OpenPOWER Memory Bus Specification for OMI

Memory Function Unit Interface

Workgroup Specification

Revision 1.93 (April 26, 2021)

OpenPOWER™

www.openpowerfoundation.org

OpenPOWER Memory Bus Specification for OMI: Memory Function Unit Interface

Memory Work Group <memwg-chair@openpowerfoundation.org>
OpenPower Foundation

Revision 1.93 (April 26, 2021)

Abstract

This document is a Standard Track, Work Group Specification work product owned by the Memory Workgroup and handled in compliance with the requirements outlined in the *OpenPOWER Foundation Work Group (WG) Process* document. It was created using the *Master Template Guide* version 1.0.0. Comments, questions, etc. can be submitted to the public mailing list for this document at <memwg-opmb@mailinglist.openpowerfoundation.org>.

The OpenPOWER Memory Bus Specifiation defines the OpenPOWER Memory Bus Architecture for the development of Memory Function Units(MFU) and the integration of those MFUs into the OpenPOWER system structure. An MFU is a logic block developed by a member of the OpenPOWER eco-system to enable the integration of specialized memory technology and functional processing for data stored in the memory technology.

The OpenPOWER Memory Bus Specifiation is a standards track work product of the OpenPOWER Memory Workgroup.

Table of Contents

List of Figures

List of Tables

Preface

1. Conventions

The OpenPOWER Foundation documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Changes

At certain points in the document lifecycle, knowing what changed in a document is important. In these situations, the following conventions will used.

- *New text will appear like this.* Text marked in this way is completely new.
- ~~Deleted text will appear like this.~~ Text marked in this way was removed from the previous version and will not appear in the final, published document.
- Changed text will appear like this. Text marked in this way appeared in previous versions but has been modified.

Command prompts

In general, examples use commands from the Linux operating system. Many of these are also common with Mac OS, but may differ greatly from the Windows operating system equivalents.

For the Linux-based commands referenced, the following conventions will be followed:

$ prompt Any user, including the `root` user, can run commands that are prefixed with the $ prompt.

 # prompt The `root` user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

Document links

Document links frequently appear throughout the documents. Generally, these links include a text for the link, followed by a page number in parenthesis. For example, this link, Preface [vi], references the Preface chapter on page vi.

2. Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

Revision Date	Summary of Changes
October 11, 2020	• Revision 0.91 - Draft to review for voting

1. Overview

This document defines the OpenPOWER Memory Bus Architecture for the development of Memory Function Units (MFU) and the integration of those MFUs into the OpenPOWER system structure. A MFU is a logic block developed by a member of the OpenPOWER eco-system to enable the integration of specialized memory technology and functional processing for data stored in that memory technology.

This document is intended to define the framework for the creation of different MFUs to the same OPMB interface. For details on specific MFU implementations, please see the documentation for that MFU.

1.1. Who Should Read This Manual

This manual is intended for designers who plan to develop products that use the OPMB. Readers of this manual should be familiar with the documents listed in Section 1.3, “ Related Publications” [2]. In addition, individual implementations of the OPMB should have their own documentation for that implementation.

1.2. Document Organization

The OPMB Specification document is divided into two primary parts: the logical interface between the MFU and service layer abstracting the power system Open Memory Interface (OMI) called OMI Service Layer (OMI) and the register definitions used by the system firmware to initialize and manage the MFU. Together, these two environments define the OPMB. Implementation details and compliance with the architecture for a specific implementation are provided separately.

Document Division	Description
Chapter 1, “Overview” [1]	Describes this document, related documents, the intended audience, and other general information.
Chapter 2, “Introduction” [5]	Provides a high-level overview of the OpenPOWER Memory Bus Interface Architecture.
Chapter 3, “Interface Definitions” [8]	Detailed descriptions of the Signal Interfaces of the OpenPOWER Memory Bus.
Chapter 4, “Commands” [16]	Detailed description of Commands, Functions, and Addressing capabilities of the OpenPOWER Memory Bus.
Chapter 5, “Protocol and Timing” [26]	Detailed description of the Protocols and Timings of the OpenPOWER Memory Bus.
Chapter 6, “Registers” [27]	Describes control and status register maps of the OpenPOWER Memory Bus Interface. This chapter describes the common register sets that exist for all OPMB-MFU implementations that the firmware may use for initialization, monitoring and control of the OPMB interface. MFU-specific control and status registers are not described in this document. Descriptions of MFU-specific control and status registers are to be found in MFU-specific documentations.
Chapter 7, “Meta Bit Requirements” [42]	Describes the minimum requirements for use of metabits in an MFU attached to the OpenPOWER Memory Bus Interface.
Chapter 8, “Firmware Interaction” [44]	Describes the firmware interaction with an MFU attached to the OpenPOWER Memory Bus Interface.
Chapter 9, “Development, Prototyping and Bring-Up Insights” [45]	Provides insight into development, prototyping and bring-up of the MFU using OpenPOWER Memory Bus Interface in an OpenPOWER System.

1.3. Related Publications

The following documents can be helpful when reading this specification. For additional information please follow this link to the OpenPOWER Foundation. *https://openpowerfoundation.org/technical/resource-catalog/* .

Power ISA User Instruction Set Architecture - Book I (Version 3.0C)

Power ISA Virtual Environment Architecture - Book II (Version 3.0C)

Power ISA Operating Environment Architecture (Server Environment) - Book III-S (Version 3.0C)

I/O Design Architecture (IODA3)

Coherent Accelerator Processor Interface User's Manual

OpenCAPI 3.1 Transaction Layer Specification *https://opencapi.org/technical/specifications/*

OpenCAPI 3.0 Data Link Layer Specification *https://opencapi.org/technical/specifications/*

OpenPOWER Foundation Memory Workgroup *https://members.openpowerfoundation.org/wg/MEMWG/dashboard*

Github: Organization = OpenPOWER Foundation; Repository = Memory_WG *https://github.com/*

Github: Organization = OpenCAPI; Repository = omi_device_ice Gemini/ICE FPGA Memory Buffer - Working Example of MFU using OpenPOWER Memory Bus Specification *https://github.com/*

1.4. Conventions and Notation

This document orders bits and bytes in descending order from left to right. Thus, for a 4-byte word, bit 31 is the least-significant bit, and bit 0 is the most-significant bit.

LSb																													MSb
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Numbers are generally shown in decimal format, unless designated as follows:

- Hexadecimal values are preceded by an "x" and enclosed in single quotation marks or preceded by "0x".

  For example: x'0A00' or 0x0A00.

- Binary values in sentences are shown in single quotation marks.

  For example: '1010'.

Note: A bit value that is immaterial, which is called a "don't care" bit, is represented by an "X."

This document uses the following software documentation conventions:

1. Command or instruction names are written in **bold** type. For example: **afu_wr** and **afu_rd.**

2. Field names and variables are written in italic type. Required parameters are enclosed in angle brackets. Optional parameters are enclosed in brackets. For example: *afu <f,b>_wr[a]*.

This document uses the following symbols:

&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
%	Modulus
=	Equal to
! =	Not equal to
≥	Greater than or equal to
≤	Less than or equal to
x >> y	Shift to the right; for example, 6 >> 2 = 1; least-significant y bits are dropped
x << y	Shift to the left; for example, 3 << 2 = 12; least-significant y bits are replaced zeros
||	Concatenate

1.5. References to Registers, Fields, and Bits

Registers are referred to by their full name or by their short name (also called the register mnemonic). Fields are referred to by their field name or by their bit position. The following table describes how registers, fields, and bit ranges are referred to in this document and provides examples.

Table 1.1. Register References

Type of Reference	Format	Example
Reference to a specific register and a specific field using the register short name and the field name	Register_Short_Name[Field_Name]	MSR[R]
Reference to a field using the field name	[Field_Name]	[R]
Reference to a specific register and to multiple fields using the register short name and the field names	Register_Short_Name[Field_Name1, Field_Name2]	MSR[FE0, FE1]
Reference to a specific register and to multiple fields using the register short name and the bit positions.	Register_Short_Name[Bit_Number, Bit_Number]	MSR[52, 55]
Reference to a specific register and to a field using the register short name and the bit position or the bit range.	Register_Short_Name[Bit_Number]	MSR[52]
	Register_Short_Name[Starting_Bit_Number:Ending_Bit_Number]	MSR[39:44]
A field name followed by an equal sign (=) and a value indicates the value for that field.	Register_Short_Name[Field_Name]= *n*	MSR[FE0] = '1' MSR[FE] = x'1'
	Register_Short_Name[Bit_Number]= *n*	MSR[52] = '0' MSR[52] = x'0'
	Register_Short_Name[Starting_Bit_Number:Ending_Bit_Number]= *n*	MSR[39:43] = '10010' MSR[39:43] = x'11'
Where n is the binary or hexadecimal value for the field or bits specified in the brackets.		

1.6. Endian Order

The Power ISA supports both big-endian and little-endian byte-ordering modes. This spec uses little-endian. *Book I* of the *Power ISA* describes these modes.

2. Introduction

The OpenPOWER Memory Bus (OPMB) Specification for OMI (Open Memory Interface based on OpenCAPI 3.1) defines a memory function unit interface structure for attaching one or more memory funtion units (MFUs) to an OpenPOWER™ System with OpenCAPI using a Open Memory Interface (OMI). The intent for the creation of this documentation is to foster the development of an eco-system and enable the implementation of different memory buffering devices that support different MFUs and enable the use of different memory technologies in different configurations.

Physically, an OPMB-compliant memory buffering device can consist of a single chip, a multi-chip module (or modules), or multiple single-chip modules on a system board or other second-level package. The design depends on the technology used, and on the cost and performance characteristics of the intended design point.

Figure 2.1. An Abstract OPMB-Compliant Memory System

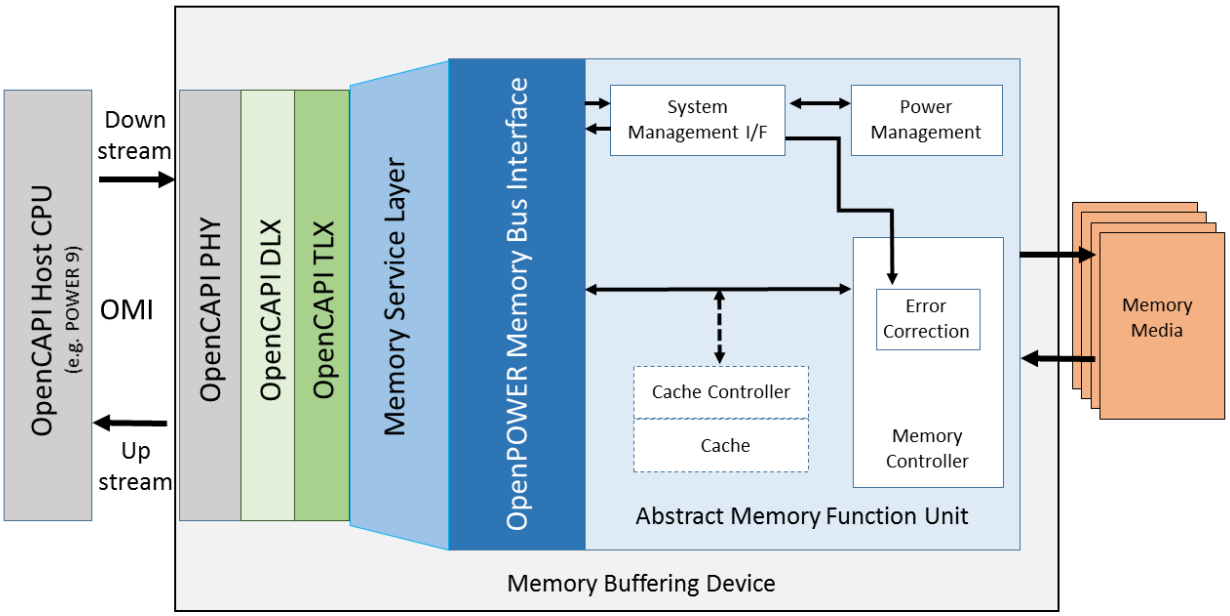


Figure 2.1, “An Abstract OPMB-Compliant Memory System” [5] illustrates a OPMB-compliant memory system using OpenCAPI, including: a host CPU with a Open Memory Interface (OMI), a memory buffering device that receives commands and data as well as return data and status information through the OMI to the host CPU, and memory media controlled by the memory buffering device.

2.1. Open Memory Interface (OMI)

The Open Memory Interface (OMI) is a physical layer interface that enables the transport of memory command, address and data encapsulated in flits through high speed Serdes links to and from the host CPU. The OMI is an open standard memory interface implemented using either the OpenCAPI

3.1 specifications. For more information on OpenCAPI, please visit the OpenCAPI Consortium at opencapi.org

2.2. Memory Service Layer (MSL)

The Memory Service Layer (MSL) connects the high-speed OMI to the OpenPOWER Memory Bus. The role of the MSL is to convert command, address and data encapsulated in flits transported over the OMI to OPMB-compliant format. The MSL also takes data and status information from the OPMB-compliant interface, converts and encapsulates them for return through the OMI back to the host CPU. The MSL is an IP block that interfaces between the OpenCAPI 3.1 host interface and the standard-based OpenPOWER Memory Bus. For more information on OpenCAPI, please visit the OpenCAPI Consortium at opencapi.org

2.3. OpenPOWER Memory Bus (OPMB)

The OpenPOWER Memory Bus (OPMB) is a virtual memory bus interface that exists within an OPMB-compliant memory buffering device. The OPMB enables an abstracted interface between a OMI-equipped host processor and memory media. The OPMB assembles memory commands from command flits provided by the MSL and passes the commands to the targeted Memory Function Unit. The OPMB also accepts read data returned by the Memory Function Unit(s) and passes it back to the MSL. The OPMB is a tagged, split-transaction interface, as commands received by the OPMB may be processed by the OPMB and the associated Memory Function Unit(s) in an ordering that is different than the ordering from which they were received by the OPMB.

2.4. Memory Function Unit (MFU)

An OPMB-compliant memory buffering device includes one or more MFUs.

The purpose of a MFU is to provide management functionality of memory media on behalf of the host CPU. It is envisioned that the OPMB should contain sufficient flexibility to support different MFUs that enable the attachment of different types of memory media to OpenPOWER CPU's. A MFU may also optionally implement logical functions so that logical operations may be performed on data stored in the memory media managed by the MFU.

2.5. Memory Attributes

2.5.1. Address Spaces

This section describes basic details on the address spaces.

Table 2.1 lists the sizes of memory address spaces.

Table 2.1. Sizes of Main Storage Address Spaces

Address Space	Size	Description
Physical Address Space	2^m bytes	Where m ≤ 64 (the max size is implementation specific, the OpenCAPI channel supports up to 64b of address, however depending the Power system and the OpenCAPI bus being attached to the actual physical address space limitiations will vary. Address space, limitaions and configuration. Contact OpenPOWER system designer and OpenCAPI specifications.

Address Space	Size	Description
Register Address Space	2^n bytes	Where n ≤ 32
Note: The values of "m" and "n" are implementation dependent.		

2.6. Conformance to the OPMB

In the future a behavior model may be available. Otherwise, follow the guidance provided in the example design (Gemini/ICE) available on github (link provided in chapter 1) to ensure spec compliance and actual, physical interoperability.

1. Must implement at least one MFU

3. Interface Definitions

This chapter describes the interface definitions used to specifiy the OpenPOWER Memory Bus (OPMB)as attached to an OpenCAPI enabled OpenPower system. Valid interfaces include both the upstream and downstream command, data, address and control interfaces. All appropriate signals required to implement the OPMB Memory Service layer are defined within this chapter.

Note:

Register interface

The sideband register interface definition is outside the scope of this document. A detailed register interface description is provided with the MSL user implementation guide.

3.1. Interface Variations

There are a wide range of potential applications which may employ the OPMB MSL, this spec covers one variant of the MSL interface. This variant is most applicable to FPGA type implementations. At a later time other variants may be described within this spec. Additional variations may be optimized for higher speed applications such as ASICs. This section will cover the signaling of any specified MSL variant.

Table 3.1. MSL Interface Variations

OMI Data Rate	OPMB Speed	OMI to OPMB Gear Ratio	DS Flits per OPMB Cycle	US Flits per OPMB Cycle
FPGA Optimal OPMB				
25.6 Gbps	400 MHz	64:1	1	1
21.33 Gbps	333 MHz	64:1	1	1

Figure 3.1. OPMB Downstream Signal Interface

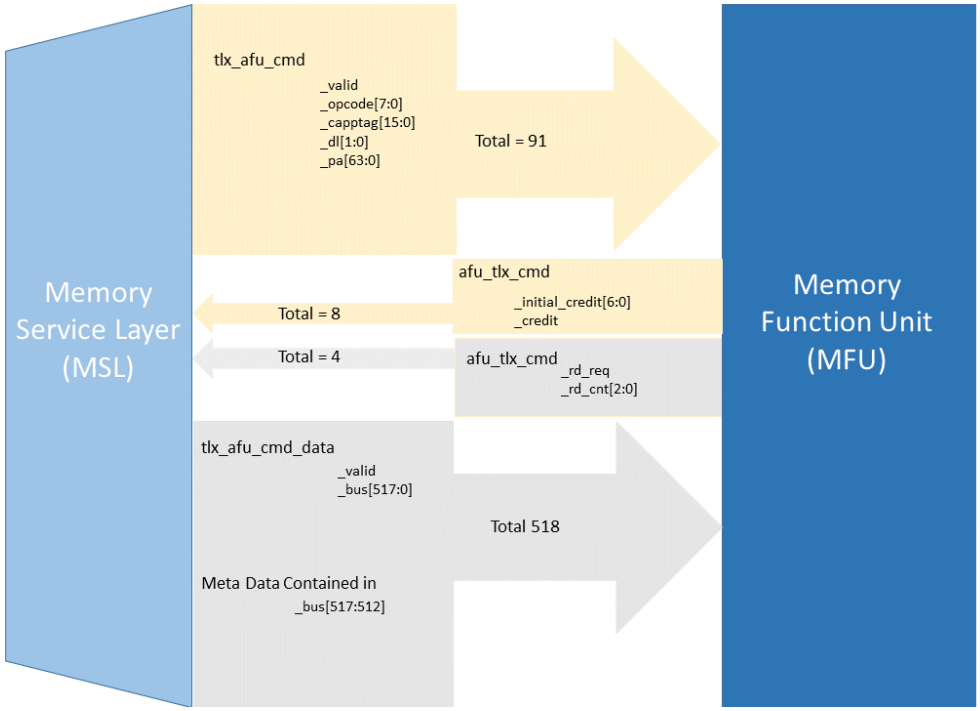


Figure 3.1, “OPMB Downstream Signal Interface” [9] Illustration of the OPMB downstream signal interface between the MSL and MFU

Figure 3.2. OPMB Upstream Signal Interface

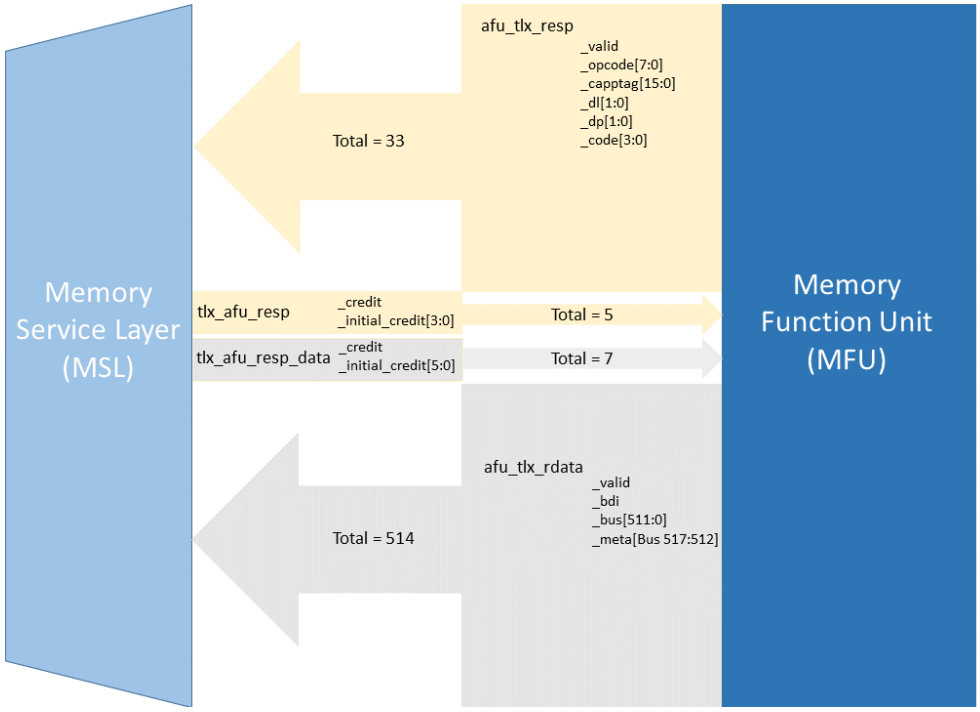


Figure 3.2, “OPMB Upstream Signal Interface” [9] Illustration of the OPMB upstream signal interface between the MSL and MFU

Figure 3.3. OPMB Inband Configuration Signal Interface

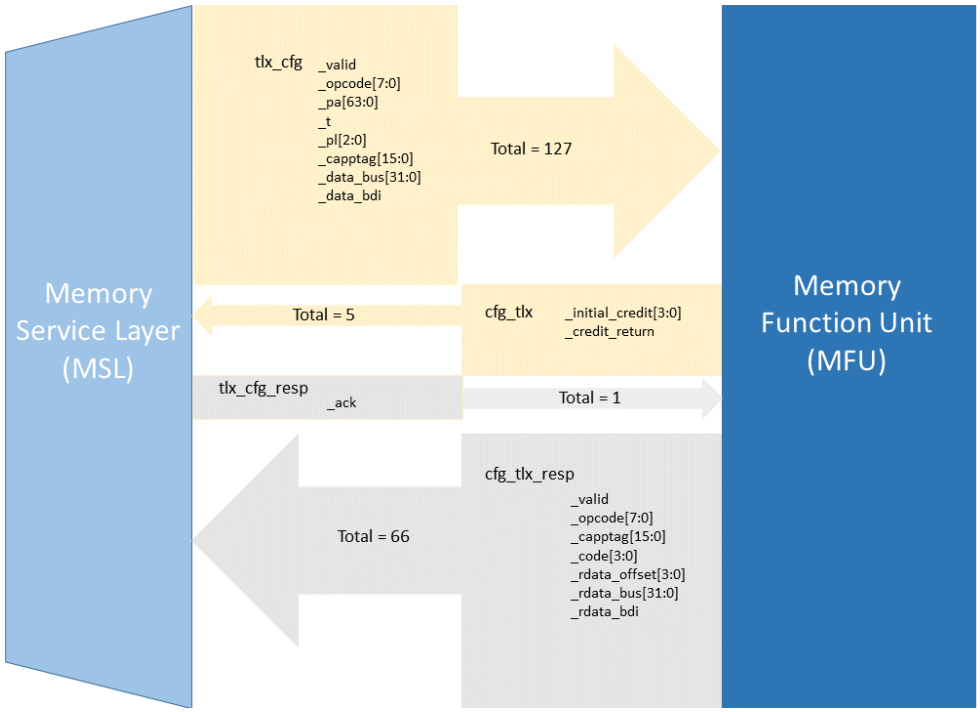


Figure 3.3, “OPMB Inband Configuration Signal Interface” [10] Illustration of the OPMB inband configuration interface between the MSL and MFU

3.2. Interface Signaling

All MSL (memory service layer) interfaces are defined here. The defined interfaces include upstream and downstream command, data, address and control. Each subsection defines all variations of each interface as required to implement any variation of the MSL.

3.2.1. Downstream Interface Signals

This section describes the downstream command and data signals for of the MSL to OPMB interface. This interface presents at most one command, one inband configuration request and 64B of data per MSL clock cycle.

Table 3.2. Downstream MSL Command and Data Signals

Signal Name	Width	MSL I/O	Description
tlx_afu_cmd_valid	1	O	When high, indicates that a valid command is present for the associated downstream flit.
tlx_afu_opcode[7:0]	8	O	When the associated tlx_afu_cmd_valid bit is high, this bus will have a valid command encoding. The command encodings are provided elsewhere in this specification.
tlx_afu_cmd_capptag[15:0]	16	O	When the associated tlx_afu_cmd_valid bit is high, this bus will have a valid command tag. A command tag may not be reissued

Signal Name	Width	MSL I/O	Description
			by the service layer until the tag has been released by the MFU through an appropriate upstream response.
tlx_afu_cmd_pa[63:0]	64	O	This bus is valid when a valid command is present. When the command is present, the encoding represents a physical address (PA).
tlx_afu_cmd_dl[1:0]	2	O	This bus is valid when the tlx_afu_cmd_valid is high. The data length (DL) indicates the number of data bytes associated with a command. 0b00 = Reserved, 0b01 = 64B, 0b10 = 128B and 0b11 = 256B. 0b00 = Reserved 0b01 = 64B 0b10 = 128B 0b11 = 256B (Reserved: Not supported in current OpenPOWER Systems)
afu_tlx_cmd_initial_credit[6:0]	7	I	Describes the number of downstream commands that the MFU can accept.
afu_tlx_cmd_credit	1	I	Command credit return. Indicates that the MFU can now accept another command.
tlx_afu_data_valid	1	O	When high, indicates that valid data is present for the associated downstream flit.
tlx_afu_data[517:0]	518	O	Dowstream data associated with a write command. Includes 512b of data and 6b of meta data
tlx_afu_data_sue (encoded in data)	2	O	Meta data associated with the downstream data, SUE bit. Included in tlx_afu_data[517:512]. Minimum SUE requirements are described in chapter 7 of this document.
afu_tlx_data_rd_req	1	I	Command Read Request from the MFU to the MSL.
afu_tlx_rd_req_cnt[2:0]	3	I	The amount of data the MFU wants to read from the MSL 3b’000 512 Bytes 3b’001 64 Bytes 3b’010 128 Bytes 3b’011 256 Bytes 3b’100 192 Bytes 3b’101 320 Bytes 3b’110 384 Bytes 3b’111 448 Bytes

3.2.2. Upstream Interface Signals

This section describes the upstream command and data signals for the OPMB to MSL interface. The interface consumes one flit per MSL clock cycle.

Table 3.3. Upstream MSL Command and Data Signals

Signal Name	Width	MSL I/O	Description
afu_tlx_resp_valid	1	I	When high, indicates that a valid response has arrived from the MFU. Any response field that pertains to the arriving opcode should contain valid information at this time. Other response fields are undefined.

Signal Name	Width	MSL I/O	Description
afu_tlx_resp_opcode[7:0]	8	I	When afu_tlx_resp_valid is high, indicates the opcode associated with the response.
afu_tlx_resp_capptag[15:0]	16	I	When afu_tlx_resp_valid is high, indicates the tag associated with the response,
afu_tlx_resp_dl[1:0]	2	I	When afu_tlx_resp_valid is high, indicates the response data length. 2b'00 Reserved 2b'01 64 Bytes (1 data packet) 2b'10 128 Bytes (2 data packets) 2b'11 256 Bytes (4 data packets) - Reserved: Not currently supported for OpenPOWER Systems
afu_tlx_resp_dp[1:0]	2	I	When afu_tlx_resp_valid is high, indicates the data content of the current response packet
afu_tlx_resp_code[3:0]	4	I	When afu_tlx_resp_valid is high, describes the reason for a failed transaction. Details on the encoding will be described elsewhere in this document.
tlx_afu_resp_initial_credit[5:0]	6	O	The MFU must maintain a counter of credits for sending responses to the MSL. The counter should be decremented whenever a response is sent to the MSL. The counter should be incremented whenever this signal is asserted. The counter should be initialized using the value from tlx_afu_resp_initial_credit. The MFU should not send any responses to the MSL when the credit count is zero.
tlx_afu_resp_credit	1	O	When this signal is asserted, the MFU can send another response to the MSL. The MFU must maintain a counter of credits for sending responses to the MSL. The counter should be decremented whenever a response is sent to the MSL. The counter should be incremented whenever this signal is asserted. The counter should be initialized using the value from tlx_afu_resp_initial_credit. The MFU should not send any responses to the MSL when the credit count is zero.
afu_tlx_rdata_valid	1	I	When high, indicates that a valid packet of response data has arrived from the MFU. The data bus and the bdi bit contain valid information.
afu_tlx_rdata_bus[511:0]	512	I	When afu_tlx_rdata_valid is high there is valid response data on this bus.
afu_tlx_rdata_meta[Data Bus 517:512]	6	I	When afu_tlx_rdata_valid is high there is valid response data on this bus.
afu_tlx_rdata_bdi	1	I	When afu_tlx_rdata_valid is high and this signal is high the response data is bad.
afu_tlx_rdata_credit	1	I	When this signal is asserted, the MFU can send another response data packet to the MSL. The MFU needs to maintain a counter of credits for sending response data packets to the MSL. The counter should be initialized using the value from tlx_afu_rdata_initial_credit. The counter should be incremented whenever tlx_afu_rdata_credit is asserted. The counter should be decremented whenever a response with data is sent to the MSL. (The counter should be decremented by the number of immediate data packets that are associated with the response.) The MFU should not send any responses with immediate data to the MSL unless there are enough data credits in this counter to support the number of immediate data packets associated with the response.

3.2.3. Inband Configuration Interface Signals

This section describes the inband configuration command and data signals for the OPMB to MSL interface.

Table 3.4. MSL Inband Configuration Signals

Signal Name	Width	MSL I/O	Description
tlx_cfg_valid	1	O	When high, indicates that a valid inband configuration command has arrived from the MSL. Any other signals that pertains to the arriving command should contain valid information at this time.
tlx_cfg_opcode[7:0]	8	O	When tlx_cfg_valid is high, indicates the opcode associated with the configuration command
tlx_cfg_pa[63:0]	64	O	When tlx_cfg_valid is high, indicates the the physical address of the command.
tlx_cfg_t	1	O	When tlx_cfg_valid is high, indicates the type of configuration command '0': Indicates a type 0 configuration read or write command '1': Indicates a type 1 configuration read or write command
tlx_cfg_pl[2:0]	3	O	When tlx_cfg_valid is high, indicates the partial length 3b'000 1 Byte 3b'001 2 Bytes 3b'010 4 Bytes 3b'011 8 Bytes 3b'100 16 Bytes 3b'101 32 Bytes 3b'110-111 Reserved
tlx_cfg_capptag[15:0]	16	O	When tlx_cfg_valid is high, specifies the unique tag associated with the configuration command
tlx_cfg_data_bus[31:0]	32	O	When tlx_cfg_valid is high, specifies the data associated with the configuration command
tlx_cfg_data_bdi	1	O	When tlx_cfg_valid is high, indicates whether the data on tlx_cfg_data_bus is bad
cfg_tlx_initial_credit[3:0]	4	I	Initial setting of credits to be used to send configuration commands to the MFU
cfg_tlx_credit_return	1	I	Indicates that the MFU is ready and able to accept an additional configuration command.
cfg_tlx_resp_valid	1	I	When asserted this signal indicates that a valid configuration command response is on the configuration command response.
cfg_tlx_resp_opcode[7:0]	8	I	Indicates that a valid response has arrived from the MFU. Any response field that pertains to the arriving opcode should contain valid information at this time
cfg_tlx_resp_capptag[15:0]	16	I	When cfg_tlx_resp_valid is asserted, this bus will have the unique tag associated with the command response.
cfg_tlx_resp_code[3:0]	4	I	When cfg_tlx_resp_valid is asserted, this bus will have the response code. Response codes are describes in another location of this document.
tlx_cfg_resp_ack	1	O	The MSL pulses this signal when the config response is accepted. The MFU configuration logic must then drop the

Signal Name	Width	MSL I/O	Description
			cfg_tlx_resp_valid signal for at least one cycle before sending another config response.

3.3. Interface Parity Protection

Both the upstream and downstream interfaces are protected by parity. The largest grouping of interface signal bits covered by parity is 32. The parity scheme used is odd parity where the parity bit is generated such that the signal bits covered by said parity bit and said parity bit represent an odd number of '1's. For example if three signal bits are covered by odd parity and those bits are binary '000', the parity bit would be binary '1'. If those same signal bits were binary '100', the parity bit would be binary '0'.

Handling parity errors in a manner to allow recovery of the parity fail is difficult and would require a complex solution. Therefore, in order to get the most out of the parity protection with the least cost solution parity protection will be handled differently for data vs. all other parity protected bits on the downstream and upstream buses.

3.3.1. Downstream Parity Protection

3.3.1.1. Data Parity Error Handling

Upon a parity error detection by the MFU on a downstream write data transfer from the MSL, the MFU will mark the appropriate meta bits to indicate a write error SUE with the downstream data. The SUE encoding must be maintained within the meta bits of the associated data so that during a read the data will be returned with the marked SUE bit indicating an failure on the write.

The MFU designers may choose to record other information regarding the fail.

3.3.1.2. All Other Parity Error Handling

Upon a parity error detection by the MFU on a downstream transfer from the MSL on non data signals, the MFU at a minimum should disregard the transfer. It is recommended that the parity fail be logged in a error reporting register which may be accessed by the system. When the MFU disregards the erroneous transfer the command associated with the transfer will naturally timeout in the host.

3.3.2. Upstream Parity Protection

3.3.2.1. Data Parity Error Handling

Upon a parity error detection by the MSL on a upstream read data transfer from the MFU, the MSL at a minimum will mark the appropriate meta bits to indicate a read SUE with the upstream data.

The MSL may choose to record other information regarding the fail.

3.3.2.2. All Other Parity Error Handling

Upon a parity error detection by the MSL on a upstream transfer from the MFU on non data signals, the MSL at a minimum should zero out all of the bits covered by the parity bit. By zeroing out the bits

associated with the parity error the MSL will effectively be dropping the command information which will result in a timeout. It is recommended that the parity fail be logged in a error reporting register which may be accessed by the system.

3.4. Additional Required Interfaces

The OMI channel is not the only interface utilized by the OpenPOWER System to communicate with a memory functional unit. Additional interfaces such as I2C / SMBus are often used in key steps of enabling an MFU on an OMI memory module.

3.4.1. I2C / SMBus

In addition to the OMI channel OpenPOWER OMI enabled systems also use an I2C SMBus for early initialization steps. The I2C/SMBus can also be used to access MFU registers should a failure occur which causes the OMI link to go down and MMIO/Inband access is no longer available. This interface deploys specific protocols to access registers as well as issue specific commands and check responses.

The I2C/SMBus may be considered outside the scope of this document. However, because it is a required component of a memory device deployed on the OMI channel in OpenPOWER systems it should be noted. This logic would typically be deployed within the MSL to access key required registers. However, some modification to expand access to additional MFU registers may be desireable. This type of expansion is outside the scope of this document. Please refer to the example design Gemini/ICE located in Github for an example of implementation (section 1.3 has a link to the design).

The I2C / SMBus should be designed to follow the SMBus specification and allow for speeds up to 400Kbps.

4. Commands

This chapter describes the command set specified for the OpenPOWER Memory Bus (OPMB) Memory Service Layer (MSL). Commands and their respective encodings described within this section represent the valid commands presented on the downstream command interface signals and the I2C / SMBus signals defined in chapter 3 of this specification.

4.1. MSL Command Definitions

This section describes the command set specified for the OpenPOWER Memory Bus (OPMB) Memory Service Layer (MSL). Commands and their respective encodings described within this section represent the valid commands presented on the downstream command interface signals defined in chapter 3 of this specification. The downstream commands are presented on the downstream interface signals with prefix ds_flt0_cmd. It is worth noting that not all commands may apply for a given MFU. In the case a command is presented on the command bus which does not directly apply to a function or feature of the implemented MFU the presented command may be interpreted as another similar command or in some specific cases a predefined default response mechanism may be employed by the MFU. Appropriate command substitutions and default responses are listed in the command descriptions.

Any commands presented to the MFU on the MSL and should be responded to in accordance with the protocols defined within this specification. The commands are defined by command groupings, associated command information, response information, description and a general encoding for the particular command type. Certain commands may have fixed bits in their encoding as well as variable bits. This section defines the fixed bits, later in this chapter the variable command bit encodings are described.

It should be noted that command encodings not defined are reserved for other applications. Good design practices would dictate that should a command be presented with an undefined encoding, that the undefined command be logged and an indication of this event be stored in an accessable status register.

Table 4.1. OpenCAPI 3.1 Device Interface Class Commands

Command	Encoding	Associated Command Info	Response Info	Description
Configuration Commands				
config_read	1110 0000	-PA -CAPPTag -pL	-Response Data w/ Tag -Response w/Tag -mem_rd_response or mem_rd_response.ow or mem_rd_response.xw or mem_rd_fail	The host is issuing a read to the MFU's configuration address space. The number of bytes transfered is specified by pLength (pL) field. The starting address shall be naturally aligned based on the number of bytes requested. For this command, the pLength value is limited to a transfer size of 1, 2, or 4 bytes. That is, the specification of pLength shall limit the transfer size to 2n bytes where n = {0..2}. The response to this command is mem_rd_response, mem_rd_response.ow, mem_rd_response.xw, or mem_rd_fail. When a mem_rd_response is received, the data is found in the 64-byte data flit (only one is returned for this command). The data is address aligned. When a mem_rd_response.ow is received, the data is found in the 32-byte datum field found

				in a control flit (only one is returned for this command). The data is address aligned. When a mem_rd_response.xw is received, the data is found in the 8-byte datum field found in a control flit and is address aligned. The mem_rd_fail response indicates the operation failed. See the response packet for encoding and specifications.
config_write	1110 0001	-PA -CAPPTag -pL -Data	-Response with CAPPTag -mem_wr_response or mem_wr_fail	The host is issuing a write to the AFU's configuration address space. The number of bytes transfered is specified by pLength (pL) field. The starting address shall be naturally aligned based on the number of bytes requested. For this command, pLength is limited to a transfer size of 1, 2, or 4 bytes. That is, the specification of pLength shall limit the transfer size to 2n bytes where n = {0..2}. The T field, defined in Table 2-1 on page 56, specifies the configuration type of the command. When T is set to 0, the AFU learns its BDF value located in the PA field. This command is specified with immediate data. The data may be sent in a data flit, or it may be sent in a 32- or 8-byte datum field found in control flits. Credits for both the VC and DCP shall be obtained before this command is serviced by the TL. The mem_wr_response and mem_wr_fail responses to this command indicate the status of the write to memory operation. The mem_wr_response indicates a successful completion of the operation. The mem_wr_fail response indicates that the operation failed. See the response packet for encoding and specifications.
Write and Read Data Commands				
write_mem	1000 0001	-PA -CAPPTag -dL -R	-Response w/Tag -mem_wr_response or -mem_wr_fail	The host is writing a 64-, 128-, or 256-byte block of data to the MFU memory. The starting address is specified by the PA field and shall be naturally aligned based on the length of the data as specified by the dLength (dL) field. This command is specified with immediate data. The data may be transfered using data flits or 32-byte datum fields found within a control flit. Credits for both the VC and DCP shall be obtained before this command is serviced by the TL. The mem_wr_response and mem_wr_fail responses to this command indicate the status of the write to memory operation. The mem_wr_response indicates a successful completion of the operation. The mem_wr_fail response indicates that the operation failed. See the response packet description for encoding and specifications
pr_wr_mem	1000 0110	-PA -CAPPTag -pL -R	-Response w/Tag -mem_wr_response or -mem_wr_fail	Note: OpenPOWER systems may have limitied support for partial write/read commands. Check with the systems engineers to determine if these commands are supported. The host is writing the data to the MFU memory. The starting address is specified by the PA and shall be naturally aligned based on the length of the data as specified by the pLength (pL) field. The combination of the address and the pLength shall not cross a 64-byte address boundary. This command is specified with immediate data. The data may be sent in a data flit, or it may be

				sent in a 32- byte datum field. When the pL field indicates a length of 8 bytes or less, the data may be sent in an 8-byte datum field found in control flits. Credits for both the VC and DCP shall be obtained before this command is serviced by the TL. The mem_wr_response and mem_wr_fail responses to this command indicate the status of the write to memory operation. The mem_wr_response indicates a successful completion of the operation. The mem_wr_fail response indicates that the operation failed. See the response packet for encoding and specifications.
write_mem.be	1000 0010	-PA -CAPPTag -Byte Enable	-Response w/Tag -mem_wr_response or -mem_wr_fail	The host is writing a 64-byte data block to the MFU memory using a 64-bit byte-enable field. Each bit corresponds to one byte of the 64-byte aligned data block specified by the PA, where bit 0 of the BE determines if byte 0 of the data is written. When BE(n) is set to 1, byte n is written where n={0..63}. This command is specified with immediate data. The data shall be sent using a single 64-byte data flit. Credits for both the VC and DCP shall be obtained before this command is serviced by the TL. The mem_wr_response and mem_wr_fail responses to this command indicate the status of the write to memory operation. The mem_wr_response indicates a successful completion of the operation. The mem_wr_fail response indicates that the operation failed. See the response packet for encoding and specifications.
write_meta	1000 0011	-PA -CAPPTag -Mask -Meta -dL	-Response w/Tag -mem_wr_response or -mem_wr_fail	This command updates the metadata associated with a 64-, 128- or 256-byte naturally aligned data block held in MFU memory. The update is performed under the control of the Mask field. The Meta field provides the new data used to update the metadata held by the MFU memory for each 64-byte block of the data block specified by the address field (PA) and the dLength field (dL). Support of metadata and metadata operations are optional. Implementations shall respond to the receipt of a write_meta command regardless of its support for the operation. The mem_wr_response and mem_wr_fail responses to this command indicate the status of the metadata update operation. The mem_wr_response indicates a successful completion of the operation or that the operation is not supported. The mem_wr_fail response indicates that the operation is supported and has failed. See the response packet for encoding and specifications.
rd_mem	0010 0000	-PA -CAPPTag -dL	-Response Data w/ Tag -Response w/Tag -mem_rd_response or -mem_rd_response.ow or	The host is requesting data to be read from the MFU memory. The starting address specified by the PA, supports a critical OW request. The data is a naturally aligned data block with a length specified by the dLength field (dL). See the host's platform architecture for the specification of the MAD field. The response to this command is mem_rd_response, mem_rd_response.ow, or

			-mem_rd_fail	mem_rd_fail. The mem_rd_fail response indicates the operation failed. See the response packet for encoding and specifications.
pr_rd_mem	0010 1000	-PA -CAPPTag -pL -R	-Response Data w/ Tag -Response w/Tag -mem_rd_response or -mem_rd_response.ow or -mem_rd_response.xw or -mem_rd_fail	Note: OpenPOWER systems may have limitied support for partial write/read commands. Check with the systems engineers to determine if these commands are supported. The host is requesting data to be read from the MFU memory. The number of bytes transfered is specified by pLength field(pL), and the starting address shall be naturally aligned based on the number of bytes requested. The pLength field limits the transfer size to 2n bytes where n = {0..5}. The response to this command is mem_rd_response, mem_rd_response.ow, read_response.xw, or mem_rd_fail. When a mem_rd_response is received, the data is found in the 64-byte data flit (only one is returned for this command). The data is address aligned. When a mem_rd_response.ow is received, the data is found in the 32-byte datum field specified by some control flits. The data is address aligned within the datum field. The read_response.xw response may be used only when the data length specified by pL is 8 or fewer bytes. The mem_rd_fail response indicates the operation failed. See the response packet for encoding and specifications.
Other Commands				
intrp_rdy	0001 1010	-MFUTag -Response Code	-intrp_response or	The host is sending an asynchronous status notification for a previously attempted interrupt. The MFU determines its actions based on the Resp_code received. The remaining fields of the command identify the prior TLX command. The TL is required to maintain the order of the matching intrp_resp carrying the MFUTag and the Resp_code = intrp_pending when loading the VC. That is, the TL shall ensure that the response packets precede the intrp_rdy command in the VC.
nop	0000 0000	-MFUTag -Response Code	-NA	This command has no operands and preforms no action. It is discarded at the MSL.

4.2. I2C / SMBus Command Definitions

This section describes the required supported I2C / SMBus commands. These commands are typical commands that the system would issue to access registers, check status, and initiate specific actions on the memory device.

OpenPOWER Systems utilize 64b registers. Typical I2C / SMBus operations deployed by the current MSL are 32b. Therefore, to access 64b registers two accesses are required, a 32b "left" side access and a 32b "right" side access. The firmware always issues a read to the left side first and the right side second. The same applies to a write. Both the left side and right side are always accessed in pairs. The firmware shall always issue a 32b left side access followed by a 32b right side access, no other variation is allowed.

4.2.1. I2C Boot Config Command

The Boot Config Command is an integral part of the initialization process. Once status indicates a runtime mode the system will continue with initialization. The Boot Config command is used by the system to indicate basic core configuration with the MFU and to kick off training of the DL layer. Some MFU may automatically start the DL training sequence as soon as reset is deasserted, whatever the capabilities and design of the MFU may be, at a minimun, the MFU shall confirm that the boot config command matches with its expectations before returning a good non zero status. If a MFU can operate at multiple speeds, this command should be used to begin the initialization of the MFU for that speed and to start OMI training. In order to issue an I2C Status Commmand use the following flow on the I2C bus.

8b command ID encoding = 0x01

Figure 4.1. I2C Boot Config Command

- Boot Config Command
  - START
  - 7b Device Address + 1b R/W# bit
    - Memory Buffer 0b010,0000 + R/W# 0b0 = 0x40
  - Byte 0 = Command ID = 0x01 (Boot Config Command)
  - Byte 1 = Number of bytes to follow this byte = 0x04 for 32b write data
  - Byte 2 = Data Byte 0
  - Byte 3 = Data Byte 1
  - Byte 4 = Data Byte 2
  - Byte 5 = Data Byte 3
  - STOP

Figure 4.2. I2C Boot Config Command Encodings

31:16-> Reserved for future use
14:13 -> Boot Mode
- 0b00 = Normal Boot
- 0b01 Manufacturing Mode
- 0b10 Product Qual Mode
- 0b11 Reserved

12 -> OMI Loopback Testing
- 0b0 = No loopback testing
- 0b1 = Perform loopback testing

11:10 -> Transport Layer
- 0b00 = OMI
- All others reserved

9:8 -> Boot Config Command Stage
- 0b00 = Stage 0: Only SerDes will be Initialized
- 0b01 = Stage 1: DL Training will be run
- All others reserved

7 -> DFE Enable if supported
6:4 -> Lane Mode
- 0b000 = Undefined, error
- 0b001 = 8 Lane
- 0b010 = 4 Lane
- All others reserved

3:0 -> OMI Frequency
- 0b0000 = Undefined, error
- 0b0001 = 21.33 Gbps
- 0b0010 = 23.46 Gbps
- 0b0011 = 25.6 Gbps
- All others reserved

After issuing a Boot Config command the status shall be checked by issuing a status command over I2C. The following figure shows the status bit encodings for the boot config command.

Figure 4.3. I2C Boot Config Command Status Encodings

0x00 = Success
0x01 = Invalid Boot Config Command
0x02 = Loopback Test Fail
0x03 = OMI SerDes Init Fail
0x04 = DLx Config Fail
0x10 = Lane Inversion configuration fail
0x20 = SerDes hardware parity and UECC errors
0xFE = I2C Interface Busy

4.2.2. I2C Status Command

The Status Command is an integral part of the initialization process. After deasserting reset to the OMI channel the system begins to poll the memory buffer using the status command. Once status indicates a runtime mode the system will continue with initialization. In order to issue an I2C Status Commmand use the following flow on the I2C bus.

8b command ID encoding = 0x02

Figure 4.4. I2C Status Command

- Status Command
  - START
  - 7b Device Address + 1b R/W# bit
    - Memory Buffer 0b010,0000 + R/W# 0b0 = 0x40
  - Byte 0 = Command ID = 0x02 (Status Command)
  - STOP
  - START
  - 7b Device Address + 1b R/W# bit
    - Memory Buffer 0b010,0000 + R/W# 0b1 = 0x41
  - Byte 0 = Number of bytes to follow this byte = 0x04 for 32b read data
  - Byte 1 = Data Byte 0
  - Byte 2 = Data Byte 1
  - Byte 3 = Data Byte 2
  - Byte 4 = Data Byte 3
  - STOP

Figure 4.5. I2C Status Command Response Encodings

31:24-> Reserved for future use
23:18 -> Reserved
17:16 -> Boot Stage
- 0b00 = NA
- 0b01 = Boot ROM Stage or Not initialized
- 0b10 = Reserved, do not use
- 0b11 = Ready, Runtime

15:8 -> MFU Status Code, depends on MFU
7:0 -> Command ID that status is reported for

4.2.3. I2C Register Address Latch Command

In order to issue an I2C Register Address Latch Commmand use the following flow on the I2C bus.

8b command ID encoding = 0x03

Figure 4.6. I2C Register Address Latch Command

- Address Latch Command
  - START
  - 7b Device Address + 1b R/W# bit
    - Memory Buffer 0b010,0000 + R/W# 0b0 = 0x40
  - Byte 0 = Command ID = 0x03 (Address Latch Command)
  - Byte 1 = Number of bytes to follow this byte = 0x04 for a 32b address
  - Byte 2 = Byte 0 of 32b address -> Left most byte of 32b address
  - Byte 3 = Byte 1 of 32b address
  - Byte 4 = Byte 2 of 32b address
  - Byte 5 = Byte 3 of 32b address -> Right most byte of 32b address
  - STOP

After issuing a Register Latch command the status should be checked by issuing a status command over I2C. The following figure shows the status bit encodings for the Address Latch command.

Figure 4.7. I2C Register Address Latch Command Status Encodings

0x00 = Success
Non-Zero = Error

4.2.4. I2C Register Read Command

In order to issue an I2C Register Read Commmand the Register Address Latch Command should be issued first followed by the Register Read Command. Refer to the following flow on the I2C bus.

8b command ID encoding = 0x04

Figure 4.8. I2C Register Read Command

- Address Latch Command
  - START
  - 7b Device Address + 1b R/W# bit
    - Memory Buffer 0b010,0000 + R/W# 0b0 = 0x40
  - Byte 0 = Command ID = 0x03 (Address Latch Command)
  - Byte 1 = Number of bytes to follow this byte = 0x04 for a 32b address
  - Byte 2 = Byte 0 of 32b address -> Left most byte of 32b address
  - Byte 3 = Byte 1 of 32b address
  - Byte 4 = Byte 2 of 32b address
  - Byte 5 = Byte 3 of 32b address -> Right most byte of 32b address
  - STOP
- Read Command
  - START
  - 7b Device Address + 1b R/W# bit
    - Memory Buffer 0b010,0000 + R/W# 0b0 = 0x40
  - Byte 0 = Command ID = 0x04 (Read Command)
  - Byte 1 = Number of bytes to follow this byte = 0x04 for a 32b address
  - Byte 2 = Byte 0 of 32b address -> Left most byte of 32b address
  - Byte 3 = Byte 1 of 32b address
  - Byte 4 = Byte 2 of 32b address
  - Byte 5 = Byte 3 of 32b address -> Right most byte of 32b address
  - START
  - 7b Device Address + 1b R/W# bit
    - Memory Buffer 0b010,0000 + R/W# 0b1 = 0x41
  - Byte 0 = Length -> should receive a 0x04
  - Byte 1 = Data byte 0
  - Byte 2 = Data byte 1
  - Byte 3 = Data byte 2
  - Byte 4 = Data byte 3
  - STOP

After issuing a Read command the status should be checked by issuing a status command over I2C. The following figure shows the status bit encodings for the Read command.

Figure 4.9. I2C Read Command Status Encodings

0x00 = Success
<address> = Invalid address

4.2.5. I2C Register Write Command

In order to issue an I2C Register Write refer to the following flow on the I2C bus.

8b command ID encoding = 0x05

Figure 4.10. I2C Register Write Command

- Write Command
  - START
  - 7b Device Address + 1b R/W# bit
    - Memory Buffer 0b010,0000 + R/W# 0b0 = 0x40
  - Byte 0 = Command ID = 0x05 (Write Command)
  - Byte 1 = Number of bytes to follow this byte = 0x08 for a 32b address + 4 bytes data
  - Byte 2 = Byte 0 of 32b address -> Left most byte of 32b address
  - Byte 3 = Byte 1 of 32b address
  - Byte 4 = Byte 2 of 32b address
  - Byte 5 = Byte 3 of 32b address -> Right most byte of 32b address
  - Byte 6 = Data byte 0
  - Byte 7 = Data byte 1
  - Byte 8 = Data byte 2
  - Byte 9 = Data byte 3
  - STOP

After issuing a Write command the status should be checked by issuing a status command over I2C. The following figure shows the status bit encodings for the Write command.

Figure 4.11. I2C Write Command Status Encodings

0x00 = Success
0xC0 = Error

5. Protocol and Timing

This chapter describes the protocol and timing of the OpenPOWER Memory Bus (OPMB) MSL. It is reccomended that the reader of this document utilize the example design provided in github (link available in chapter 1) as well as OpenCAPI Specification noted in section 1.3 to review the protocol and timing interactions of the OPMB MSL.

Note

There is a maximum response latency for commands. If this latency is not met then the upstream timeouts will trigger. This is dependent on the host system. The user should be aware of any system level timeouts when designing an MFU.

Maximum Response Latency <= Host system dependent. Check with system engineers.

6. Registers

The register definition including the registers and access methods are outside the scope of this specification. MSL registers and access interface will be described in the user implementation guide.

6.1. Inband Register Access

It should be noted that during the inband register access described in chapter 4, some inband accesses may be directed to the MSL. These accesses will be handled by the MSL and an upstream response will be sent as appropriate.

6.2. Minimum Required Register Set

This section defines the minimum required register set that shall be deployed in the MSL for operation in OpenPOWER Systems with an OMI3.1 memory interface. These registers include registers that pertain to the OpenCAPI 3.1 specification and registers that must be deployed in the MFU.

Table 6.1. Required registers

RTL Mnemonic	Address Offset	Description
OpenCAPI 3.1 Configuration Registers		
AFU_MAC.CFG.ICE0DID	0x00000000	OpenCAPI Function 0 Device ID & Vendor ID Register
AFU_MAC.CFG.ICE0SYID	0x0000002C	OpenCAPI Function 0 MMIO Subsystem ID & Subsystem Vendor ID Register
AFU_MAC.CFG.ICE1BARH	0x00010014	OpenCAPI Function 1 MMIO Base Upper Address 0 Register
AFU_MAC.CFG.ICE1BARL	0x00010010	OpenCAPI Function 1 MMIO Base Lower Address 0 Register
AFU_MAC.CFG.ICE1DID	0x00010000	OpenCAPI Function 1 Device ID & Vendor ID Register
AFU_MAC.CFG.ICE1MBIT	0x00010004	OpenCAPI Function 1 Memory Space Enable Register
AFU_MAC.CFG.ICE1SYID	0x0001002c	OpenCAPI Function 1 MMIO Subsystem ID & Subsystem Vendor ID Register
AFU_MAC.CFG.ICEAFUEN	0x0001050c	OpenCAPI AFU Control DVSEC AFU Enable and ID Register
AFU_MAC.CFG.ICEATAGB	0x0001051c	OpenCAPI AFU Control DVSEC acTAG Register
AFU_MAC.CFG.ICEATAGL	0x00010518	OpenCAPI AFU Control DVSEC acTAG Length Register
AFU_MAC.CFG.ICEFTAG	0x0001030c	OpenCAPI Function acTAG Register
AFU_MAC.CFG.ICEPIDB	0x00010514	OpenCAPI AFU Control DVSEC PASID Base Register
AFU_MAC.CFG.ICEPIDL	0x00010510	OpenCAPI AFU Control DVSEC PASID Length Register
AFU_MAC.CFG.ICERCAP	0x0000021c	OpenCAPI TLX Receive Template 31-0 Capabilities Register
AFU_MAC.CFG.ICERRATE	0x0000024c	OpenCAPI TLX Flit Receive Rate per Template 7-0 Capabilities Register
AFU_MAC.CFG.ICETCFG	0x00000224	OpenCAPI TLX Transmit Template Configurations Register
AFU_MAC.CFG.ICETRATE	0x0000026c	OpenCAPI TLX Flit Transmit Rate per Templates 7-0 Configurations Register
AFU_MAC.CFG.ICEVER	0x00000210	OpenCAPI Transport Layer Configuration Register
MFU Required Registers		
AFU_MAC.MMIO.ICE_REGS.ICECFG0	0x0801240c	Configuration Register 0
AFU_MAC.MMIO.ICE_REGS.ICECFG1	0x0801240d	Configuration Register 1
AFU_MAC.MMIO.ICE_REGS.ICECFG2	0x0801240e	Configuration Register 2

AFU_MAC.MMIO.ICE_REGS.ICEINTHLD0	0x08012410	Interrupt Handle Register 0
AFU_MAC.MMIO.ICE_REGS.ICEINTHLD1	0x08012411	Interrupt Handle Register 1
AFU_MAC.MMIO.ICE_REGS.ICEINTHLD2	0x08012412	Interrupt Handle Register 2
AFU_MAC.MMIO.ICE_REGS.ICEINTHLD3	0x08012413	Interrupt Handle Register 3

6.2.1. Detailed Register Descriptions

This section defines the detailed bit level definitions of the minimum required register set that shall be deployed in the MSL for operation in OpenPOWER Systems with an OMI3.1 memory interface.

Table 6.2. OpenCAPI Function 0 Device ID & Vendor ID Register

RTL Mnemonic			**AFU_MAC.CFG.ICE0DID**	
Name			OpenCAPI Function 0 Device ID & Vendor ID Register	
Reset Value			0x06361014	
Description			OpenCAPI Device ID and Vendor ID Register for Function 0	
Address Type			**Address**	
Configuration			0x00000000	
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:0	0x06361014	Reserved	RO	constant=0b00000110001101100001000000010100 This is the value that an OpenPOWER system recognizes as the ID for an OCMB (OpenCAPI Memory Buffer).

Table 6.3. OpenCAPI Function 0 MMIO Subsystem ID & Subsystem Vendor ID Register

RTL Mnemonic			**AFU_MAC.CFG.ICE0SYID**	
Name			OpenCAPI Function 0 MMIO Subsystem ID & Subsystem Vendor ID Register	
Reset Value			0x06361014	
Description			Function 0 Subsystem ID and Subsystem Vendor ID	
Address Type			**Address**	
Configuration			0x0000002c	
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:0	0x06361014	Reserved	RO	constant=0b00000110001101100001000000010100 This is the value that an OpenPOWER system recognizes as the ID for an OCMB (OpenCAPI Memory Buffer).

Table 6.4. OpenCAPI Function 1 MMIO Base Upper Address 0 Register

RTL Mnemonic	**AFU_MAC.CFG.ICE1BARH**
Name	OpenCAPI Function 1 MMIO Base Upper Address 0 Register
Reset Value	0xFFFFFFF8
Description	Indicates the MMIO Base Address for window 0, Function 1. This is used to access MMIO registers.
Address Type	**Address**
Configuration	0x00010014

Bit Range	Initial Value	Field Name	Field Type	Description
31:3	0x1fffffff	ICE1BARH_BAR_ADDRESS	RW	Upper 29 bits for Base Address for window. The window is 32GB in size so the lower 35 bits of the base address is zero. This field is disruptive to write during runtime.
2:0	0x0	Reserved	RO	Constant=0b000

Table 6.5. OpenCAPI Function 1 MMIO Base Lower Address 0 Register

RTL Mnemonic				AFU_MAC.CFG.ICE1BARL
Name				OpenCAPI Function 1 MMIO Base Lower Address 0 Register
Reset Value				0x00000004
Description				Indicates the MMIO Base Address for window 0, Function 1. This is used to access MMIO registers.
Address Type				**Address**
Configuration				0x00010010
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:4	0x0	Reserved	RO	constant=0b00000000000000000000000000000
3	0x0	ICE1BARL_PREFETCHABLE	RO	This field is 0 to indicate the window is not prefetchable. RO from OC config space.
2:1	0x2	ICE1BARL_TYPE	RO	This field is set to 0b10 to indicate 64-bit addressing. RO from OC config space.
0	0x0	ICE1BARL_ADDRESS_SPACE	RO	This field is set to 0 to indicate this is a Memory Address Space. RO from OC config space.

Table 6.6. OpenCAPI Function 0 Device ID & Vendor ID Register

RTL Mnemonic				AFU_MAC.CFG.ICE1DID
Name				OpenCAPI Function 0 Device ID & Vendor ID Register
Reset Value				0x06361014
Description				OpenCAPI Device ID and Vendor ID Register for Function 1
Address Type				**Address**
Configuration				0x00010000
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:0	0x06361014	Reserved	RO	constant=0b00000110001101100001000000010100

Table 6.7. OpenCAPI Function 1 Memory Space Enable Register

RTL Mnemonic				AFU_MAC.CFG.ICE1MBIT
Name				OpenCAPI Function 1 Memory Space Enable Register
Reset Value				0x00100000
Description				OpenCAPI Function 1 Memory Space Enable Register
Address Type				**Address**
Configuration				0x00010004
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:21	0x0	Reserved	RO	constant=0b00000000000
20	0x1	ICE1MBIT_CAPABILITIES_LIST	RO	Capabilities list pointer supported = 1
19:2	0x0	Reserved	RO	constant=0b0000000000000000

1	0x0	ICE1MBIT_MEMORY_SPACE	RW	Enables response to BAR memory accesses This field can be freely written during runtime.
0	0x0	Reserved	RO	constant=0b0

Table 6.8. OpenCAPI Function 1 MMIO Subsystem ID & Subsystem Vendor ID Register

RTL Mnemonic		AFU_MAC.CFG.ICE1SYID		
Name		OpenCAPI Function 1 MMIO Subsystem ID & Subsystem Vendor ID Register		
Reset Value		0x00000000		
Description		Function 1 Subsystem ID and Subsystem Vendor ID		
Address Type		**Address**		
Configuration		0x0001002c		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:0	0x0	Reserved	RO	constant=0b00000110001101100001000000010100

Table 6.9. OpenCAPI AFU Control DVSEC AFU Enable and ID Register

RTL Mnemonic		AFU_MAC.CFG.ICEAFUEN		
Name		OpenCAPI AFU Control DVSEC AFU Enable and ID Register		
Reset Value		0x00000000		
Description		This register contains AFU controls and PASID processing data.		
Address Type		**Address**		
Configuration		0x0001050c		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:28	0x0	ICEAFUEN_AFU_UNIQUE	RW	These bits are defined by each specific AFU implementation This field can be freely written during runtime.
27:26	0x0	Reserved	RO	constant=0b00
25	0x0	ICEAFUEN_FENCE_AFU	RW	Not supported in OCMB. This field is not disruptive to write during runtime.
24	0x0	ICEAFUEN_ENABLE_AFU	RW	Allows the AFU to send Commands and Data on the OpenCAPI link. Does not affect receiving commands or sending responses. This field is disruptive to write during runtime.
23	0x0	ICEAFUEN_RESET_AFU	WO	Not supported in OCMB. WO from OC config space. This field can be freely written during runtime.
22:21	0x0	Reserved	RO	constant=0b00
20	0x0	ICEAFUEN_TERMINATE_VALID	RMX	PASID Termination Valid. Not supported in OCMB. This bit will stay on for 15 CFG clock cycles (375 us @ 400MHz) before resetting. RWX from OC config space. This field can be freely written during runtime.
19:0	0x0	ICEAFUEN_ PASID_TERMINATION_VALUE	RW	PASID to terminate. Not supported in OCMB.

Table 6.10. OpenCAPI AFU Control DVSEC acTAG Register

RTL Mnemonic	AFU_MAC.CFG.ICEATAGB
Name	OpenCAPI AFU Control DVSEC acTAG Register
Reset Value	0x00000000

Description		This register contains acTag support information		
Address Type		**Address**		
Configuration		0x0001051c		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:12	0x0	Reserved	RO	constant=0b00000000000000000000
11:0	0x0	ICEATAGB_AFU_ACTAG_BASE	RW	This is the first acTag this AFU is allowed to use. This field is disruptive to write during runtime.

Table 6.11. OpenCAPI AFU Control DVSEC acTAG Length Register

RTL Mnemonic		**AFU_MAC.CFG.ICEATAGL**		
Name		OpenCAPI AFU Control DVSEC acTAG Length Register		
Reset Value		0x00000000		
Description		This register contains AFU acTag Length information		
Address Type		**Address**		
Configuration		0x00010518		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:28	0x0	Reserved	RO	constant=0b0000
27:16	0x0	ICEATAGL_ AFU_ACTAG_LENGTH_ENABLED	RW	Number of consecutive acTags this AFU is allowed to use This field is disruptive to write during runtime.
15:0	0x0	Reserved	RO	constant=0b0000000000000001

Table 6.12. OpenCAPI Function acTAG Register

RTL Mnemonic		**AFU_MAC.CFG.ICEFTAG**		
Name		OpenCAPI Function acTAG Register		
Reset Value		0x00000000		
Description		This register is set up by the host and indicates the base acTAG and number of acTAGs that can be used.		
Address Type		**Address**		
Configuration		0x0001030c		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:28	0x0	Reserved	RO	constant=0b0000
27:16	0x0	ICEFTAG_ACTAG_BASE	RW	Starting acTag that can be used by this Function. This field is disruptive to write during runtime.
15:12	0x0	Reserved	RO	constant=0b0000
11:0	0x0	ICEFTAG_ACTAG_LENGTH	RW	Number of consecutive acTags this Function is allowed to use. This field is disruptive to write during runtime.

Table 6.13. OpenCAPI AFU Control DVSEC PASID Base Register

RTL Mnemonic	**AFU_MAC.CFG.ICEPIDB**
Name	OpenCAPI AFU Control DVSEC PASID Base Register
Reset Value	0x00000000
Description	This register contains AFU metadata and PASID controls
Address Type	**Address**
Configuration	0x00010514

Bit Range	Initial Value	Field Name	Field Type	Description
31	0x0	Reserved	RO	constant=0b1
30	0x0	ICEPIDB_METADATA_ENABLED	RW	Metadata enabled. RW from OC config space This field is disruptive to write during runtime. Dial enums: METADATA_IS_NOT_ENABLED=>0b0 METADATA_IS_ENABLED=>0b1
29:27	0x0	ICEPIDB_HTRL	RW	Not used in OpenCAPI v3.x. Has no effect in OCMB. Reserved for use in OpenCAPI v4.0 devices. This field can be freely written during runtime.
26:20	0x0	Reserved	RO	constant=0b0000000
19:0	0x0	ICEPIDB_PASID_BASE	RW	First PASID this AFU is allowed to use. RW from OC config space This field is disruptive to write during runtime.

Table 6.14. OpenCAPI AFU Control DVSEC PASID Length Register

RTL Mnemonic				AFU_MAC.CFG.ICEPIDL
Name				OpenCAPI AFU Control DVSEC PASID Length Register
Reset Value				0x00000000
Description				This register contains AFU PASID controls
Address Type				**Address**
Configuration				0x00010510
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:13	0x0	Reserved	RO	constant=0b0000000000000000000
12:8	0x0	ICEPIDL_PASID_LENGTH_ENABLED	RW	Number of consecutive PASIDs this AFU is allowed to use = 2^pasid_length_enabled RW from OC config space Valid PASID range = pasid_base + ((2^pasid_length_enabled)-1) This field is disruptive to write during runtime.
7:0	0x0	Reserved	RO	constant=0b00000000

Table 6.15. OpenCAPI TLX Receive Template 31-0 Capabilities Register

RTL Mnemonic				AFU_MAC.CFG.ICERCAP
Name				OpenCAPI TLX Receive Template 31-0 Capabilities Register
Reset Value				0x00000000
Description				Indicates which receive templates OCMB supports. This design currently supports receive templates 4, 1 & 0
Address Type				**Address**
Configuration				0x0000021c
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:8	0x0	Reserved	RO	constant=0b000000000000000000000000
7	0x0	ICERCAP_TEMPLATE_7	RO	When set indicates that Template 7 is supported. This field is disruptive to write during runtime.
6:5	0x0	Reserved	RO	constant=0b00
4	0x0	ICERCAP_TEMPLATE_4	RO	When set indicates that Template 4 is supported. This field is disruptive to write during runtime.
3:2	0x0	Reserved	RO	constant=0b00
1	0x0	ICERCAP_TEMPLATE_1	RO	When set indicates that Template 1 is supported. This field is disruptive to write during runtime.

0	0x0	ICERCAP_TEMPLATE_0	RO	When set indicates that Template 0 is supported. This field is disruptive to write during runtime.

Table 6.16. OpenCAPI TLX Flit Receive Rate per Template 7-0 Capabilities Register

RTL Mnemonic			AFU_MAC.CFG.ICERRATE	
Name			OpenCAPI TLX Flit Receive Rate per Template 7-0 Capabilities Register	
Reset Value			0x00000000	
Description			Indicates the receive rate that can be supported for templates 7-0 by the design.	
Address Type			**Address**	
Configuration			0x0000024c	
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:28	0x0	ICERRATE_TEMPLATE_7	RO	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 7. A value of zero indicates that a control flit can be sent in the following flit cycle. This field can be freely written during runtime.
27:20	0x0	Reserved	RO	constant=0b00000000
19:16	0x0	ICERRATE_TEMPLATE_4	RO	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 4. A value of zero indicates that a control flit can be sent in the following flit cycle. This field can be freely written during runtime.
15:8	0x0	Reserved	RO	constant=0b00000000
7:4	0x0	ICERRATE_TEMPLATE_1	RO	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 1. A value of zero indicates that a control flit can be sent in the following flit cycle. This field can be freely written during runtime.
3:0	0x0	ICERRATE_TEMPLATE_0	RO	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 0. A value of zero indicates that a control flit can be sent in the following flit cycle. This field can be freely written during runtime.

Table 6.17. OpenCAPI TLX Transmit Template Configurations Register

RTL Mnemonic			AFU_MAC.CFG.ICETCFG	
Name			OpenCAPI TLX Transmit Template Configurations Register	
Reset Value			0x00000001	
Description			The host must set this to indicate which Transmit Templates this design should use. Note that Templates 0, 1 & 5 are supported by this design. Template 0 must always be supported.	
Address Type			**Address**	
Configuration			0x00000224	
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31	0x0	ICETCFG_TEMPLATE_31	RW	When set indicates that Template 31 is supported. This field is disruptive to write during runtime.
30	0x0	ICETCFG_TEMPLATE_30	RW	When set indicates that Template 30 is supported. This field is disruptive to write during runtime.
29	0x0	ICETCFG_TEMPLATE_29	RW	When set indicates that Template 29 is supported. This field is disruptive to write during runtime.
28	0x0	ICETCFG_TEMPLATE_28	RW	When set indicates that Template 28 is supported. This field is disruptive to write during runtime.

27	0x0	ICETCFG_TEMPLATE_27	RW	When set indicates that Template 27 is supported. This field is disruptive to write during runtime.
26	0x0	ICETCFG_TEMPLATE_26	RW	When set indicates that Template 26 is supported. This field is disruptive to write during runtime.
25	0x0	ICETCFG_TEMPLATE_25	RW	When set indicates that Template 25 is supported. This field is disruptive to write during runtime.
24	0x0	ICETCFG_TEMPLATE_24	RW	When set indicates that Template 24 is supported. This field is disruptive to write during runtime.
23	0x0	ICETCFG_TEMPLATE_23	RW	When set indicates that Template 23 is supported. This field is disruptive to write during runtime.
22	0x0	ICETCFG_TEMPLATE_22	RW	When set indicates that Template 22 is supported. This field is disruptive to write during runtime.
21	0x0	ICETCFG_TEMPLATE_21	RW	When set indicates that Template 21 is supported. This field is disruptive to write during runtime.
20	0x0	ICETCFG_TEMPLATE_20	RW	When set indicates that Template 20 is supported. This field is disruptive to write during runtime.
19	0x0	ICETCFG_TEMPLATE_19	RW	When set indicates that Template 19 is supported. This field is disruptive to write during runtime.
18	0x0	ICETCFG_TEMPLATE_18	RW	When set indicates that Template 18 is supported. This field is disruptive to write during runtime.
17	0x0	ICETCFG_TEMPLATE_17	RW	When set indicates that Template 17 is supported. This field is disruptive to write during runtime.
16	0x0	ICETCFG_TEMPLATE_16	RW	When set indicates that Template 16 is supported. This field is disruptive to write during runtime.
15	0x0	ICETCFG_TEMPLATE_15	RW	When set indicates that Template 14 is supported. This field is disruptive to write during runtime.
14	0x0	ICETCFG_TEMPLATE_14	RW	When set indicates that Template 14 is supported. This field is disruptive to write during runtime.
13	0x0	ICETCFG_TEMPLATE_13	RW	When set indicates that Template 13 is supported. This field is disruptive to write during runtime.
12	0x0	ICETCFG_TEMPLATE_12	RW	When set indicates that Template 12 is supported. This field is disruptive to write during runtime.
11	0x0	ICETCFG_TEMPLATE_11	RW	When set indicates that Template 11 is supported. This field is disruptive to write during runtime.
10	0x0	ICETCFG_TEMPLATE_10	RW	When set indicates that Template 10 is supported. This field is disruptive to write during runtime.
9	0x0	ICETCFG_TEMPLATE_9	RW	When set indicates that Template 9 is supported. This field is disruptive to write during runtime.
8	0x0	ICETCFG_TEMPLATE_8	RW	When set indicates that Template 8 is supported. This field is disruptive to write during runtime.
7	0x0	ICETCFG_TEMPLATE_7	RW	When set indicates that Template 7 is supported. This field is disruptive to write during runtime.
6	0x0	ICETCFG_TEMPLATE_6	RW	When set indicates that Template 6 is supported. This field is disruptive to write during runtime.
5	0x0	ICETCFG_TEMPLATE_5	RW	When set indicates that Template 5 is supported. This field is disruptive to write during runtime.
4	0x0	ICETCFG_TEMPLATE_4	RW	When set indicates that Template 4 is supported. This field is disruptive to write during runtime.
3	0x0	ICETCFG_TEMPLATE_3	RW	When set indicates that Template 3 is supported. This field is disruptive to write during runtime.
2	0x0	ICETCFG_TEMPLATE_2	RW	When set indicates that Template 2 is supported. This field is disruptive to write during runtime.
1	0x0	ICETCFG_TEMPLATE_1	RW	When set indicates that Template 1 is supported. This field is disruptive to write during runtime.

0	0x0	ICETCFG_TEMPLATE_0	RW	When set indicates that Template 0 is supported. This field is disruptive to write during runtime.

Table 6.18. OpenCAPI TLX Flit Receive Rate per Template 7-0 Capabilities Register

RTL Mnemonic		AFU_MAC.CFG.ICETRATE		
Name		OpenCAPI TLX Flit Transmit Rate per Templates 7-0 Configurations Register		
Reset Value		0x00000000		
Description		Indicates the transmit rate that can be supported for templates 7-0 for the design		
Address Type		**Address**		
Configuration		0x0000026c		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:28	0x0	ICERRATE_TEMPLATE_7	RW	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 7. A value of zero indicates that a control flit can be sent in the following flit cycle. This field can be freely written during runtime.
27:24	0x0	ICETRATE_TEMPLATE_6	RW	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 6. A value of zero indicates that a control flit can be sent in the following flit cycle. This field can be freely written during runtime.
23:20	0x0	ICETRATE_TEMPLATE_5	RW	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 5. A value of zero indicates that a control flit can be sent in the following flit cycle. This field is disruptive to write during runtime.
19:16	0x0	ICETRATE_TEMPLATE_4	RW	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 4. A value of zero indicates that a control flit can be sent in the following flit cycle. This field can be freely written during runtime.
15:12	0x0	ICETRATE_TEMPLATE_3	RW	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 3. A value of zero indicates that a control flit can be sent in the following flit cycle. This field can be freely written during runtime.
11:8	0x0	ICERRATE_TEMPLATE_2	RW	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 2. A value of zero indicates that a control flit can be sent in the following flit cycle. This field can be freely written during runtime.
7:4	0x0	ICERRATE_TEMPLATE_1	RW	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 1. A value of zero indicates that a control flit can be sent in the following flit cycle. This field is disruptive to write during runtime.
3:0	0x0	ICERRATE_TEMPLATE_0	RW	Indicates the size of gap (measured in flits) before the next control flit can be sent for Template 0. A value of zero indicates that a control flit can be sent in the following flit cycle. This field is disruptive to write during runtime.

Table 6.19. OpenCAPI Transport Layer Configuration Register

RTL Mnemonic	AFU_MAC.CFG.ICEVER
Name	OpenCAPI Transport Layer Configuration Register

Reset Value		0x31000000		
Description		This register is configured by the host to indicate the level of OpenCAPI TL specification OCMB must use.		
Address Type		**Address**		
Configuration		0x00000210		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
31:24	0x31	ICEVER_TL_ MAJOR_VERSION_CONFIGURATION	RW	Indicates, x, the major version of OpenCAPI TL specification OCMB must use (as in version x.y. For example v3.1) This field can be freely written during runtime.
23:16	0x0	ICEVER_TL_ MINOR_VERSION_CONFIGURATION	RW	Indicates, y, the minor version of OpenCAPI TL specification OCMB must use (as in version x.y. For example v3.1) This field is disruptive to write during runtime.
15:8	0x0	Reserved	RO	constant=0b00000000
7:4	0x0	ICEVER_LONG_BACK_OFF_TIMER	RW	Long back off timer for long back-off retry responses. Not supported in OCMB. This field can be freely written during runtime.
3:0	0x0	ICEVER_SHORT_BACK_OFF_TIMER	RW	Short back off timer for short back-off retry responses. This field is disruptive to write during runtime.

Table 6.20. ICE Configuration Register 0

RTL Mnemonic		**AFU_MAC.MMIO.ICE_REGS.ICECFG0**		
Name		ICE Configuration Register 0		
Reset Value		0x0000000000000000		
Description		NA		
Address Type		**Address**		
Configuration		0x0801240c		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:63	0x0	ICECFG0_?	RW	Default register used for operations targeting unimplemented registers. This register's fields and definition are MFU design dependent and outside the scope of this document.

Table 6.21. ICE Configuration Register 1

RTL Mnemonic		**AFU_MAC.MMIO.ICE_REGS.ICECFG1**		
Name		ICE Configuration Register 1		
Reset Value		0x0000040000450000		
Description		NA		
Address Type		**Address**		
Configuration		0x0801240c		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0	0x0	ICECFG1_TRAP_CLEAR	RW	when set, all trap and hold registers will be cleared
1	0x0	ICECFG1_INJECT_XSTOP	RW	When set will inject a channel checkstop error to be reported to the host via the interrupt interface Interrupts are generated on the rising edge of an error. To inject a second interrupt of this type the bit will need to be cleared and then set again.

2	0x0	ICECFG1_INJECT_MCHK	RW	When set will inject aapplication interrupt to be reported to the host via the interrupt interface Interrupts are generated on the rising edge of an error. To inject a second interrupt of this type the bit will need to be cleared and then set again.
3	0x0	ICECFG1_TRAP_UPDATE	RW	when set, trap updates every cycle and does not freeze on error
4	0x0	ICECFG1_HOLD_ACUM	RW	when set, hold reg acumulates on error and does not freeze on fir error out
5:15	0x0	Reserved	RO	Constant 0b00000000000
16:20	0x0	ICECFG1_EDPL_RESERVED0	RW	EDPL Reserved, Constant 0b0000
21	0x1	ICECFG1_EDPL_ENABLE	RW	EDPL Enable
22	0x0	ICECFG1_ EDPL_MAX_COUNTER_RESET	RW	EDPL Max Counter Reset
23:31	0x0	ICECFG1_EDPL_RESERVED1	RW	EDPL reserved
32:39	0x0	ICECFG1_EDPL_INJECT_PER_LANE	RW	EDPL Inject Per lane
40	0x0	ICECFG1_EDPL_RESERVED2	RW	EDPL reserved
41:43	0x4	ICECFG1_ EDPL_ERROR_THRESHOLD	RW	EDPL Error Threshold
44:47	0x5	ICECFG1_EDPL_TIME_WINDOW	RW	EDPL Time Window
48:51	0x0	ICECFG1_INTRP_CMDFLAG_0	RW	ICE Interrupt handle 0 command flag. Fills the cmd_flag portion of the intrp_re_q command found in section 2.3 of the OpenCAPI TL 3.1 Specification. Used for Channel Checkstop Interrupt Request.
52:55	0x0	ICECFG1_INTRP_CMDFLAG_1	RW	ICE Interrupt handle 1 command flag Fills the 4-bit cmd_flag portion of the intrp_req command found in section 2.3 of the OpenCAPI TL 3.1 Specification. Used for Recoverable Attention Interrupt Request.
56:59	0x0	ICECFG1_INTRP_CMDFLAG_2	RW	ICE Interrupt handle 2 command flag Fills the cmd_flag portion of the intrp_req command found in section 2.3 of the OpenCAPI TL 3.1 Specification. Used for Special Attention Interrupt Request.
60:63	0x0	ICECFG1_INTRP_CMDFLAG_3	RW	ICE Interrupt handle 3 command flag Fills the cmd_flag portion of the intrp_req command found in section 2.3 of the OpenCAPI TL 3.1 Specification. Used for Application Interrupt Request.

Table 6.22. ICE Configuration Register 2

RTL Mnemonic		**AFU_MAC.MMIO.ICE_REGS.ICECFG2**		
Name		ICE Configuration Register 2		
Reset Value		0x0000000000000000		
Description		NA		
Address Type		**Address**		
Configuration		0x0801240e		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:31	0x0	ICECFG2_ECID	RO	Unique identifier for this design.
32:63	0x0	ICECFG2_DEVICE_ID_REG	RO	Device Id information for the design. For example... 32:35 = Major Release 36:39 = Minor Release 44:47 = Location Code 56:63 = Chip ID

Table 6.23. ICE Interrupt Handle Register 0

RTL Mnemonic		AFU_MAC.MMIO.ICE_REGS.ICEINTHLD0		
Name		ICE Interrupt Handle Register 0		
Reset Value		0x0000000000000001		
Description		NA		
Address Type		**Address**		
Configuration		0x08012410		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:63	0x00000000 00000001	ICEINTHLD0_TLXT_INTHLD_0_REG	RW	This field is disruptive to write during runtime. ICE Interrupt Handle Register 0 Used for Channel Checkstop Interrupt Fills the obj_handle portion of the intrp_re_q command found in section 2.3 of the OpenCAPI TL 3.1 Specification.

Table 6.24. ICE Interrupt Handle Register 1

RTL Mnemonic		AFU_MAC.MMIO.ICE_REGS.ICEINTHLD1		
Name		ICE Interrupt Handle Register 1		
Reset Value		0x0000000000000002		
Description		NA		
Address Type		**Address**		
Configuration		0x08012411		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:63	0x00000000 00000002	ICEINTHLD1_TLXT_INTHLD_1_REG	RW	This field is disruptive to write during runtime. ICE Interrupt Handle Register 1 Used for Recover-able Attn Interrupt Fills the obj_handle portion of the intrp_req command found in section 2.3 of the OpenCAPI TL 3.1 Specification.

Table 6.25. ICE Interrupt Handle Register 2

RTL Mnemonic		AFU_MAC.MMIO.ICE_REGS.ICEINTHLD2		
Name		ICE Interrupt Handle Register 2		
Reset Value		0x0000000000000003		
Description		NA		
Address Type		**Address**		
Configuration		0x08012412		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:63	0x00000000 00000003	ICEINTHLD2_TLXT_INTHLD_2_REG	RW	This field is disruptive to write during runtime. ICE Interrupt Handle Register 2 Used for Special Attn Interrupt Fills the obj_handle portion of the intrp_req command found in section 2.3 of the OpenCAPI TL 3.1 Specification.

Table 6.26. ICE Interrupt Handle Register 3

RTL Mnemonic	AFU_MAC.MMIO.ICE_REGS.ICEINTHLD3
Name	ICE Interrupt Handle Register 3
Reset Value	0x0000000000000004

Description		NA		
Address Type		**Address**		
Configuration		0x08012413		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:63	0x00000000 00000004	ICEINTHLD3_TLXT_INTHLD_3_REG	RW	This field is disruptive to write during runtime. ICE Interrupt Handle Register 3 Used for Application Interrupt Fills the obj_handle portion of the intrp_req command found in section 2.3 of the OpenCAPI TL 3.1 Specification.

Table 6.27. ICE Trap Register 0

RTL Mnemonic		**AFU_MAC.MMIO.ICE_REGS.ICETRAP0**		
Name		ICE Trap Register 0		
Reset Value		0x0000000000000000		
Description		NA		
Address Type		**Address**		
Configuration		0x08012424		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:63	0x0	ICETRAP0_DEBUG_BUS	ROX	TLX Parser Trace 0 = dlx_tlx_link_up 1 = dlx_tlx_flit_crc_err 2 = dlx_tlx_flit_valid 3 = data_rx 4:7 = dlx_tlx_flit(451 downto 448) - this is the data runlength 8:13 = dlx_tlx_flit(465 downto 460) - this is the template type 14:17 = cflit(451 downto 448) - runlength in use on output of input fifo 18:23 = cflit(465 downto 460) template in use on output of input fifo 24 = dl_full - the input fifo is full 25 = start_ctl - start to parse a control flit 26:29 = parse_st - counts thru possible slots in the template looking for cmd packets 30:37 = packet_q(7:0) - opcode of parser output 38:40 = packet_q(111 downto 109) - length (pl or dl) of parser packet output 41 = dcp_we - write enable strobe for data (command data or config data) 42:47 = cmdd_w_crc_q - count of cmd data that is in the cmd data fifo and has good crc (units of 64B) 48 = cfg_data_empty - fifo for config write data has nothing with good crc in it 49 = cmd write - write a command packet into the command fifo 50 = cfg_write - write a config command into the config fifo 51 = ds_write - write an entry into the data steering fifo 52:59 = ds_din - data steering fifo entry contains cmd/cfg flag and address offset if cfg. 60 = resp_write - write an intrp_resp packet to the resp fifo 61 = resp_cred_ok - there is enough credit for tlx to send a response to afu 62 = cmd_cred_ok - there is enough credit for tlx to send a command to afu 63 = cfg_cred_ok - there is enough credit for tlx to send a config command to the cfg

Table 6.28. ICE Trap Register 2

RTL Mnemonic	**AFU_MAC.MMIO.ICE_REGS.ICETRAP2**
Name	ICE Trap Register 2
Reset Value	0x0000000000000000
Description	NA
Address Type	**Address**

Configuration		0x08012426		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:63	0x0	ICETRAP2_DEBUG_BUS	ROX	MMIO/error Reporting Trace 0 = xlate_mmio_cmd_val 1 = xlate_mmi_cmd_rd 2:17 = xlate_mmio_cmd_tag 18 = cmd_pop_q 19 = cmd_data_out(0) 20:35 = cmd_data_out(51:36) 36:53 = sc_addr_v 54:56 = cmd_addr_offset(2:0) 57:58 = intrp_req_chan_xstop 59:60 = intrp_req_app_intrp

Table 6.29. ICE Trap Register 3

RTL Mnemonic		**AFU_MAC.MMIO.ICE_REGS.ICETRAP3**		
Name		ICE Trap Register 3		
Reset Value		0x0000000000000000		
Description		NA		
Address Type		**Address**		
Configuration		0x08012427		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:63	0x0	ICETRAP3_DEBUG_BUS	ROX	TLX Framer Trace 0 = rsp_valid_d1 1 = injct_cfg_rsp 2:4 = rsp_packet_reg(2:0) 5:20 = rsp_packet_reg(23:8) 21 = valid_rsp_leaving_fifo 22:24 = vc0_fifo_output_reg(2:0) 25:40 = vc0_fifo_output_reg(23:8) 41:43 = rsp_tmpl 44 = rsp_cf_was_taken 45:48 = rcf_df_count_last 49:52 = meta_run_length 53:56 = framer_run_length 57 = metadata_enabled 58:60 = meta_packet_ptr(2:0) 61:63 = meta_packet_ptr(2:0)

Table 6.30. ICE Trap Register 5

RTL Mnemonic		**AFU_MAC.MMIO.ICE_REGS.ICETRAP5**		
Name		ICE Trap Register 5		
Reset Value		0x0000000000000000		
Description		NA		
Address Type		**Address**		
Configuration		0x08012429		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:31	0x0	ICETRAP5_DLX_DEBUG_REG04	ROX	DLX Debug Reg 04
32:63	0x0	ICETRAP5_DLX_DEBUG_REG05	ROX	DLX debug Reg 05

Table 6.31. ICE Trap Register 6

RTL Mnemonic		**AFU_MAC.MMIO.ICE_REGS.ICETRAP6**		
Name		ICE Trap Register 6		
Reset Value		0x0000000000000000		
Description		NA		
Address Type		**Address**		
Configuration		0x0801242a		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**

0:31	0x0	ICETRAP6_DLX_DEBUG_REG06	ROX	DLX Debug Reg06
32:63	0x0	ICETRAP6_DLX_DEBUG_REG07	ROX	DLX Debug Reg07

Table 6.32. ICE Trap Register 8

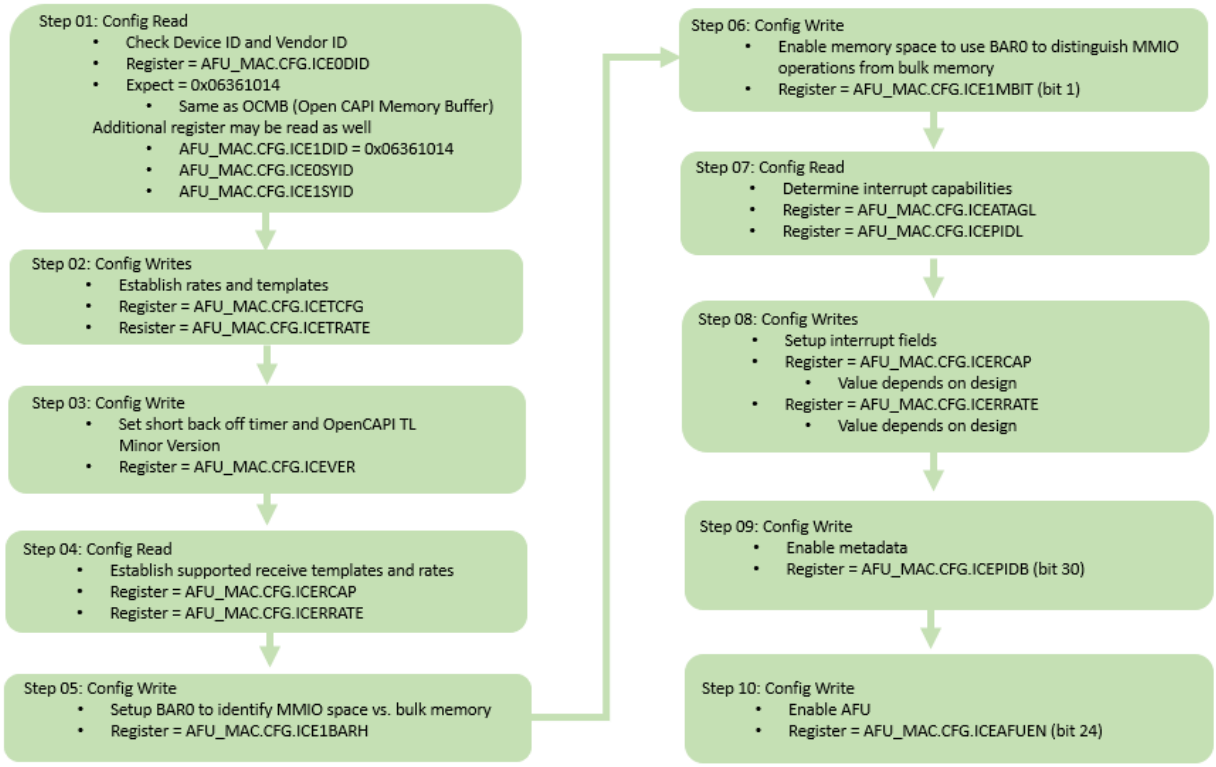
RTL Mnemonic		AFU_MAC.MMIO.ICE_REGS.ICETRAP8		
Name		ICE Trap Register 8		
Reset Value		0x0000000000000000		
Description		NA		
Address Type		**Address**		
Configuration		0x0801242c		
Bit Range	**Initial Value**	**Field Name**	**Field Type**	**Description**
0:63	0x0	ICETRAP8_DEBUG_BUS	ROX	The debug bus register fields and definition are MFU dependent and outside the scope of this document. Refer to the example design Genini/ICE for example guidance.

6.3. Configuration Register Initialization

After the OMI link is trained and the checked for errors, the OMI configuration is performed. To understand where this step occurs in the initialization flow, refer to the initialization flow in chapter eight.

The following figure depicts the configuration register initialization flow.

Figure 6.1. OMI Configuration Register Initialization Flow



7. Meta Bit Requirements

This chapter describes the minimum metabit requirements when deploying an MFU connected to the OpenPOWER Memory Bus (OPMB) as attached to an OpenCAPI enabled OpenPower system. Some systems may require usage of additional metabits above the minimum stated here. Therefore, when designin an MFU, it is advisable to refer to the appropriate system level documentation for correct implementation of all required meta bits.

At a minimum the MFU will encode the metabits to distinguish between a failure on a downstream write transfer from the host or a failure on the read path from the attached media.

Encoding metabits for a read or write failure is called a SUE encoding. Two of the six metabits are deployed for SUE encoding, metabit 2 and metabit 5.

Downstream write failures shall be encoded as a write failure SUE anywhere in the path where a failure occurs starting with the data parity check on the downstream interface from the MSL to the MFU at a minimum. The MFU must also maintain an encoded SUE received in the metabits from the host on the MSL on the downstream MSL interface.

Upstream read failures shall be encoded as a media read failure SUE anywhere in the path where a failure occurs ending with a data integrity check (ECC, parity, etc.) on the read data prior to transfer from the MFU to the MSL. Write path SUE must be maintained with the data and should be present on the read transaction.

7.1. Meta Bit Definitions

This section defines the meta bit encodings.

Table 7.1. OpenPOWER System SUE Meta Bit Encodings

Condition	Write Path Meta(2,5)	Read No-Media Error Meta(2,5)	Read Media Error Meta(2,5)	Description
Good data from MSL Meta Values, both 0	(0,0)	(0,0)	(0,1)	Good data from the host should always have matching meta bit values. The meta bit values are maintained if there is no read error from the media thus returning the same values back to the host. On a read fail from the media the meta bits are always set to Meta(2,5) = (0,1).
Good data from MSL Meta Values, both 1	(1,1)	(1,1)	(0,1)	Good data from the host should always have matching meta bit values. The meta bit values are maintained if there is no read error from the media thus returning the same values back to the host. On a read fail from the media the meta bits are always set to Meta(2,5) = (0,1)
Known-bad write data from MSL	(0,1) or (1,0)	(1,0)	(0,1)	Known bad data from the host has mismtaching meta bit values. These values will be stored in the media. When the data is retreived from the media and the meta bits indicate stored bad data the meta bits must always be Meta(2,5)=1,0 even if they were stored as 0,1. If the data is read from the media and there is a fail on the read the meta bits will be encoded as Meta(2,5)=0,1. The reasoning for this is to differentiate if the data was corrupted in the write path from host to the media or during a read from the media to the host. If both the data

				was corrupted on the write and a read back of the poisoned data was erroneous then the read fail indication takes priority.
Uncorrectable error on write	Any value UE Decoded	(1,0)	(0,1)	Similar to when known bad data from the host is presented with mismtaching meta bit values. In this case the write data from the TLX has bad parity or uncorrectable ECC encoding. In addition, data could be corrupted somwhere in the MFU prior to writing out to the media. In this case the MFU should mark the data with META(2,5)=(1,0) prior to writing to the media. When the data is retreived from the media and the meta bits indicate stored bad data the meta bits must always be Meta(2,5)=1,0. If the data is read from the media and there is a fail on the read the meta bits will be encoded as Meta(2,5)=0,1. The reasoning for this is to differentiate if the data was corrupted in the write path from host to the media or during a read from the media to the host. If both the data was corrupted on the write and a read back of the poisoned data was erroneous then the read fail indication takes priority.

8. Firmware Interaction

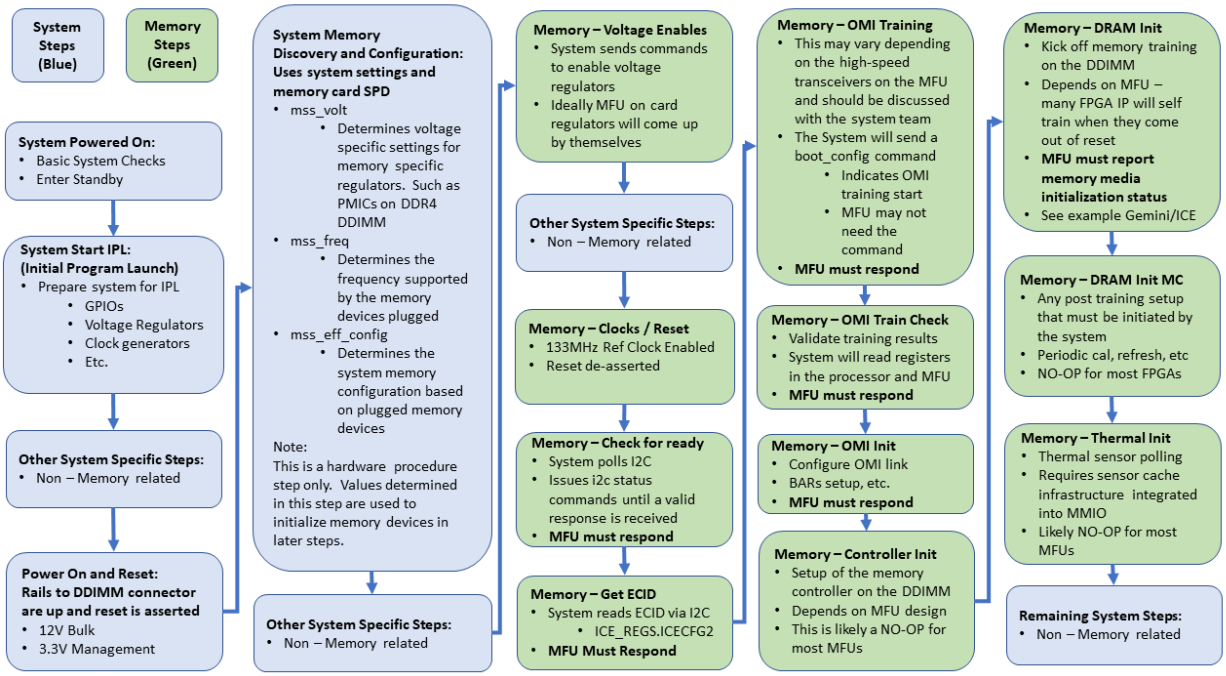
This chapter describes the requirements for interaction with the host firmware in most OMI enabled OpenPOWER systems.

It is outside the scope of this document to discuss every detailed interaction of the host level firmware and software with the deployed MFU as each MFU will vary in its functionality and such variance will likely impact the system software design. However, there are key interactions between the system firmware and the deployed MFU to achieve the basics of allowing the OpenPOWER OMI system to understand what type of memory device is available and how to initialize the device and map its available memory space.

8.1. Firmware Initialization Flow

There are a wide range of potential applications which may deploy an MFU attached to the OPMB MSL. Variants of the MFU may require specific updates to the firmware controlled hardware initialization flow. This section describes the basic high level flow expected to initialize the MFU. This document covers MFUs connected to an OMI channel in an OpenPOWER system, other deployments of OMI and MFU are outside the scope of this document. There are OpenPOWER system variations and it is advised that developers obtain the laters initialization flow from their OpenPOWER representative. In addition, some MFUs may require modification to the initialization flow to support specific features, again, it is adivsed that the developers contact their OpenPOWER representative for support. This document covers only the minimum requirements to intialize an MFU on the OMI channel in an OpenPOWER system.

Figure 8.1. High Level Firmware Initialization Flow



9. Development, Prototyping and Bring-Up Insights

This chapter provides insights and some level of guidance on getting started with development, prototyping and bring-up of your OMI MFU design.

No system required! Don't worry, you don't need a lot of hardware to get going. In fact a RaspberryPi and a reasonable FPGA card can get you going. In fact, that's what we started with when developing the Gemini / ICE deign found on Github (see section 1.3). Now it's even easier as most of the tools you will need to ensure compatibility of your design with an OpenPOWER system are available through the OpenPOWER Foundation via the OpenPOWER Toolkit Memtools and through the OpenCAPI consortium.

Depending upon the state of MFU development the OpenPOWER ecosystem can enable various levels of the prototyping process. By no means is deploying an MFU into an OpenPOWER system an all or nothing process. The OpenPOWER Ecosystem enables memory development tools and actual system level hardware procedures to run on stand alone MFU boards. This allows for low cost early development work without the need of a OpenPOWER OMI enabled system. In fact, the example design available on GitHub (Gemini/Ice in section 1.3 of this document) was developed and tested in a stand alone environment before any OMI systems were available. The example design was also used as a vehicle to test OMI Memory initialization procedures before real system hardware was available. The OpenPOWER Memory Development ecosystem was designed with an understanding that not all components of the design, including both hardware and software would be understood at the beginning and will require an iterative design process. From early prototyping on low cost test benches to final product integration and qualification, the OpenPOWER Ecosystem has it covered.

Appendix A. OpenPOWER Foundation overview

The OpenPOWER Foundation was founded in 2013 as an open technical membership organization that will enable data centers to rethink their approach to technology. Member companies are enabled to customize POWER CPU processors and system platforms for optimization and innovation for their business needs. These innovations include custom systems for large or warehouse scale data centers, workload acceleration through GPU, FPGA or advanced I/O, platform optimization for SW appliances, or advanced hardware technology exploitation. OpenPOWER members are actively pursing all of these innovations and more and welcome all parties to join in moving the state of the art of OpenPOWER systems design forward.

To learn more about the OpenPOWER Foundation, visit the organization website at openpowerfoundation.org.

A.1. Foundation documentation

Key foundation documents include:

- *Bylaws of OpenPOWER Foundation*
- *OpenPOWER Foundation Intellectual Property Rights (IPR) Policy*
- *OpenPOWER Foundation Membership Agreement*
- *OpenPOWER Anti-Trust Guidelines*

More information about the foundation governance can be found at openpowerfoundation.org/about-us/governance.

A.2. Technical resources

Development resouces fall into the following general categories:

- Technical Steering Committee
- Foundation work groups
- OpenPOWER Ready documentation, products, and certification criteria
- Resource Catalog

  To find all OpenPOWER resources of the following types, select the specificied combination of **Resource Type**/**Main Category**/**Sub-category** in the Resource Catalog:

Specifications	`Developer Resources` / `OpenPOWER Documents` / `Specifications`
Work Group Notes	`Developer Resources` / `OpenPOWER Documents` / `Work Group Notes`

Cloud development virtual machines	`Developer Resources` / `Software Developer Cloud Resources` / <empty>
Developer Tools	`Developer Resources` / `Developer Tools` / <empty>

Note

Use the **Search** field to focus your search using key words or phrases for specific resources.

A.3. Contact the foundation

To learn more about the OpenPOWER Foundation, please use the following contact points:

- General information -- `<info@openpowerfoundation.org>`
- Membership -- `<membership@openpowerfoundation.org>`
- Technical Work Groups and projects -- `<tsc-chair@openpowerfoundation.org>`
- Events and other activities -- `<admin@openpowerfoundation.org>`
- Press/Analysts -- `<press@openpowerfoundation.org>`

More contact information can be found at openpowerfoundation.org/get-involved/contact-us.

Appendix B. Glossary

architecture	A detailed specification of requirements for a processor or computer system. It does not specify details of how the processor or computer system must be implemented; instead it provides a template for a family of compatible implementations.
big endian	A byte-ordering method in memory where the address n of a word corresponds to the most-significant byte. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most-significant byte. See little endian.
cache	High-speed memory close to a processor. A cache usually contains recently-accessed data or instructions, but certain cache-control instructions can lock, evict, or otherwise modify the caching of data or instructions.
caching inhibited	A memory update policy in which the cache is bypassed, and the load or store is performed to or from system memory. A page of storage is considered caching inhibited when the 'I' bit has a value of '1' in the page table. Data located in caching inhibited pages cannot be cached at any memory hierarchy that is not visible to all processors and devices in the system. Stores must update the memory hierarchy to a level that is visible to all processors and devices in the system.
CAIA	Coherent Accelerator Interface Architecture. Defines an architecture for loosely coupled coherent accelerators. The Coherent Accelerator Interface Architecture provides a basis for the development of accelerators coherently connected to a POWER processor.
coherence	Refers to memory and cache coherence. The correct ordering of stores to a memory address, and the enforcement of any required cache write-backs during accesses to that memory address. Cache coherence is implemented by a hardware snoop (or inquire) method, which compares the memory addresses of a load request with all cached copies of the data at that address. If a cache contains a modified copy of the requested data, the modified data is written back to memory before the pending load request is serviced.
CRC	Cyclic redundancy check.
DAR	Data Address Register.
DMA	Direct memory access. A technique for using a special-purpose controller to generate the source and destination addresses for a memory or I/O transfer.
DMI	Differential Memory Interface.
DS	Down Stream - Indicating direction of command and data flow from DMI to OPMB
EA	Effective address. An address generated or used by a program to reference memory. A memory-management unit translates an effective address to a virtual address, which it then translates to a real address (RA) that accesses real (physical) memory. The maximum size of the effective-address space is 2^{64} bytes.
ECC	Error correction code. A code appended to a data block that can detect and correct bit errors within the block.
ERAT	Effective-to-real-address translation, or a buffer or table that contains such translations, or a table entry that contains such a translation.
ESID	Effective segment ID.
exception	An error, unusual condition, or external signal that can alter a status bit and will cause a corresponding interrupt, if the interrupt is enabled. See interrupt.
fetch	Retrieving instructions from either the cache or system memory and placing them into the instruction queue.
guarded	Prevented from responding to speculative loads and instruction fetches. The operating system typically implements guarding, for example, on all I/O devices.
HAUR	Hypervisor accelerator utilization record.
HCA	Hot/cold assist.
hypervisor	A control (or virtualization) layer between hardware and the operating system. It allocates resources, reserves resources, and protects resources among (for example) sets of AFUs that might be running under different operating systems.

implementation	A particular processor that conforms to the architecture but might differ from other architecture-compliant implementations for example in design, feature set, and implementation of optional features.
INT	Interrupt. A change in machine state in response to an exception.
ISA	Instruction set architecture.
ISN	Interrupt source number.
IVT	Interrupt vector table.
IVTE	Interrupt vector table entry.
KB	Kilobyte.
LA	A local storage (LS) address of a PSL list. It is used as a parameter in a PSL command.
LISN	Logical interrupt source number.
little endian	A byte-ordering method in memory where the address *n* of a word corresponds to the least-significant byte. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the most-significant byte. See big endian.
LISN	Logical interrupt source number.
LPAR	Logical partitioning. A function of an operating system that enables the creation of logical partitions.
LPC	Lowest point of coherency.
LPID	logical-partition identity.
LSb	Least-significant bit. The bit of least value in an address, register, data element, or instruction encoding.
main storage	The effective-address space. It consists physically of real memory (whatever is external to the memory-interface controller), local storage, memory-mapped registers and arrays, memory-mapped I/O devices, and pages of virtual memory that reside on disk. It does not include caches or execution-unit register files.
mask	A pattern of bits used to accept or reject bit patterns in another set of data. Hardware interrupts are enabled and disabled by setting or clearing a string of bits, with each interrupt assigned a bit position in a mask register.
MB	Megabyte.
memory coherency	An aspect of caching in which it is ensured that an accurate view of memory is provided to all devices that share system memory.
memory mapped	Mapped into the Coherent Attached Accelerator’s addressable-memory space. Registers, local storage (LS), I/O devices, and other readable or writable storage can be memory-mapped. System software does the mapping.
MFU	Memory Function Unit
MMIO	Memory-mapped input/output. See memory mapped.
MMU	Memory management unit. A functional unit that translates between effective addresses (EAs) used by programs and real addresses (RAs) used by physical memory. The MMU also provides protection mechanisms and other functions.
MSb	Most-significant bit. The highest-order bit in an address, registers, data element, or instruction encoding.
MSI	Message signaled interrupt.
MSL	Memory Service Layer. The interface logic that provides the connectivity between the DMI interface and the OPMB interface.
MSR	Machine State Register.
no-op	No-operation. A single-cycle operation that does not affect registers or generate bus activity.
OCMB	OpenCAPI Memory Buffer
OMI	Open Memory Interface - Derived from OpenCAPI 3.1 Transaction Layer and 3.0 Data Link Layer Specifications
OPMB	Openpower Memory Bus

OS	Operating system.
page	A region in memory. The Power Architecture defines a page as a 4 KB area of memory, aligned on a 4 KB boundary or a large page size that is implementation dependent.
PBA	Pending bit array.
PBT	Push block transfer.
PCE	PCIe Configuration Environment.
PCIe	Peripheral Component Interface Express.
PEAM	Process element authority mask.
POR	Power-on reset.
POWER	Of or relating to the Power ISA or the microprocessors that implement this architecture.
Power ISA	A computer architecture that is based on the third generation of reduced instruction set computer (RISC) processors.
privileged mode	Also known as supervisor mode. The permission level of operating system instructions. The instructions are described in *PowerPC Architecture, Book III* and are required of software that accesses system-critical resources.
problem state	The permission level of user instructions. The instructions are described in *Power ISA, Books I and II* and are required of software that implements application programs.
PTE	Page table entry. A table that maps virtual addresses (VAs) to real addresses (RAs) and contains related protection parameters and other information about memory locations.
RA	Real address. An address for physical storage, which includes physical memory, local storage (LS), and memory mapped I/O registers. The maximum size of the real-address space is 2^{62} bytes.
ROM	Read-only memory.
SLB	Segment lookaside buffer. It is used to map an effective address to a virtual address.
slbia	SLB invalidate all instruction.
slbie	SLB invalidate entry instruction.
slbmte	SLB move to entry instruction.
SSE	System software environment.
SST	Storage segment table.
STABORG	Segment table origin.
STE	Segment table entry.
storage model	A CAIA-compliant accelerator implements a storage model consistent with the Power ISA.
sync	Synchronize instruction.
synchronization	The process of arranging storage operations to complete in the order of occurrence.
system software	Software that has access to the privileged modes of the architecture. System software is a reference to hypervisor and operating system software.
TAG	PSL command tag.
tag group	A group of PSL commands. Each PSL command is tagged with an n-bit tag group identifier. An AFU can use this identifier to check or wait on the completion of all queued commands in one or more tag groups.
TLB	Translation lookaside buffer. An on-chip cache that translates virtual addresses (VAs) to real addresses (RAs). A TLB caches page-table entries for the most recently accessed pages, thereby eliminating the necessity to access the page table from memory during load-store operations.
tlbie	Translation lookaside buffer invalidate entry instruction.
US	Up Stream - Indicating direction of command and data flow from OPMB to DMI
VA	Virtual address. An address to the virtual-memory space, which is typically much larger than the real address space and includes pages stored on disk. It is translated from an effective address by a segmentation mechanism and used by the paging mechanism to obtain the real address (RA). The maximum size of the virtual-address space is 2^{65} bytes.
VPD	Vital product data.
VPN	Virtual page number. The number of the page in virtual memory.

VSEC	Vendor-Specific Extended Configuration Capability
VSID	Virtual segment ID.
WIMG bits	Four bits in the page table, also called a page-table entry, which control the processor's accesses to cache and main storage. 'W' stands for write through, 'I' for cache inhibit, 'M' for memory coherence, and 'G' for guarded storage.
word	Four bytes.