



# openHMC

an open-source

Hybrid Memory Cube Controller

Computer Architecture Group, University of Heidelberg

in partnership with Micron Foundation

openHMC documentation Rev 1.2

©2014 Computer Architecture Group

# Contents

<b>1 About openHMC</b>	<b>3</b>
1.1 What is openHMC?	3
1.2 The Hybrid Memory Cube	3
1.3 The openHMC Memory Controller	4
1.4 Features	4
<b>2 Module description</b>	<b>6</b>
2.1 Top Module (hmc_controller_top.v)	6
2.2 Asynchronous RX and TX FIFOs (hmc_async_fifo.v)	6
2.3 TX Link (tx_link.v)	6
2.4 RX Link (rx_link.v)	11
2.5 Register File (hmc_controller_8x_rf.v and hmc_controller_16x_rf.v)	13
2.6 Header Files	13
<b>3 Interface description</b>	<b>14</b>
3.1 Memory Controller System Interface	14
3.2 HMC Interface	14
3.3 AXI-4 Stream Protocol Interface	15
3.4 Transceiver Interface	18
3.5 Register File Interface	18
<b>4 Using the Memory Controller</b>	<b>21</b>
4.1 Clocking and Reset	21
4.2 Power Up and (Re-)Initialization	21
4.3 Sleep Mode	22
4.4 Link retraining	22
4.5 Link Retry	22
4.6 Memory Controller Configuration	24
4.7 HMC Configuration	25
<b>5 Implementation Results</b>	<b>27</b>
5.1 Configurations	27
5.2 Resource Utilization	27
5.3 Resource Optimization	28

<b>A Acronyms</b>	<b>i</b>
<b>B Register File Contents</b>	<b>ii</b>
<b>C Directory Structure</b>	<b>v</b>
<b>D Revision History</b>	<b>vi</b>
<b>E List of Figures</b>	<b>viii</b>
<b>F List of Tables</b>	<b>ix</b>
<b>References</b>	<b>x</b>

# 1 | About openHMC

## 1.1 What is openHMC?

openHMC is an open-source project developed by the Computer Architecture Group (CAG) at the University of Heidelberg in Germany. It is a vendor-agnostic, AXI-4 compliant Hybrid Memory Cube (HMC) controller that can be parameterized to different data-widths, external lane-width requirements, and clock speeds depending on speed and area requirements. The main objective of developing the HMC controller is to lower the barrier for others to experiment with the HMC, without the risks of using commercial solutions. This project is licensed under the terms and conditions of version 3 of the Lesser General Purpose License[1].

## 1.2 The Hybrid Memory Cube

The HMC is memory that is built of stacked DRAM, organized in independent sections, so called vaults. Figure 1.1 shows an abstract view of the structure of an HMC. It integrates all DRAM-related management circuits and therefore off-loads the user from any DRAM timings. A single HMC features up to 4 serial links; each running with up to 16 lanes and 15 Gb/s per lane. Transactions are packetized instead of using dedicated data and address strobes. More information on the HMC and its specification is available at the official Hybrid Memory Cube Consortium (HMCC) website at <http://www.hybridmemorycube.org/>.

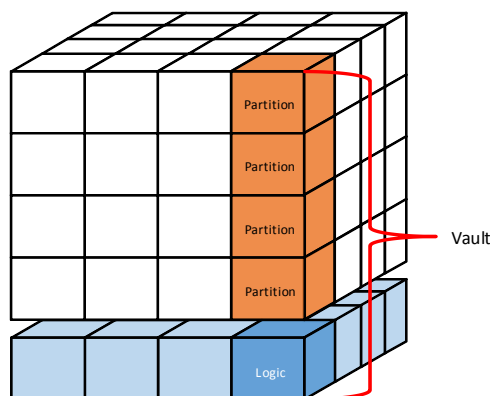
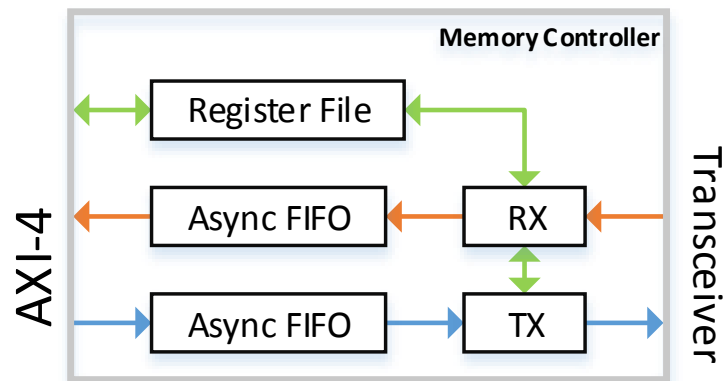


Figure 1.1: HMC: Abstract View

## 1.3 The openHMC Memory Controller

The openHMC memory controller is presented as a high-level block diagram in Figure 1.2. The asynchronous input and output FIFOs allow the user to access the memory controller from a different clock domain. On the transceiver side, a registered output holds the data reordered on a lane-by-lane basis; allowing seamless integration with any transceiver types. A register-file provides access to control and monitor the operation of the memory controller.



**Figure 1.2:** openHMC Memory Controller Block Diagram

## 1.4 Features

The openHMC memory controller implements the following features as described in the HMC specification Rev 2.0 [2]:

- Full link-training, sleep mode and link retraining
- 16Byte up to 128Byte read and write (posted and non-posted) transactions
- Posted and non-posted bit-write and atomic requests
- Mode read and write
- Error response
- Full packet flow control
- Packet integrity checks (sequence number, packet length, CRC)
- Full link retry

### 1.4.1 Supported Modes

Currently the following configurations are supported (8 or 16 lanes):

- 2 FLITs per Word / 256-bit datapath
- 4 FLITs per Word / 512-bit datapath
- 6 FLITs per Word / 768-bit datapath
- 8 FLITs per Word / 1024-bit datapath

Other configurations may require specific CRC implementations and/or initialization schemes. For a more detailed overview of commonly used configuration modes see Chapter 4.

## 2 | Module description

This chapter describes the modules included in the openHMC package. The directory structure is attached in Appendix C.

### 2.1 Top Module (hmc\_controller\_top.v)

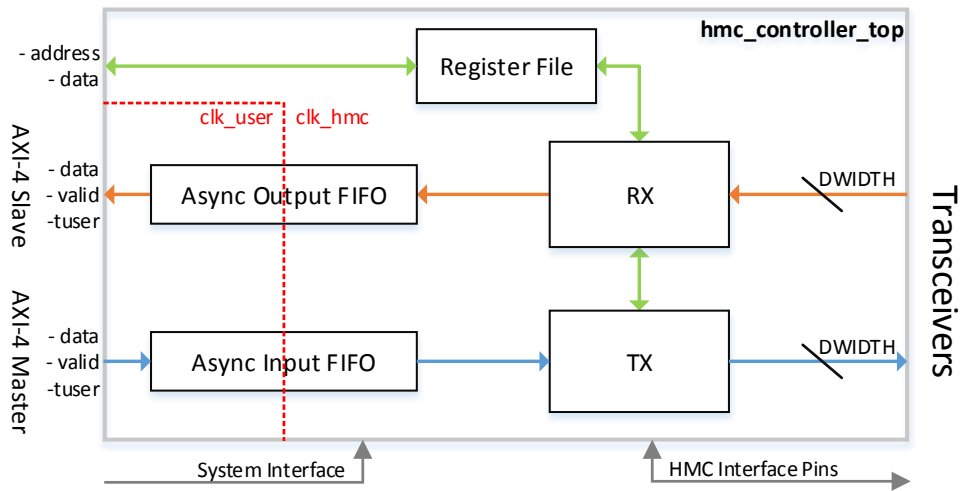
The HMC controller top module instantiates and connects all logical sub-modules and does not contain any logic itself. It provides the AXI-4 , Transceiver and Register File interfaces. Figure 2.1 shows a more detailed view of the memory controller top level including the two clock domains and main interface signals. For a full interface specification refer to Chapter 3. The memory controller is often also referred to as 'Requester' and the data flow from host to HMC is called downstream traffic, or transmit direction (TX). The requester issues request packets and receives responses. On the other hand, the HMC is the 'Responder' and any traffic flowing in host direction is called upstream traffic, or receive direction (RX). The responder receives and processes requests, and returns responses if desired by the request type. In the following, all sub-modules are described in the order they are logically passed by a request/response transaction.

### 2.2 Asynchronous RX and TX FIFOs (hmc\_async\_fifo.v)

The asynchronous FIFOs connect the user logic in the clk\_user clock domain to the HMC controller in the clk\_hmc clock domain. Both FIFOs appear as an AXI-4 Stream Protocol Interface to the user. The full interface specification can be found in Chapter 3.

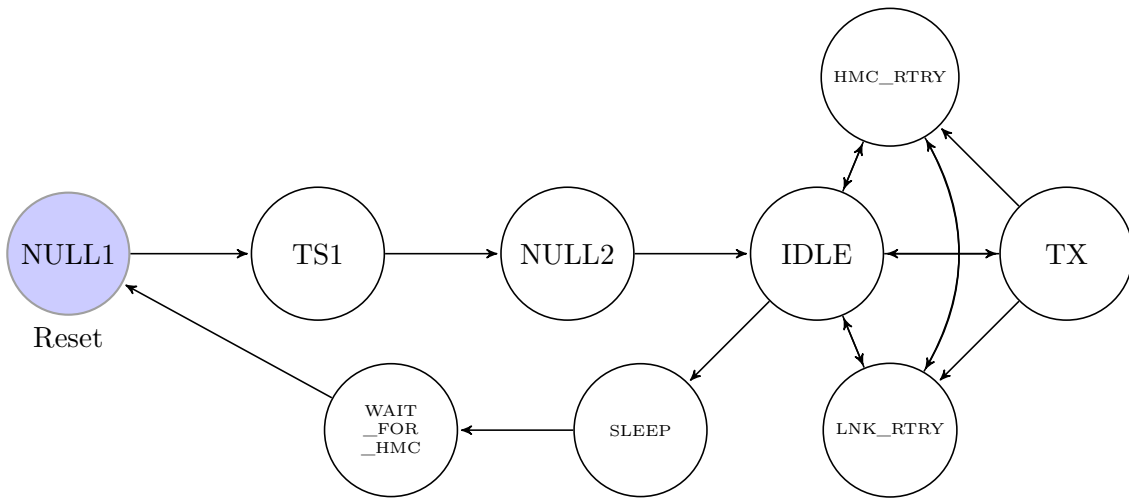
### 2.3 TX Link (tx\_link.v)

The TX Link has two main interfaces, that is the input FIFO interface to receive HMC packets and the output register stage which provides scrambled and lane-by-lane re-ordered data FLITs to connect the transceivers. The user must generate HMC packets within the user logic, including the 64bit header. Also, the user is responsible for operational closure with TAGs, if desired. Note that an unsupported command or a dIn/lng mismatch may produce undefined behavior in the current implementation. The 64bit tail must be set all to zero since it will be filled in the TX Link. Internally, the HMC controller uses register stages to



**Figure 2.1:** Detailed view of the Memory Controller Top Module

encapsulate logically-independent units, and to avoid critical paths due to excessive use of combinational logic. The main control function is implemented as the following Finite State Machine (FSM):



**Figure 2.2:** TX FSM

States and transitions are listed in Table 2.1 and Table 2.2. The next states are listed in the order of their priority. By default, the current state is maintained. For a better understanding of the initialization steps necessary after power-up refer to Section 4.2.

When in TX state, FLITs are processed as implied by the blue path in Figure 2.3. Register File (RF) signals and such that are driven by the RX link are represented by green colored, control signals by gray colored arrows. First, data FLITs are collected at the FIFO. A token handler keeps track of the remaining tokens in the HMC input buffer. With each FLIT transmitted, the token count is decremented. When the token count is sufficient and no other interrupt occurs, the Return Token Count (RTC) is added to return tokens to the HMC, which



**Table 2.1: TX FSM State Table**

State	Description
NULL1	Transmit NULL FLITs (Reset State)
TS1	Transmit the lane dependent TS1 sequence
NULL2	Transmit NULL FLITs
IDLE	Send TRET packet if there are tokens to be returned
TX	Transmit packets
HMC_RTRY	Send start retry packets
LNK_RTRY	Send clear retry packets and perform link retry
SLEEP	Set LxRXPS = low to request HMC sleep mode
WAIT_FOR_HMC	Wait until corresponding LxTXPS pin is high

**Table 2.2: TX FSM Transition Table**

State	Next State & Trigger
NULL1	TS1: RX received NULL FLITs
TS1	NULL2: RX descramblers aligned
NULL2	IDLE: link_is_up
IDLE	HMC_RTRY: force_hmc_retry LNK_RTRY: tx_link_retry_request SLEEP: rf_hmc_sleep TX: retry_buffer !full and tokens are available
TX	HMC_RTRY: force_hmc_retry LNK_RTRY: tx_link_retry_request IDLE: no more data to transmit
HMC_RTRY	LNK_RTRY: tx_link_retry_request TX: retry_buffer !full and tokens are available IDLE: no more data to transmit
LNK_RTRY	HMC_RTRY: force_hmc_retry TX: retry_buffer !full and tokens are available IDLE: no more data to transmit
SLEEP	WAIT_FOR_HMC: as rf_hmc_sleep_requested is de-asserted
WAIT_FOR_HMC	NULL1: as hmc_LxTXPS transitions to high

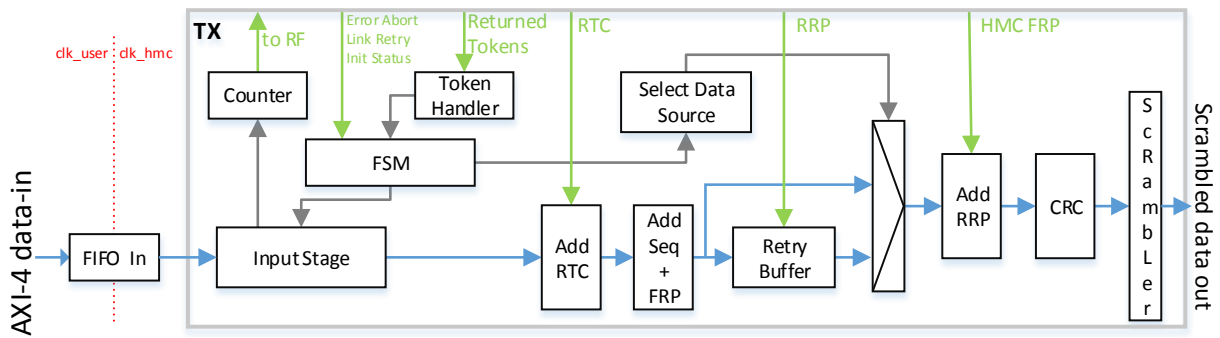


Figure 2.3: TX Link Diagram

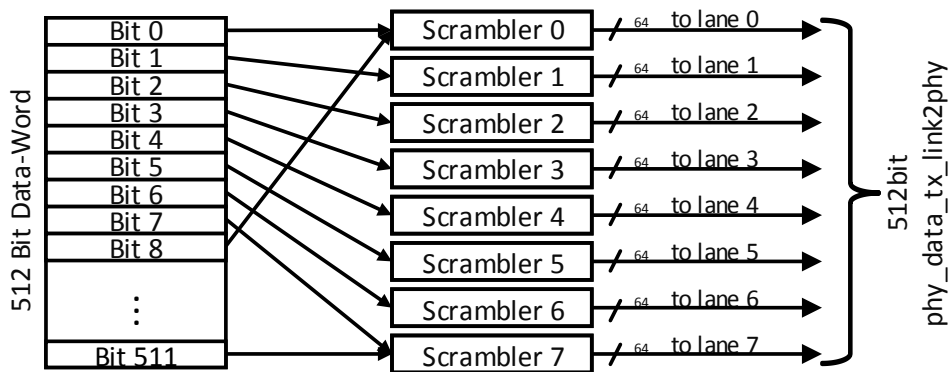


Figure 2.4: Data-Reordering: 4FLIT/512bit example

indicates the number of FLITs that passed the RX input buffer. Afterwards, the Sequence Number (SEQ) and the Forward Retry Pointer (FRP), which is also the retry buffer read pointer, are added. At this point, all FLITs are also stored in the retry buffer. If there is a link retry request (signaled by tx\_link\_retry\_request) data is retransmitted out of the retry buffer, otherwise the regular data path is used. Finally the Return Retry Pointer (RRP) which is the last received HMC FRP is added, the CRC generated, and data is scrambled and reordered on a lane-by-lane basis depending on the configuration (NUM\_LANES and DWIDTH). Figure 2.4 shows an example for a 512-bit / 8-lane configuration where each transceiver connects to 64bit of the parallel output stage.

### 2.3.1 TX Retry Buffer (hmc\_ram.v)

The retry buffer holds a copy of each FLIT transmitted for possible retransmission. NULL FLITs and flow packets, except TRET, are not subject to flow control and retransmission, and are therefore not saved in the retry buffer. The retry buffer actually consists of FPW times 128-bit RAMs so that each FLIT can be addressed independently. One address, which is also the FRP is generated for each packet header. Since the required and accumulated RAM space is defined by the pointer size (FRP = RRP = 8 bit = 256 FLITs), the depth per

**Table 2.3:** RAM Configurations

Datawidth in FPW	Depth per RAM [bits / entries]
2	7 / 128
4	6 / 64
6	5 / 32
8	5 / 32

RAM in this implementation is defined as 256 entries divided by FLITs per Word (FPW). Table 2.3 summarizes the RAM properties for different data-width configurations. Note that a 6-FLIT configuration results in reduced RAM capacity since 6 is not a power of 2 and therefore the next higher of LOG\_FPW must be chosen, leaving some addresses unused. The least significant bits address the target RAM while the remaining bits refer to a specific FLIT within that RAM. The entire value is called FRP, and at the same time is the RAM write pointer. As a result of this addressing scheme, FRPs are not generated consecutively but still incremental, as packets may consist of more than one FLIT. The read pointer of the RAM moves with each RRP received at the RX Link, following the write pointer and therefore excluding potential FLITs from retransmission. The link retry mechanism is described in Section 4.5.

### 2.3.2 Scrambler (tx\_scrambler.v)

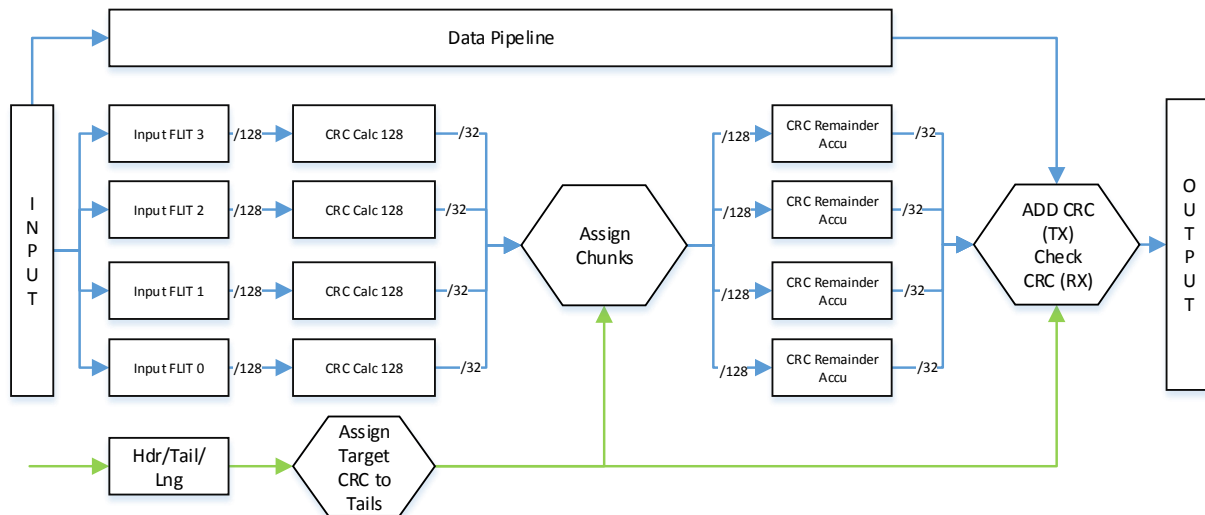
Scramblers are used to ensure Clock-Data Recovery (CDR) over high-speed serial links and replace encodings such as 8b/10b. One scrambler per lane is initialized and preloaded with a lane-specific seed.

### 2.3.3 Lane Run Length Limiter (tx\_run\_length\_limiter.v)

The HMC specification defines a maximum of 85 bits per lane without a logical transition to ensure CDR. When a lane reaches this limitation, a transition must be forced to so that the receiver's Phase-Locked Loops (PLLs) stay locked. The granularity of the run length limiter is adjustable and can be set depending on die area and speed requirements (generally: lower granularity = more logic and area utilization). Also consider technological conditions when determining the best value, e.g. which Loop-Up Tables (LUTs) are used.

### 2.3.4 CRC (tx\_crc\_combine.v)

The CRC architecture was specifically chosen to scale with different data-widths. As can be seen in Figure 2.5 it consists of one 128-bit CRC per FLIT (crc\_128bit\_init). While the CRCs are calculated a specific logic assigns the targeted CRC to the tail of the corresponding



**Figure 2.5:** Scalable CRC Architecture: FPW=4 Example

packet. After the CRCs are calculated all 32-bit remainder that belong to the same packet are shifted to a dedicated accumulation CRC stage, where the remainders form the actual CRC within a single cycle. Finally, the output CRCs are added to the tail of the packets.

### 2.3.5 General Notes on TX Link

The TX link only returns one flow packet per cycle, which is sufficient and an easy way to save some logic. However, (re-)initialization for instance will take some additional cycles to transmit all available tokens since only 31 tokens may be returned within a single Token Return (TRET) packet.

## 2.4 RX Link (rx\_link.v)

The RX Link receives responses issued by the HMC. It then performs data integrity checks, unpacks all valid and required information out of header and tail and forwards the information to the TX Link. After the checks were passed, valid FLITs enter the input buffer and can be collected at the AXI-4 slave interface, including all types of responses. Figure 2.6 shows a block diagram of the RX Link where the data flow is indicated by orange, signals to the TX Link and to the RF by green, and control signals by gray colored arrows. Note that the regular datapath is only selected after link initialization is done. For this purpose the initialization FSM controls a Multiplexer (MUX) to distribute input data.

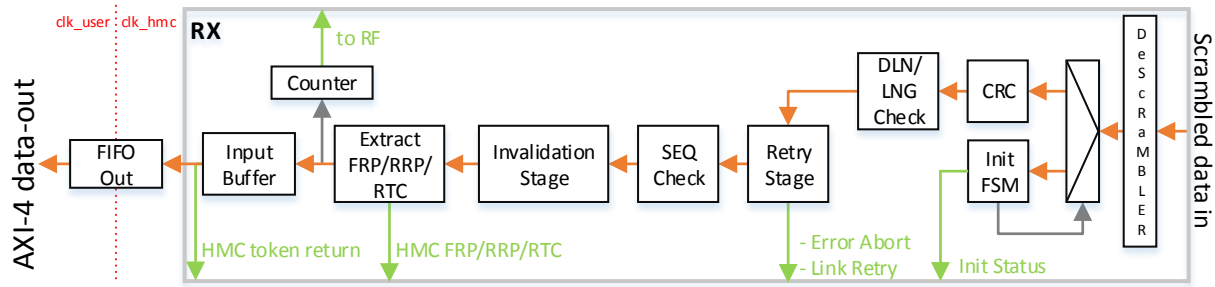


Figure 2.6: RX Link Diagram

### 2.4.1 CRC (rx\_crc\_compare.v)

The `rx_crc_compare` module is very similar to the `tx_crc_combine` instantiated in the TX Link. The biggest difference is that the FLITs are not combined at the end of the data pipeline, but compared. The corresponding poisoned or error flag for the tail of the faulty packet is set if a mismatch occurs. Additionally, the data pipeline of this module holds information bits for valid/header/tail FLITs as this information will be used in the RX link.

### 2.4.2 Descrambler (rx\_descrambler.v)

The `rx_descrambler` module is instantiated once per lane and is self-seeding, which means that it automatically determines the correct value for the internal Linear Feedback Shift Register (LFSR). As the seed for a descrambler is determined, the descrambler is locked. Additionally each descrambler expects a dedicated, so called 'bit\_slip' single input which is used to compensate lane to lane skew. When `bit_slip` is set, input data on the specific lane is delayed by one bit during initialization. This procedure is applied until all descramblers are fully aligned / synchronous to each other.

### 2.4.3 Input Buffer (sync\_fifo\_simple.v)

The input buffer holds  $2^{**} \text{LOG\_MAX\_RTC}$  entries, where each entry is as wide as the datapath (DWIDTH). This results in more resource utilization, but allows a series of  $2^{**} \text{LOG\_MAX\_RTC}$  cycles, carrying one valid FLIT each to be shifted-in without a need for additional buffer distribution and utilization logic. Each valid FLIT at the buffer output on a `shift_out` event returns 1 token to the TX link to be returned as RTC to the HMC. **Note:** Currently, the input buffer is as FPW times bigger than necessary. A specific FLIT assignment and shift in logic is necessary to fully utilize the input buffer and adjust its size.

## 2.5 Register File (hmc\_controller\_8x\_rf.v and hmc\_controller\_16x\_rf.v)

The Register File features three main types of registers: Control, Status, and Counter. Control registers directly affect the memory controller or HMC operation. Status registers can be used to monitor the status of the memory controller, especially during initialization. Counters allow to monitor the performance of the memory controller. For a full list of available registers, see Appendix B. Note that there are several 'reserved' fields which are not listed in the table of registers. These reserved fields provide some space to add additional information, and also align the fields within a register. These unused fields will be tied to constant 0 during synthesis. There are two different RFs that provide the same registers, but a few different signal widths depending on the HMC link configuration (half-width/full-width). The correct RF is instantiated automatically according to the NUM\_LANES parameter.

## 2.6 Header Files

The following header files are present:

### **hmc\_field\_functions.h**

hmc\_field\_functions contains useful functions that return fields such as length or the CRC out of HMC headers or tails

## 3 | Interface description

The following sections contains an interface description for the top module `hmc_controller_top.v`. Since the memory controller can be controlled through parameters, most of the signal-widths depend on the configuration used. The memory controller top module contains a set of parameters that can be used to override the default configuration. All parameters used are listed in Table 3.1.

**Table 3.1:** Configuration Parameters

Parameter	Description	Default
FPW	Desired data-width in FLITs (1FLIT = 128bit). Valid: 4/6/8	4
LOG_FPW	Log of the desired data-width in FLITs	2
DWIDTH	FPW*128, width of the databus in bits	512
LOG_NUM_LANES	Log of the amount of HMC Lanes. Valid: 3/4	4
NUM_LANES	Amount of HMC Lanes (8 or 16)	16
NUM_DATA_BYTES	FPW*16, defines the AXI-4 TUSER bus width in bytes	64
HMC_RF_WWIDTH	Register file <code>rf_write_data</code> bus size in bits	64
HMC_RF_RWIDTH	Register file <code>rf_read_data</code> bus size in bits	64
HMC_RF_AWIDTH	Register file <code>rf_address</code> bus size in bits	4
LOG_MAX_RTC	Log of the max RX input buffer space in FLITs	8
HMC_RX_AC_COUPLED	Set to 0 if HMC RX is DC coupled to Memory Controller TX	1

### 3.1 Memory Controller System Interface

The memory controller top module (`hmc_controller_top`) expects a clock and a reset per clock domain, where each reset must be synchronous to the corresponding clock. Figure 3.1 shows the system interface. Note that both resets are active low.

### 3.2 HMC Interface

The HMC provides the four signals presented in Figure 3.2. Note that the HMC reset `P_RST_N` and the both power-reduction pins `LxRXPS` and `LxTXPS` are active low. The

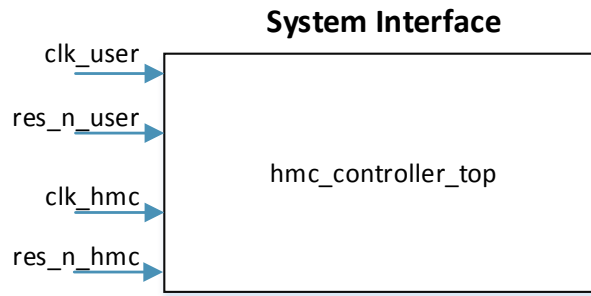


Figure 3.1: System Interface Diagram

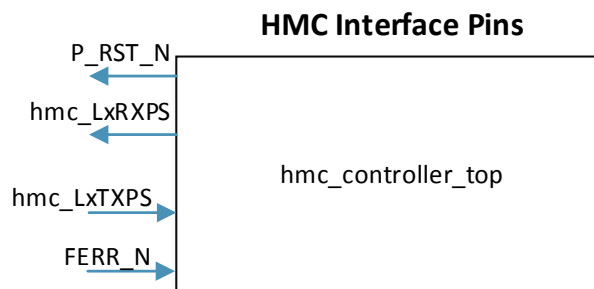


Figure 3.2: HMC Interface Pins Diagram

active low fatal error indicator FERR\_N is not connected in this revision of the memory controller and can be left unconnected.

### 3.3 AXI-4 Stream Protocol Interface

Both AXI-4 interfaces comply with the ARM AMBA AXI-4 Interface Protocol Specification v1.0 [3]. However, not all signals are used. Figure 3.3 provides an interface diagram of the master and slave interfaces used in this implementation. The use and the corresponding size of these signals is described below.

**TREADY** 1 bit

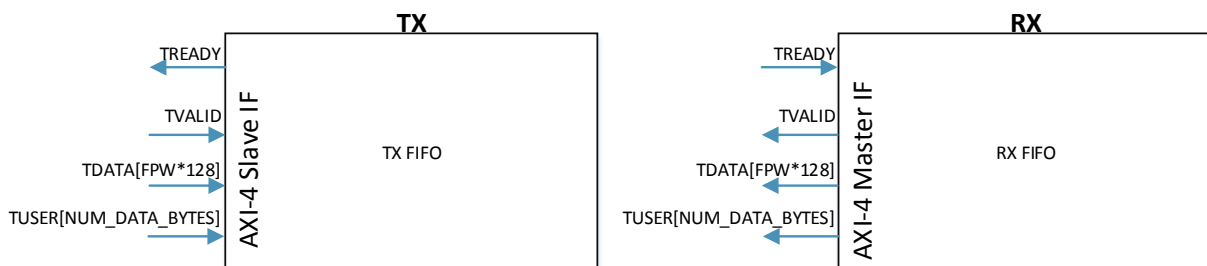


Figure 3.3: AXI-4 Interface Diagram



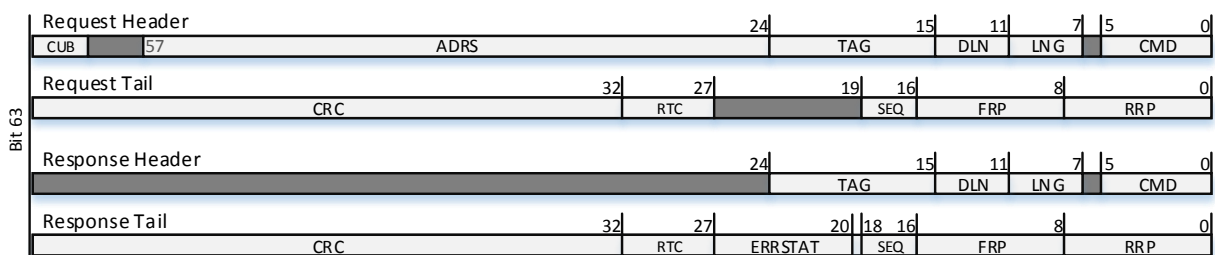
- TX: Memory controller is ready to sample TDATA and TUSER
- RX: Valid data on TDATA and TUSER

**TVALID** 1 bit

- TX: TDATA and TUSER are sampled on TX when TVALID=1 and TREADY=1. TVALID may be held high even when TREADY=0.
- RX: TDATA and TUSER are valid when TVALID=1. TREADY may be held high even when TVALID=0. TDATA and TUSER will not change when TREADY=0.

**TDATA** FPW\*128 bit

The TDATA bus expects complete HMC request packets, starting with the 64bit header followed by data FLITs. The user is responsible to populate all request header fields (see Figure 3.4 or refer to the HMC documentation, chapter 'Request Commands'). Note that the TAG field is optional, but required for operational request/response closure. The tail must be set to all zeros. Figure 3.5 shows an example transaction of multiple different packet types. Packets may start at any 128-bit/ FLIT border. 'Bubbles' between packets are allowed as long as the corresponding valid bit(s) is/are kept low. All FLITs of a packet must be transmitted throughout consecutive FLITs. Also when a packet spreads over multiple 512-bit cycles, TVALID must be held high until the entire packet (including its tail) was transmitted. On RX, the memory controller outputs complete HMC response packets. Data is valid when TVALID=1 and the output will not change while TREADY=0. Contrary to TX, the user has full control on when TREADY is set. When seen a response header the packet does not need to be sampled consecutively throughout its tail.



**Figure 3.4:** HMC Header and Tail

**TUSER** NUM\_DATA\_BYTES bit

The user is responsible to set the following information on the TX TUSER bus respectively the controller provides these information at the RX TUSER bus:

**valid** at TUSER index [FPW-1:0]: Valid FLIT indicator (including header and tail), one bit per FLIT

		Cycle					
		0	1	2	3		
FLIT3 TDATA[511:384]	Data0		Data2 Hdr2			Tail4 Data4	Paket0: 64 Byte Write
FLIT2 TDATA[383:256]	Data0		Tail1 Hdr1			Data4 Hdr4	Paket1: Read
FLIT1 TDATA[255:128]	Data0			Tail2 Data2		Tail3 Data3	Paket2: 32 Byte Write Paket3: 16 Byte Write
FLIT0 TDATA[127:0]	Data0 Hdr0		Tail0 Data0	Data2		Data3 Hdr3	Paket4: 16 Byte Write

**Figure 3.5:** Example transactions on the AXI TX TDATA bus for FPW=4

		Cycle			
		0	1	2	3
Tail TUSER[11:8]	4'b0000		4'b0101	4'b0010	4'b1010
Hdr TUSER[7:4]	4'b0001		4'b1100	4'b0000	4'b0101
Valid TUSER[3:0]	4'b1111		4'b1101	4'b0011	4'b1111
TUSER[11:0]	0x01F		0x5CD	0x203	0xA5F

**Figure 3.6:** TUSER Example for FPW=4

**hdr** at TUSER index  $[(2*FPW)-1:FPW]$ : Header indicator, one bit per FLIT

**tail** at TUSER index  $[(3*FPW)-1:2*FPW]$ : Tail indicator, one bit per FLIT

Every FLIT on the TDATA bus corresponds to one bit in the valid, hdr, and tail fields on TUSER. FLIT 0 at TDATA[127:0] is defined by valid[0] (TUSER[0]), hdr[0](TUSER[FPW]), and tail[0](TUSER[2\*FPW]).

Example:

TDATA holds a header on FLIT position 0 (TDATA[127:0]). Set hdr[0] respectively TUSER[FPW] to 1. Since a header is a valid FLIT, set valid[0] / TUSER[0] to 1. This scheme applies to all FLITs on the TDATA bus. Figure 3.6 illustrates how to set the TUSER signal according to the content of the TDATA bus in Figure 3.5.



### Important

To guarantee proper interface operation, all FLITs of a packet must be shifted in continuously, this means without any 'bubble' FLITs or cycles in between. There is no constraint on inter-packet transmission. Additionally the frequency of the user clock driving the AXI-4 interface must be equal to or higher than `clk_hmc`.

## 3.4 Transceiver Interface

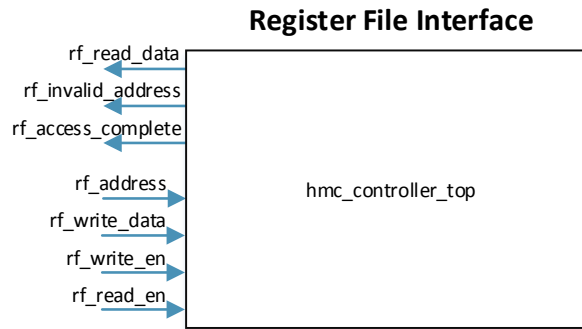
The TX Link provides a DWIDTH wide register output `phy_data_tx_link2phy` with scrambled and lane-by-lane ordered data, driven by `clk_hmc`. The bits `[LANE_WIDTH-1:0]` contain data for lane 0, `[(LANE_WIDTH*2)-1:LANE_WIDTH]` data for lane 1 and so on. An additional input `phy_ready` should be connected to transceivers 'reset\_done' (or similar) to allow monitoring of the transceiver status. The RX Link's data input register `phy_data_rx_phy2link` expects input data by the receivers using the same ordering as explained for the TX Link. Lane reversal is detected and applied in the RX Link and does not affect ordering. Additionally, the RX Link outputs `bit_slip` wires, one per lane used to compensate lane-to-lane skew during initialization. The signals are summarized in Table 3.2.

**Table 3.2:** Transceiver Interface Signals

Signal	Width	Description
<code>phy_data_tx_link2phy</code>	DWIDTH	Lane by lane ordered output
<code>phy_data_rx_phy2link</code>	DWIDTH	Lane by lane ordered input
<code>phy_ready</code>	1	Signalize that the transceivers are ready
<code>bit_slip</code>	HMC_NUM_LANES	Bit_slip is used to compensate lane to lane skew. Bit_slip is controlled by the RX Block for each lane individually

## 3.5 Register File Interface

A Register File module allows to control and monitor the operation of the memory controller. The interface signals are shown in Figure 3.7 and described in Table 3.3. When accessed by software, the user must apply an address. For a write, `write_data` must hold the 64-bit value to be written. Data is sampled when `write_enable` is asserted. For a read the `read_enable` signal must be asserted instead. Each operation is confirmed by the `access_complete` signal set for one cycle. In case that an invalid address was applied, `invalid_address` will remain as long as `read_en` or `write_en` are active. The user must not assert `write_en` and `read_en`



**Figure 3.7:** Register File Interface Diagram

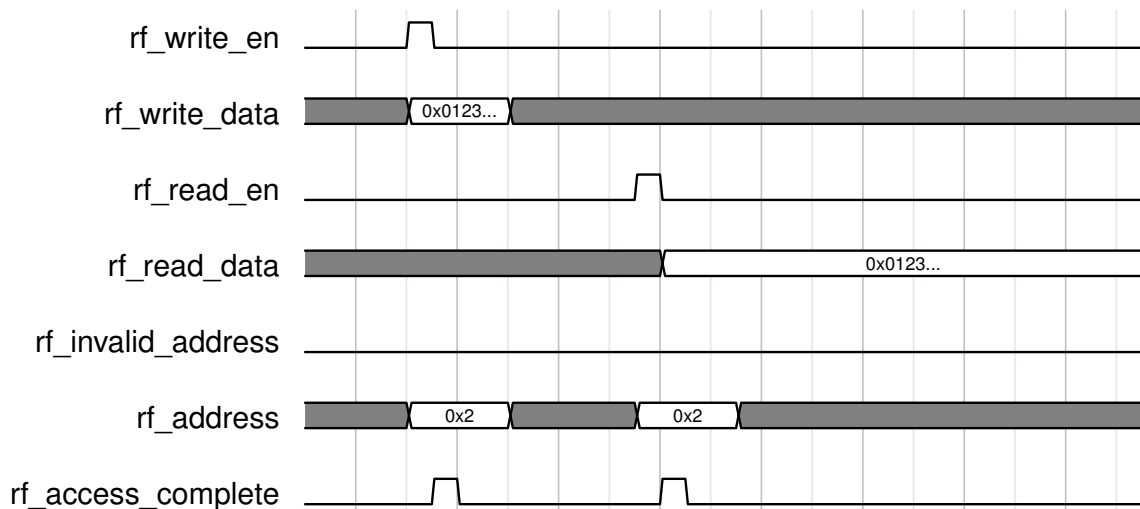
within the same cycle. The RF resides in the `clk_hmc` clock domain and uses the active low `res_n_hmc` reset signal. Figure 3.8 provides an example for a register write followed by a read to address 0x10. Refer to Table 3.4 for the address mapping. For a full listing of all fields within the RF see Appendix B.

**Table 3.3:** Register File Interface Signals

Signal	Width	Description
<code>rf_write_data</code>	<code>HMC_RF_WWIDTH</code>	Value to be written
<code>rf_read_data</code>	<code>HMC_RF_RWIDTH</code>	Requested Value. Valid when <code>access_complete</code> is asserted
<code>rf_address</code>	<code>HMC_RF_AWIDTH</code>	Address to be read or written to.
<code>rf_read_en</code>	1	Read the address provided
<code>rf_write_en</code>	1	Write the value of <code>write_data</code> to the address provided
<code>rf_invalid_address</code>	1	Address out of the valid range
<code>rf_access_complete</code>	1	Indicates a successful operation

**Table 3.4:** Register File Address Map

Register	Address	Description
status_general	0x0	General HMC Controller Status
status_init	0x1	Debug register for initialization
control	0x2	Control register
sent_p	0x3	Number of posted requests issued
sent_np	0x4	Number of non-posted requests issued
sent_r	0x5	Number of read requests issued
poisoned_packets	0x6	Number of poisoned packets received
rcvd_rsp	0x7	Number of responses received
counter_reset	0x8	Reset all counter
tx_link_retries	0x9	Number of Link retries performed on TX
errors_on_rx	0xA	Number of errors seen on RX
run_length_bit_flip	0xB	Number of bit flips performed due to run length limitation

**Figure 3.8:** Register File Access: Write and read register 0x2

## 4 | Using the Memory Controller

The following chapter provides information on how to properly configure and use the openHMC memory controller.

### 4.1 Clocking and Reset

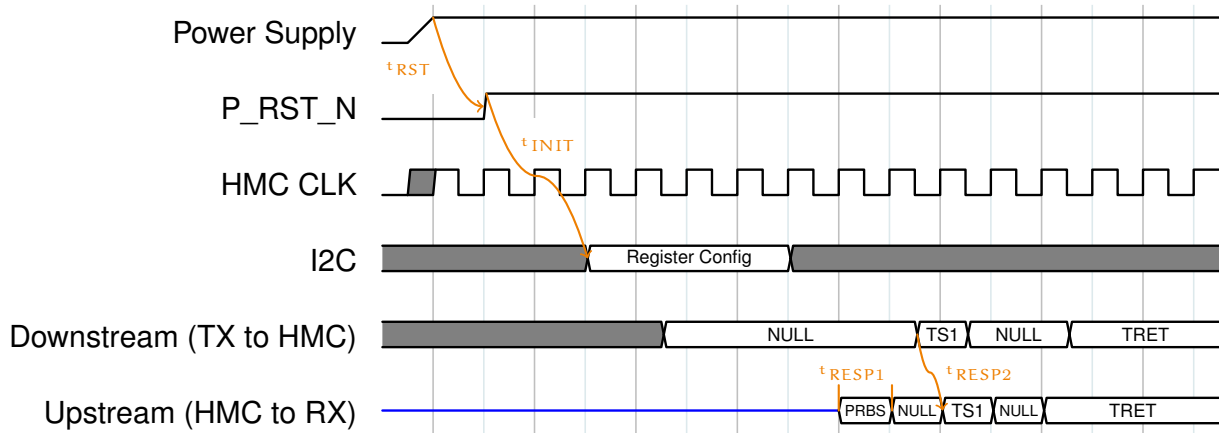
Always keep both reset signals, `res_n_user` and `res_n_hmc` synchronous to their corresponding clock. Although the `‘ifdef ASYNC_RES` macro is used for all clock-triggered `always@` blocks, asynchronous reset should not be used where the target registers do not provide a dedicated asynchronous reset path. This is the case for (almost) all FPGAs.

### 4.2 Power Up and (Re-)Initialization

As soon as both clocks are stable and the low-active `res_n_hmc` has been de-asserted, initialization can begin. The `p_rst_n` bit in the control register can be used to drive the active low HMC reset signal `P_RST_N`. The initialization process is shown in Figure 4.1 where I2C is used to load the internal HMC registers during the register load period (refer to the HMC documentation [2]). The user must set the `hmc_init_cont_set` bit in the control register to allow the descramblers to lock. It is recommended to set this bit shortly after the HMC init-continue sequence at the end of the register load period is issued. No other user activity is required until the `link_is_up` flag in the RF is set. Optionally the user can set the values provided in Table 4.1 prior the de-assertion of `res_n_hmc` which directly affect the initialization process:

**Table 4.1:** Configuration Parameters

Register	Valid values	Description
<code>RX_tokens_av</code>	$0 \leq 1023$	Set the available token space in the RX input buffer. Note: <code>LOG_MAX_RTC</code> must be adjusted to be greater or equal to <code>RX_tokens_av</code>
<code>bit_slip_time</code>	$0 \leq 255$	Cycles between two bit-slips
<code>scrambler_disable</code>	0/1	Disable scrambler and descrambler (can be useful for testing/debugging)



**Figure 4.1:** TX-Link: Initialization Timing

### 4.3 Sleep Mode

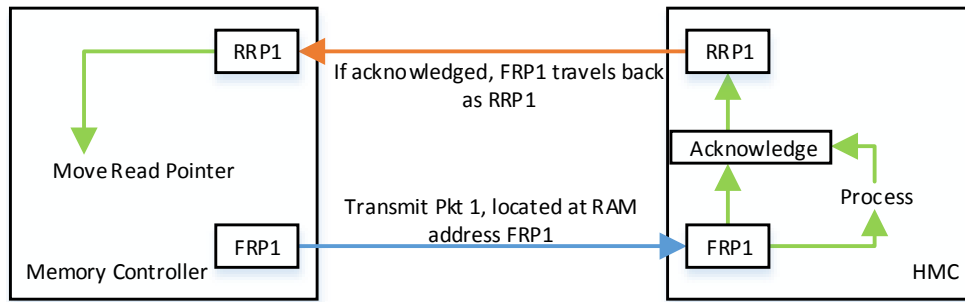
Sleep mode can be safely entered when all in-flight transactions are complete and the TX Block is in IDLE state. For instance, the performance counter in the RF can be used to track the status of outstanding requests. To request sleep mode, the corresponding `set_hmc_sleep` field in the RF control register must be set. The HMC will acknowledge sleep mode by setting the `hmc_LxTXPS` pin low. To exit sleep mode, de-assert `set_hmc_sleep`. The `sleep_mode` field within the RF `status_general` register may be used to monitor the entire process. Upon completion, the link is re-initialized as shown in Figure 4.1, except the need to exchange initial TRETs as memory contents within the HMC are maintained during sleep mode.

### 4.4 Link retraining

When detecting an unacceptable rate of link error monitored by the `link_retries` counter, sleep mode should be entered and exited to retrain the link. All steps described in Section 4.3 apply.

### 4.5 Link Retry

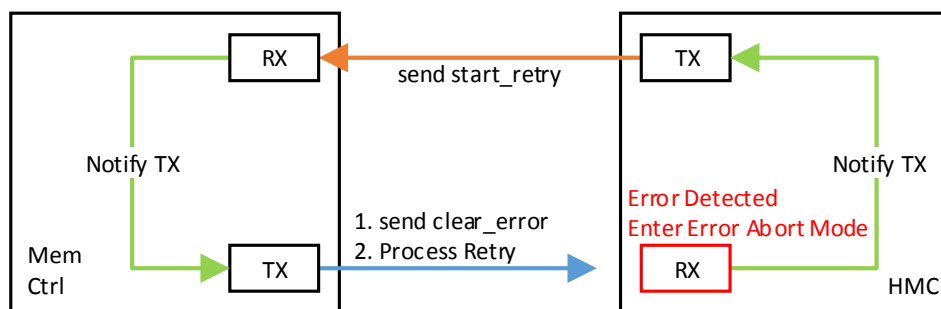
As soon as a link error occurs, the respective receiver of the faulty packet enters the 'Error Abort Mode'. There are two types of link retries that are described in the following. For a better understanding, Figure 4.2 illustrates the flow of pointer between the memory controller and the HMC. Note that both endpoints, memory controller and HMC, generate and check FRP's and RRP's the same way.



**Figure 4.2: Pointer Flow**

### TX Link Retry

In case of an error on the TX path from requester to responder, the HMC will request a link retry. Subsequent received packets arriving at the HMC are dropped, and no header/tail values are extracted. The HMC then issues a programmable series of start\_retry packets to the RX link to force a link retry. Start\_retry packets have the 'StartRetryFlag' set (FRP[0]=1). When the irtry\_received\_threshold at the Receive (RX)-Link is reached, the Transmit (TX) link starts to transmit a series of clear\_error packets that have the 'ClearErrorFlag' set (FRP[1]=1). Afterwards, the TX link uses the last received RRP as the RAM read address and re-transmits any valid FLITs in the retry buffer until the read address equals the write address, meaning that all pending packets were re-transmitted. Upon completion the RAM read address returns to the last received RRP. Re-transmitted packets may therefore be re-transmitted again if another error occurs. Figure 4.3 shows the TX link retry mechanism.

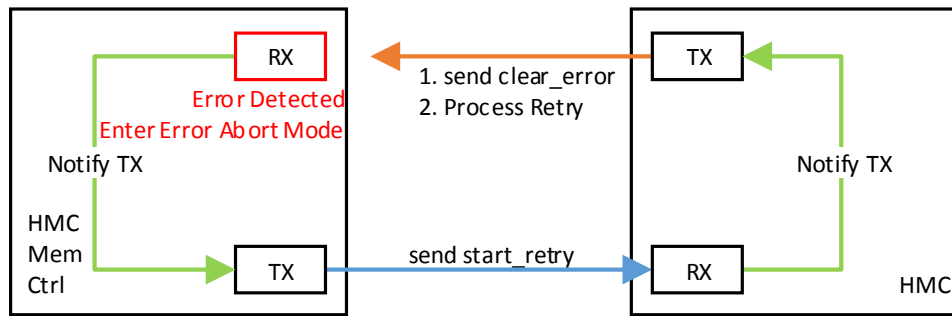


**Figure 4.3: TX Link Retry**

### HMC Retry

In case of an error on the RX path from responder to requester, the RX link will request a link retry. The TX link will then send start\_retry packets whereupon the responder will start to re-transmit all packets that were not acknowledged by the RRP yet. Meanwhile, the RX link remains in the so called error\_abort\_mode where all subsequently incoming packets are





**Figure 4.4:** HMC Retry

dropped. The TX link monitors this state and sends another series of start\_retry packets if the error\_abort\_mode was not cleared after 250cycles. Figure 4.4 shows the TX link retry mechanism.



### Link Retry

For correct link retry operation, equal to or more irtry packets (both types) must be issued than the respective receiver expects. This requirement applies to both, requester and responder. The corresponding irtry\_to\_send value must be equal to or higher than irtry\_received\_threshold in the register file, which is the case for the default values. The internal registers in the HMC must be set accordingly.

## 4.6 Memory Controller Configuration

According the configuration of the data-width (DWIDTH), half-width or full-width (NUM\_LANES) and their respective lane speed, the configurations in Table 4.2 can be applied. Table 4.3 lists all valid parameter sets. The resulting clocking frequencies are calculated with:

$$\text{Frequency[MHz]} = \frac{\text{NUM\_LANES} * \text{LANE\_SPEED}}{\text{DWIDTH}}$$



### Clocking

Assure that clk\_user is equal to or higher than clk\_hmc for proper AXI-4 interface operation

**Table 4.2:** Possible Configurations

DWIDTH [bit]	NUM_LANES	lane speed [Gbits]	clk_hmc [MHz]
256	8	10	312.5
512	8	10	156.25
512	8	12.5	195.3125
512	8	15	234.375
512	16	10	312.5
768	8	15	156.25
768	16	15	312.5
1024	16	10	156.25
1024	16	12.5	195.3125
1024	16	15	234.375

**Table 4.3:** List of valid parameter sets

Desired DWIDTH [bit]	LOG_FPW	FPW
256	1	2
512	2	4
768	3	6
1024	3	8

## 4.7 HMC Configuration

### Input Buffer Token Count

The memory controller must always make sure that no more FLITs are transmitted than the HMC can accept. The number of FLITs that are transmitted is monitored one cycle after the TX FIFOs are shifted out. The resulting value then needs another cycle to be sampled. Hence,  $2 \cdot \text{FPW}$  valid FLITs may be transmitted in this cycle. Furthermore, up to  $\text{FPW}$  FLITs may reside in the TX input buffer when the minimum threshold is reached. To avoid sending more FLITs than the HMC input buffer can hold, set this value to at least  $8'd+(3 \cdot \text{FPW})$ . Otherwise the memory controller might stay IDLE under certain conditions.

### Maximum Packet Size

The user must not send any packets bigger than 'maximum block size' in the HMC Address Configuration Register is set to.



### Token Count

Set the minimum token count within the HMC token register to at least  $8+(3*FPW)$

## 5 | Implementation Results

### 5.1 Configurations

The openHMC memory controller was verified in simulation using multiple verification environments, including the HMC BFM (bus functional model). Additionally, it was successfully implemented and tested with the device(s) listed in Table 5.1. The Xilinx Vivado Design Suite 2014.3.1 was used as implementation tool, with the Vivado Default Synthesis and Implementations settings. LOG\_MAX\_RTC was set to 8. The run\_length\_limiter was used with its granularity set to 4.

**Table 5.1:** FPGA-Verified Configurations

ID	FPW	NUM LANES	LANE SPEED [Gbit]	clk_hmc [MHz]	Target
1	4	8	10	156.25	Xilinx Virtex Ultrascale XCVU095-FFVD1924-2-e-es1
2	4	8	12.5	195.3125	Xilinx Virtex Ultrascale XCVU095-FFVD1924-2-e-es1

In addition to the FPGA-verified configurations, the openHMC controllers was successfully implemented for an 8FLIT, 1024bit datapath, 12.5Gbit configuration.

### 5.2 Resource Utilization

Table 5.2 gives an overview over the approximate resource utilization for each implementation run listed in Table 5.1, matched by the ID. Note that the presented values are the results for the openHMC controller implemented in a larger design along with other components. Also, other implementation strategies may be used to target other area or performance goals.

**Table 5.2:** Resource Utilization

ID	LUTs combined [% of device]	Registers [% of device]	BRAM B36/B18 [% of device]
1,2	16392 [3.04%]	13051[1.21%]	16 [0.92%]

## 5.3 Resource Optimization

The following design advice can be used to reduce resource utilization, if applicable.

### Disable Run Length Limiter

If HMC RX is DC coupled to Memory Controller TX, set the parameter 'HMC\_RX\_AC\_COUPLED' in `hmc_controller_top` to 0 in order to allow the synthesis tool to remove the run length limiter. DC coupled links are not subject to run length limitation.

## A | Acronyms

<b>CAG</b>	Computer Architecture Group
<b>CDR</b>	Clock-Data Recovery
<b>FPW</b>	FLITs per Word
<b>FRP</b>	Forward Retry Pointer
<b>FSM</b>	Finite State Machine
<b>HMC</b>	Hybrid Memory Cube
<b>HMCC</b>	Hybrid Memory Cube Consortium
<b>LFSR</b>	Linear Feedback Shift Register
<b>LUT</b>	Loop-Up Table
<b>MUX</b>	Multiplexer
<b>PLL</b>	Phase-Locked Loop
<b>RF</b>	Register File
<b>RRP</b>	Return Retry Pointer
<b>RTC</b>	Return Token Count
<b>RX</b>	Receive
<b>SEQ</b>	Sequence Number
<b>TRET</b>	Token Return
<b>TX</b>	Transmit

## B | Register File Contents

### Legend

**HW** Hardware access rights (through port list)

**SW** Software access rights (through RF interface)

**wo** write-only

**ro** read-only

**rw** read-write

**Table B.1:** Status General

Field	# Bits	Description & Encoding	Reset	HW	SW
link_up	1	Link is ready for operation	0	wo	ro
link_training	1	Link training in progress	0	wo	ro
sleep_mode	1	HMC is in Sleep Mode	0	wo	ro
lanes _reversed	1	0: Normal Operation 1: Connect Lane 7 to 0, Lane 6 to 1, and so on (for 8x operation)	0	wo	ro
phy_rdy	1	SerDes reset is done	0	wo	ro
hmc_tokens _remaining	10	Amount of tokens remaining in the HMC input buffer	0	wo	ro
rx_tokens _remaining	10	Amount of tokens remaining in the MemCtrl RX input buffer	0	wo	ro
lane __polarity _reversed	NUM HMC LANES	0: Normal Operation 1: Data is logically inverted lane-by- lane	0	wo	ro

**Table B.2: Status Init**

Field	# Bits	Description & Encoding	Reset	HW	SW
lane _descramblers _locked	NUM HMC LANES	Lane by lane descrambler locked	0	wo	ro
descrambler_part _aligned	NUM HMC LANES	Lane by lane descrambler partially aligned	0	wo	ro
descrambler _aligned	NUM HMC LANES	Lane by lane descrambler fully aligned	0	wo	ro
all_descramblers _aligned	1	All descramblers are aligned	0	wo	ro
tx_init_status	2	Init Status of the TX Block  0: NULL1 1: TS1 3: NULL2	0	wo	ro
hmc_init_ts1	1	HMC sends TS1 packets	0	wo	ro

**Table B.3: Performance Counter**

Field	# Bits	Description & Encoding	Reset	HW	SW
poisoned_packets	64	Number of poisoned packets received	0	wo	ro
sent_np	64	Number of non posted requests issued (including all types)	0	wo	ro
sent_p	64	Number of Posted Data Write requests issued	0	wo	ro
sent_r	64	Number of Read Data requests issued	0	wo	ro
rcvd_rsp	64	Number of responses received	0	wo	ro



**Table B.4: Control**

Field	# Bits	Description & Encoding	Reset	HW	SW
p_rst_n	1	Active low HMC reset.	1	ro	rw
hmc_init_cont_s	1	Allow descramblers to lock	1	ro	rw
set_hmc_sleep	1	Request HMC sleep mode. Sleep mode can be monitored by the 'sleep_mode' field in the Status General Register	1	ro	rw
scrambler_disable	1	Disable Scrambler and Descrambler for testing purposes	1	ro	rw
run_length_enable	1	Disable the run length limiter in the TX scrambler logic	1	ro	rw
first_cube_ID	1	Set the Cube ID of the first HMC connected. Used in irtry packets	0	ro	rw
debug_dont_send_tret	1	Prohibit memory controller from sending any TRET packets	0	ro	rw
rx_token_count	1	Set the input buffer space in the RX block	RX_TOKEN_CNT	ro	rw
irtry_received_threshold	1	Set the number of irtry packets to be received until an action is performed.	0x10	ro	rw
irtry_to_send	1	Set the number of irtry to be sent	0x14	ro	rw
bit_slip_time	1	Set the time (in cycles) between to bit_slip impulses. Used for receiver alignment during initialization	0x24	ro	rw

**Table B.5: Other Counter**

Field	# Bits	Description & Encoding	Reset	HW	SW
tx_link_retries	32	Incremental 1-bit counter: Number of Link retries performed on TX	0	wo	ro
errors_on_rx	32	Incremental 1-bit counter: Number of successful HMC retries performed	0	wo	ro
run_length_bit_flip	32	Incremental 1-bit counter: How many bit_flips were performed by the run length limiter	0	wo	ro
counter_reset	1	Reset counter in 'Other counter'. This field is automatically cleared	0	wo	ro

## C | Directory Structure

```

openhmc
├── rtl
│   ├── building_blocks
│   │   ├── counter
│   │   │   └── counter48.v
│   │   └── fifos
│   │       ├── async
│   │       │   └── hmc_async_fifo.v
│   │       ├── sync
│   │       │   ├── sync_fifo_reg_stage.v
│   │       │   ├── sync_fifo.v
│   │       │   └── sync_fifos.f
│   │       └── rams
│   │           └── hmc_ram.v
│   └── hmc_controller
│       ├── crc
│       │   ├── crc_128_init.v
│       │   └── crc_accu.v
│       ├── register_file
│       │   ├── hmc_controller_16x_rf.v
│       │   └── hmc_controller_8x_rf.v
│       ├── rx
│       │   ├── rx_crc_compare.v
│       │   ├── rx_descrambler.v
│       │   ├── rx_lane_logic.v
│       │   └── rx_link.v
│       ├── tx
│       │   ├── tx_crc_combine.v
│       │   ├── tx_link.v
│       │   ├── tx_run_length_limiter.v
│       │   └── tx_scrambler.v
│       ├── hmc_controller_top.f
│       └── hmc_controller_top.v
│   └── include
│       └── hmc_field_functions.h
├── doc
│   └── openhmc_doc_rev1_2

```

# D | Revision History

## 1.2 The following changes have been made

### Controller

- Renamed RAM and FIFO instances to avoid interference with modules names in larger designs
- Adjusted width of the performance counters to the RF read data width
- Removed unused nets 'rx\_buffer\_rtc' in rx\_link and 'rf\_link\_width' in hmc\_controller\_top
- Added debug register 'hmc\_init\_cont\_set'
- Added Register File for 8 lane configuration

### Documentation (Section Number)

- \* Updated descriptions for several chapters
- 3.3** Added additional information and examples for the AXI-4 Interface
- 3.5** Added register address table and Register File access timing diagram
- 5** Display only FPGA-verified results

## 1.1 The following changes have been made

### Controller

- Complete new CRC architecture with less overall delay, improved timing, and less resource utilization
- Added 2-FLIT configuration support
- Removed a FIFO that became obsolete due to the new CRC architecture
- Decreased the width of all irtry packet related counter to 5 bits (maximum: 31 retry packets to receive/ to send)
- Fix: Error response packets do not increment the response counter anymore

### Documentation (Section Number)

- 1** Added 2 FLIT configuration to the 'Supported Modes' listing
- 2.3.4** Exchanged the CRC structure schematic with the new architecture. Removed the note for the former CRC

**4.8** Removed design advice regarding the 'retry pointer loop time'. It does not apply to the new CRC architecture

**5.1** Added target ID 4

**5.2** Added resource utilization overview

**1.0** First release

## E | List of Figures

1.1 HMC: Abstract View . . . . .	3
1.2 openHMC Memory Controller Block Diagram . . . . .	4
2.1 Detailed view of the Memory Controller Top Module . . . . .	7
2.2 TX FSM . . . . .	7
2.3 TX Link Diagram . . . . .	9
2.4 Data-Reordering: 4FLIT/512bit example . . . . .	9
2.5 Scalable CRC Architecture: FPW=4 Example . . . . .	11
2.6 RX Link Diagram . . . . .	12
3.1 System Interface Diagram . . . . .	15
3.2 HMC Interface Pins Diagram . . . . .	15
3.3 AXI-4 Interface Diagram . . . . .	15
3.4 HMC Header and Tail . . . . .	16
3.5 Example transactions on the AXI TX TDATA bus for FPW=4 . . . . .	17
3.6 TUSER Example for FPW=4 . . . . .	17
3.7 Register File Interface Diagram . . . . .	19
3.8 Register File Access: Write and read register 0x2 . . . . .	20
4.1 TX-Link: Initialization Timing . . . . .	22
4.2 Pointer Flow . . . . .	23
4.3 TX Link Retry . . . . .	23
4.4 HMC Retry . . . . .	24

## F | List of Tables

2.1 TX FSM State Table . . . . .	8
2.2 TX FSM Transition Table . . . . .	8
2.3 RAM Configurations . . . . .	10
3.1 Configuration Parameters . . . . .	14
3.2 Transceiver Interface Signals . . . . .	18
3.3 Register File Interface Signals . . . . .	19
3.4 Register File Address Map . . . . .	20
4.1 Configuration Parameters . . . . .	21
4.2 Possible Configurations . . . . .	25
4.3 List of valid parameter sets . . . . .	25
5.1 FPGA-Verified Configurations . . . . .	27
5.2 Resource Utilization . . . . .	27
B.1 Status General . . . . .	ii
B.2 Status Init . . . . .	iii
B.3 Performance Counter . . . . .	iii
B.4 Control . . . . .	iv
B.5 Other Counter . . . . .	iv

## References

- [1] Free Software Foundation, Inc. GNU Lesser General Public License.  
<http://www.gnu.org/licenses/lgpl.html>. [last accessed 12-Sep-2014].
- [2] Hybrid Memory Cube Consortium. Hybrid Memory Cube Specification 2.0.  
<http://www.hybridmemorycube.org/>. [last accessed 12-Dec-2014].
- [3] ARM Limited. AMBA AXI4-Stream Protocol Specification v1.0.  
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ih0051a/index.html>.  
[last accessed 16-Aug-2014].