCP memory allocation and addressing space. The OpenCPI kernel driver allocates Direct Memory Access (DMA) memory from the platform for the OpenCPI framework to utilize; this allocation is done in a static location. This block of memory can then be mapped for use within the framework code.

3.1 Memory Allocation

When an assembly is loaded the runtime code maps the OCCP memory space and assigns accessors within the runtime code; this allocation is done in a dynamic location. The runtime code then takes the platform worker out of reset allowing the core primitives to start accessing and manipulating the memory. The figure below, illustrates how much, and in what order the memory is allocated for each section in the current implementation. The allocation starts at the top right and moves left and down the figure. The illustration is not to scale.

<table>
<tr><td>MSB</td><td>< - - - -</td><td>LSB</td></tr>
<tr><td colspan="2">Admin Pad
(144 B)</td><td>Admin
(112 B)</td></tr>
<tr><td colspan="2">Worker 0 Pad
(192 B)</td><td>Worker 0 Control
(64 B)</td></tr>
<tr><td colspan="2">...</td><td>...</td></tr>
<tr><td colspan="2">Worker 126 Pad
(192 B)</td><td>Worker 126 Control
(64 B)</td></tr>
<tr><td colspan="3">Worker Configuration Pad
(491520 B)</td></tr>
<tr><td colspan="3">Worker 0 Configuration
(512 KB)</td></tr>
<tr><td colspan="3">...</td></tr>
<tr><td colspan="3">Worker 126 Configuration
(512 KB)</td></tr>
</table>

Starting at the top right of the figure, the admin registers are 112 Bytes followed by a pad of 144 Bytes. Next, the control space is a fixed 64 Bytes followed by a 192 Byte pad for each worker. Lastly, configuration space is 512 Kilobytes for each worker. There is no padding between each workers configuration space. This memory allocation configuration supports a maximum of 127 workers. Memory is allocated for all 127 workers even if the HDL assembly does not contain the maximum number of workers. If the figure was expanded there would be 127 duplicated rows of worker control space for worker 0 to worker 126 as well as 127 duplicated rows of worker configuration space for worker 0 to worker 126.

3.2 Memory Addresses

Although, the runtime code allocates the memory on the platform, the memory has to also be accessed by the core primitive on the hardware side. Access is accomplished using the Advanced eXtensible Interface (AXI) on the FPGA. Within the core primitive logic, the memory is accessed by memory addresses within the ocscp_rv module. Worker memory is addressed using 24 bits of the AXI 32 bit memory address, however there are two bits appended as LSBs making the total size really 26 bits even though only the 24 MSB are communicated with the module. These two appended bits are believed to be referred to as “read bits” but there was some inconsistency in that nomenclature based on comments in the code.

The OCCP memory space and addressing has to match in both the runtime (C++) code and core primitives (VHDL) code and therefore the configurations must align. The ocscp_rv core primitive module, parses the incoming address to determine if it is an admin address, control address, or configuration address. These are checked in order as that is the order the memory spaces are allocated. In the primitive core code, all of the bit lengths are without the read bits therefore there are 24 bits total.

The 6 LSB represent the first 256 addresses each holding 1 byte of data. By this logic, anything with the 18 MSB as all zeros is consider an admin register address. Similarly, a control address would be when the 7 MSBs are all zero, and the 17 LSBs would hold the address for the workers control space; note a significant portion of this is padding as shown by the memory map. Any address with a nonzero 7 MSBs would be considered part of the config space. These boundaries of 6 and 17 LSBs, shown in the figure below, are actually measured from the LSB excluding the read bits as previously mentioned.

<table>
<tr><td colspan="3">Address Bits</td><td>Read Bits</td></tr>
<tr><td>7 Bits</td><td>11 Bits</td><td>6 Bits</td><td>2 Bits</td></tr>
<tr><td></td><td></td><td>Admin Bits</td><td></td></tr>
<tr><td></td><td colspan="2">Control Bits</td><td></td></tr>
<tr><td colspan="3">Configuration Bits</td><td></td></tr>
</table>

3.3 Configurations

The memory allocation configurations live in two places, the runtime code and the core primitives code. The runtime code, written in C++, are configured in **opencpi/runtime/ hdl/include/ HdlOCCP.hh**, while the core primitives code, written in VHDL, are configured in **opencpi/projects/core/hdl/primitives/platform/platform_pkg.vhd**.

- Runtime configurations:
  opencpi/runtime/ hdl/include/ HdlOCCP.hh
  - **OCCP_ADMIN_CONFIG_SIZE**
    - Determines the size of the pad after the **OccpAdminRegisters**
  - **OCCP_WORKER_CONFIG_WINDOW_BITS**
    - The number of config bit which controls how much spaces to allocate for the worker configs
  - **OCCP_WORKER_CONTROL_SIZE**
    - Amount of padding after each workers **OccpWorkerRegisters**
    - More importantly this correlates with the number of control bits and determines how much memory is allocated for worker control space.
  - **OCCP_MAX_WORKERS**
    - The number of workers memory will be allocated for
- Core primitive configurations:
  opencpi/projects/core/hdl/primitives/platform/platform_pkg.vhd
  - **worker_control_bits**
    - Number of control bits
  - **worker_config_bits**
    - Number of config bits
  - **worker_max_nworkers**
    - Max number of workers

OCCP MAX WORKERS	OCCP WORKER CONTROL SIZE	OCCP WORKER CONFIG WINDOW BITS	OCCP ADMIN CONFIG SIZE	Worker max nworkers	Worker config bits	Worker control bits
63	0x100	20	256	63	18	6
127	0x100	19	256	127	17	6

3.4 Configurations Synchronization

To synchronize the runtime and core primitive the following requirements must be met. The left side of the equation is the runtime code parameters, and the right side of the equation is the core primitive parameters.

- **OCCP_MAX_WORKERS** = **worker_max_nworkers**
- **OCCP_WORKER_CONTROL_SIZE** = 2^(**worker_control_bits** + 2)
  - **OCCP_WORKER_CONTROL_SIZE** is in hex
- **OCCP_WORKER_CONFIG_WINDOW_BITS** = **worker_config_bits** + 2

The plus two on the core primitive side are the two least significant bit which are appended to the memory address, these are believed to be called readsize bits in the code comments.

4 Maximum Number of Workers

It appears that many years ago the worker limit was 31, but was expanded to 63 and remained at 63 for an extended period of time. Currently, the maximum number of workers has been expanded to 127. However, the maximum worker limit could be expanded further by decrease the size of each workers configuration space by a greater amount. An attempt was made to get to a 256 and 511 worker limits with the current paradigm, however several problems arose.

4.1 63 Worker Maximum

A maximum of 63 workers was the long standing limit for OpenCPI, this configuration can still be duplicated by altering the **HdlOCCP.hh** and **platform_pkg.vhd** files. This will increase the configuration space for each worker to 1MB allowing for larger configuration workers to be utilized but decreasing the overall number of workers allowed.

4.2 127 Worker Maximum

A maximum of 127 workers is the current limit after the changes made to get here there were several alterations that needed to be made.

- The attention and present admin registers had to be expanded from 64 bits wide to 128 bits. This is done in both the runtime code in **HdlOCCP.hh** and primitive code **ocscp_rv.vhd**.
- Change m_usedImpls bitmask type from fixed 64 bit to vector of Boolean values such that the bitmask can be dynamically expanded to the worker limit as needed. This change was done in the application runtime code in **Application.cc**.
- Fix assignments within core primitive to dynamically assign address and window lengths based on configured bit sizes in **wci_master.vhd**.
- Update the build engine in **hdl-make.mk**, to be able to build assemblies with larger numbers of workers due to a limitation in the length of bash commands. This was solved by echoing the commands into a shell script and running the script. This has increased the number of workers capable of being built into an assembly. However, the echo command is still a bash command and with enough workers this limitation will again be an issue, this can be solved by using the gnu make FILE function to create and insert the list of workers into the file. After the FILE function is used, SED has to be used to remove unwanted characters to make the shell script functional.
- Cleanup and removal of legacy regions and UUID from the OCCP admin registers in **HdlOCCP.hh**. The **HdlUUID** struct used to be embedded into the **OccpAdminRegisters** struct, however in testing these values were always zero in the admin registers. Therefore, it has since been removed, which also enabled the removal of the **ocfrp_check_main.cc** tool which seemed to just pull the HdlUUID information from the admin space. The **HdlUUID** struct is still used else where in the runtime code to store the UUID information, it is just no longer used within the admin registers.
- To get over 255 workers the runtime code in **BasePValue.cc** indexing of workers has to change types from uchar to ushort to give a larger range of numbers. Although, this change was not needed for the current worker count it a good change that seemed low risk.
- Addition of buffer between worker control space and worker configuration space in **HdlOCCP.hh** to ensure the addressing schema aligns to the assumed boundaries in the core primitive.

4.3 255 Worker Maximum

A maximum of 255 workers was investigated and functional based on limited testing with the following changes.

- **OCCP_WORKER_CONFIG_WINDOW_BITS** set to 18 and **OCCP_MAX_WORKERS** set to 255 in **HdlOCCP.hh** for the core primitive code configuration.
- **worker_config_bits** set to 16 and **worker_max_nworkers** set to 255 in **platform_pkg.vhd** for the runtime code configuration.
- The attention and present admin registers have to be expanded to 256 bits. This is done in both the runtime code in **HdlOCCP.hh** and primitive code in **ocscp_rv.vhd**.

4.4 511 Worker Maximum

A maximum of 511 workers was investigated but was never functional. Alterations required for 511 worker limit are as follows.

- **OCCP_WORKER_CONFIG_WINDOW_BITS** set to 17 and **OCCP_MAX_WORKERS** set to 511 in **HdlOCCP.hh** for the core primitive code configuration.
- **worker_config_bits** set to 15 and **worker_max_nworkers** set to 511 in **platform_pkg.vhd** for the runtime code configuration.
- The attention and present admin registers have to be expanded to 512 bits. This is done in both the runtime code in **HdlOCCP.hh** and primitive code in **ocscp_rv.vhd**.

An assembly with 390 **bias_vhdl.hdl** workers was able to execute. However, several issues are preventing getting an assembly with 511 workers to function within the OpenCPI framework.

- When executing an assembly with over 256 **bias_vhdl.hdl** workers the output value seems to be wrapping. For example if the assembly has 340 bias workers and the starting value is 0 and each bais worker is adding 1 the output should be 340. However the output was 84, or 340-256, indicating something is causing the value to wrap on an 8 bit boundary.
- An assembly with 345 **bias_vhdl.hdl** workers would not run due to a worker timing out. This timeout error is only found when running local on the device, EOF error is shown when running from host. If the application was run over and over it would always be the same workout to timeout, however if the assembly was rebuilt then a different worker would timeout indicating this is probably dependent on how Vivado configures the FPGA. In an attempt to fix this issue, **worker_timeout_t** in **platform_pkg.vhd** was increased from 5 bits to 8 bits, this allowed for a 390 worker to run successfully. Increasing the worker_timeout_t to a 24 bit value (just to try something large) resulted in Vivado erroring indicating that the device ran out of LUTs for the assembly's HDL. There may be a happy medium between 24 bits and 8 bits where the device has enough resources, and the timeout is long enough for the assembly to function on the device; although this might not be a good solution. The actual cause of the timeout is not clear at this time.
- As the memory configuration allows for more workers the amount of configuration space decrease. During testing of an actual application assembly, the following error occurred when a worker had a large number of properties indicating that the configuration space for the worker was insuficent.
    - **Assertion failed: window == ((offset + nBytes) & (~0u << windowBits)) is false at ../gen/runtime/hdl/include/HdlWciControl.hh:82**