openHMC

an open-source
Hybrid Memory Cube Controller

Computer Architecture Group, University of Heidelberg

in partnership with Micron Foundation

openHMC documentation Rev 1.1

Contents

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 What is the Hybrid Memory Cube ?

The HMC is memory that is built of stacked DRAM, specified by the Hybrid Memory Cube Consortium (HMCC). It integrates all DRAM-related management circuits and therefore offloads 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. For more information on the HMC, see the official specification that can be downloaded at http://www.hybridmemorycube.org/.

1.3 The openHMC Memory Controller

The openHMC memory controller is presented as a high-level block diagram in Figure 1.1. 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.

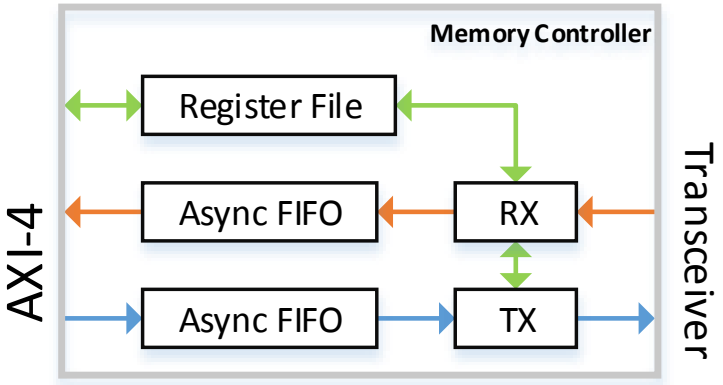

Figure 1.1: openHMC Memory Controller Block Diagram

1.4 Features

The openHMC memory controller implements the following features as described in the HMC specification Rev 1.1 [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

1.5 Missing Features

Currently the following features are missing:

- Warm reset

1.5.1 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 diretory structure of these files is attached in Appendix D.

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 only 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 depending on the packet 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 (async_fifo.v)

The asynchronous input and output FIFOs connect the user logic to the HMC controller, allowing both components to operate in different clock domains. The FIFOs connect the TX and RX Links which are in the clk_hmc clock-domain to the user AXI interfaces in the clk_user clock domain. Both FIFOs hold full HMC packets, including header, data, and tail. Additional information bits must be set to indicate the presence of valid, header, and tail FLITs. Respectively, the memory controller provides these signals at the RX output FIFO. Both interfaces appear as AXI-4 Stream Protocol interfaces with separated data and information buses to the user. The interface signals and their use are described in Chapter 3. There is no rule for packet alignment on the TDATA bus when the corresponding information bits are set accordingly. This also allows multiple packets to be transmitted/received within a single cycle.

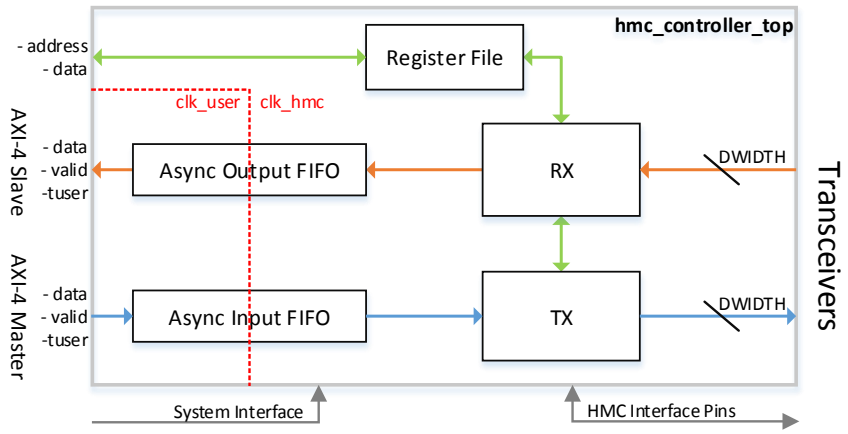

Figure 2.1: Detailed view of the Memory Controller Top Module

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 and set the fields in the header. Also, the user is responsible for operational closure with TAGs, if desired. Note that an unsupported command or a dln/lng mismatch may produce undefined behavior in the current implementation. The tail must be set to zero since it will be filled in the TX Link. Internally, the HMC controller uses register stages to 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):

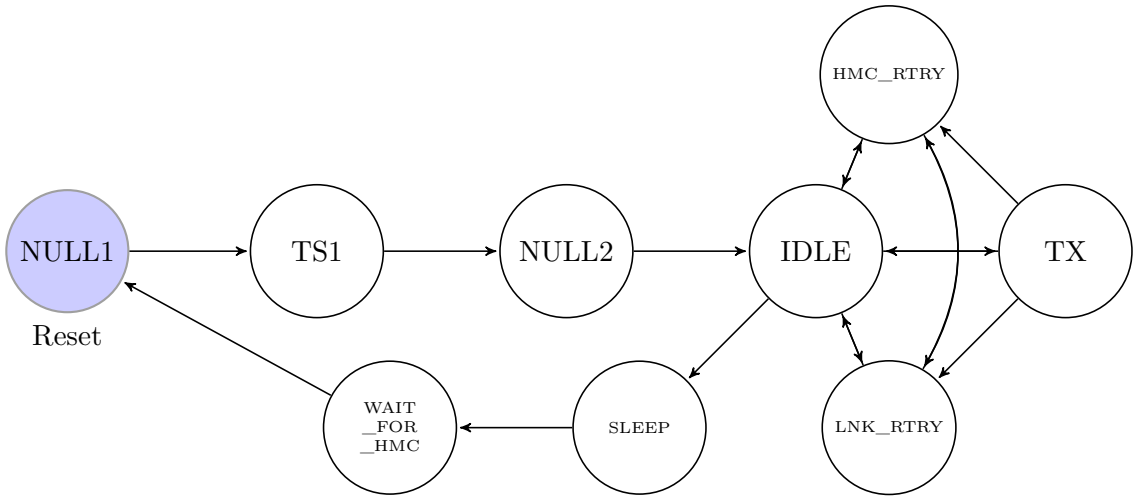

Figure 2.2: TX FSM

States and transitions are listed in Table 2.1 and Table 2.2. The next states are listed in the

Table 2.1: TX FSM State Table

State	Description
NULL1	TS1
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

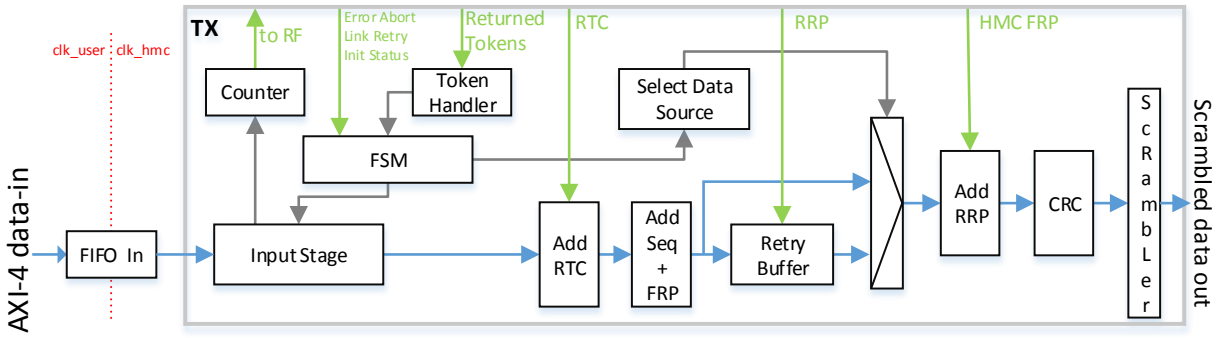

Figure 2.3: TX Link Diagram

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 grey colored arrows. First, data FLITs are collected at the FIFO outputs. A token handler keeps track of the remaining tokens in the HMC input buffer. With each FLIT transmitted, this token count is decremented accordingly. 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 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. Then, the Return Retry Pointer (RRP) which is the last received HMC FRP is added, the CRC generated, and finally the data is scrambled and reordered on a lane-by-lane basis depending on the configuration (HMC_NUM_LANES and DWIDTH). Figure 2.4 shows an example for a 512-bit / 8-lane configuration.

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

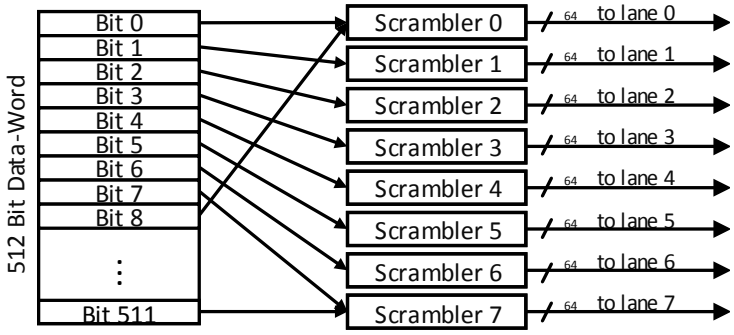

Figure 2.4: Data-Reordering

2.3.1 TX Retry Buffer (ram.v)

A 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 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 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 addressable values unused.
The least significant bits address a RAM whereas 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 RRPs 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 pre-loaded 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:

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

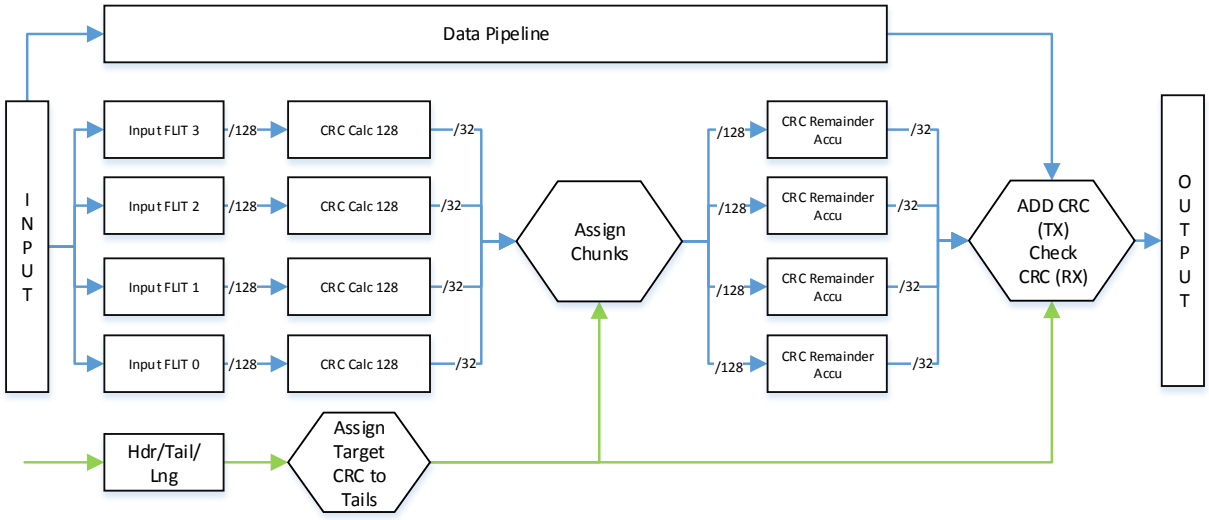

Figure 2.5: Scalable CRC Architecture: FPW=4 Example

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 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 grey 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.

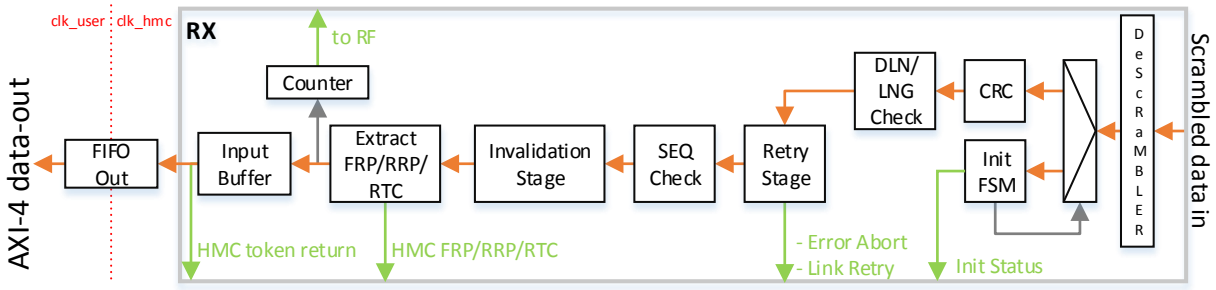

Figure 2.6: RX Link Diagram

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 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**LOG_MAX_RTC entries, where each entry is FPW times FLIT wide. This results in more resource utilization, but allows a series of 2**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 it's size.

2.5 Register File (hmc_controller_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. The most important register is P_RES_N, which is the active low HMC reset out. 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 C. 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.

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_controll_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. Valid: 4/6/8	4
LOG_FPW	Log of the desired data-width in FLITs	2
DWIDTH	FPW*128	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	64
HMC_RF_WWIDTH	Register file rf_write_data bus size	64
HMC_RF_RWIDTH	Register file rf_read_data bus size	64
HMC_RF_AWIDTH	Register file rf_address bus size	4
LOG_MAX_RTC	Log of the maximum RX input buffer space in FLITs	8

3.1 Memory Controller System Interface

The memory controller top module 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 the reset is an active low signal.

3.2 HMC Interface

The HMC provides the four control 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 (also active low) fatal error indicator FERR_N is not connected it this revision of the memory controller and be left unconnected.

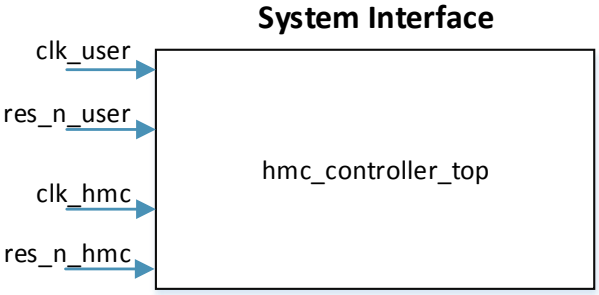

Figure 3.1: System Interface Diagram

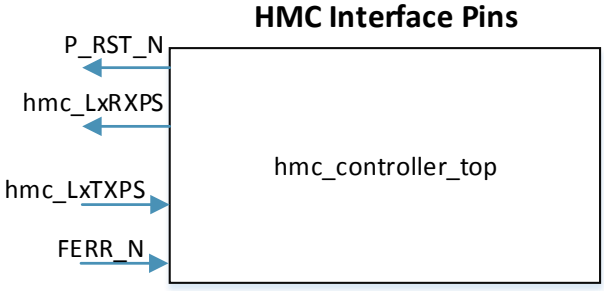

Figure 3.2: HMC Interface Pins Diagram

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

- TX: Input FIFO is ready to accept data
- RX: Valid data at the RX FIFO output

TVALID 1 bit

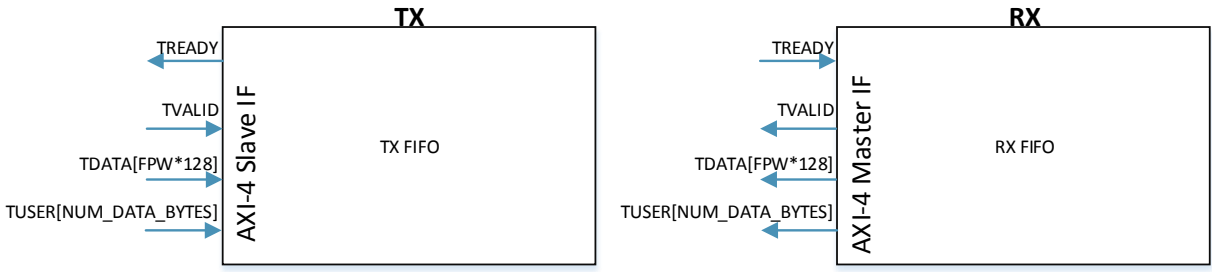

Figure 3.3: AXI-4 Interface Diagram

Indicates valid data at the FIFO input (TX) respectively the FIFO output (RX). Data is sampled when TVALID=1 and TREADY=1. The user is allowed to set TVALID for TX even if TREADY=0

TDATA FPW*128 bit

Data to be sampled (Header - Data - Tail)

TUSER NUM_DATA_BYTES bit

The user is responsible to provide the following information on the TX interface. Also, the fields described are present for data FLITs at the RX interface:
valid [FPW-1:0]: Valid FLIT indicator (including header and tail), one bit per FLIT
hdr [(2*FPW)-1:FPW]: Header indicator, one bit per FLIT
tail [(3*FPW)-1:2*FPW]: Tail indicator, one bit per FLIT

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. 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
phy_data_tx_link2phy	DWIDTH	Lane by lane ordered output
phy_data_rx_phy2link	DWIDTH	Lane by lane ordered input
phy_ready	1	Signalize that the transceivers are ready
bit_slip	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

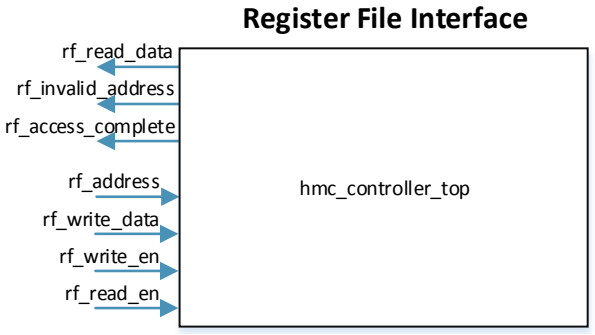

Figure 3.4: Register File Interface Diagram

3.5 Register File Interface

A Register File module provides the possibility to control the memory controller and also contains status information such as initialization progress or performance counter. For a full listing of all fields available see Appendix C. The RF provides the interface signals shown in Figure 3.4 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 and the write_enable signal must be asserted in the same cycle. For a read, apply address and assert the read_enable signal instead. Each operation is confirmed by the access_complete signal set for one cycle. In case that an unknown 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 within the same cycle. The RF resides in the clk_hmc clock domain and uses the active low res_n hmc reset signal.

Table 3.3: Register File Interface Signals

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

4 Using the Memory Controller

The following chapter provides information on how to properly configure and use the openHMC memory controller. The configuration advice must be followed to guarantee correct behavior.

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 as presented in Figure 4.1 begins. No user input is necessary 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:

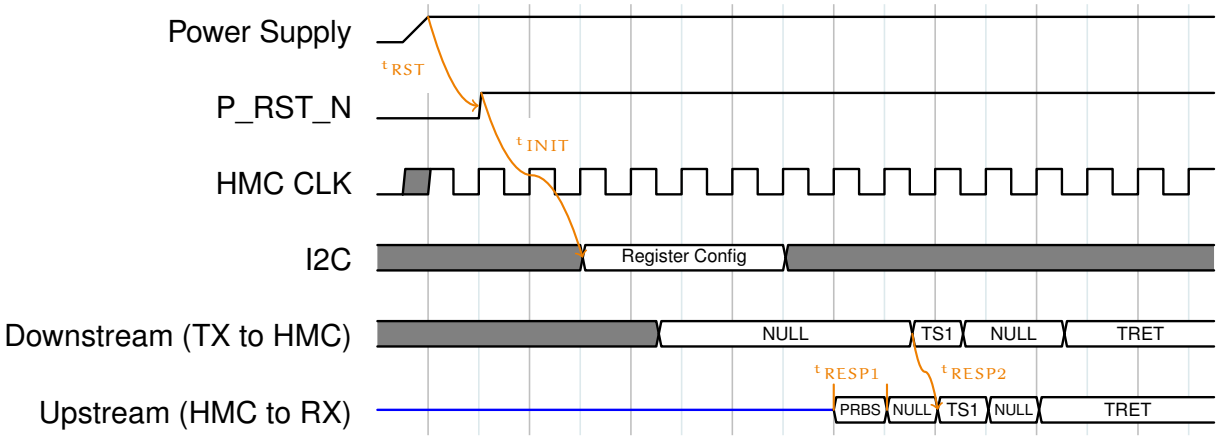

Figure 4.1: TX-Link: Initialization Timing

Table 4.1: Configuration Parameters

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

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 receiver of the poisoned 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.

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

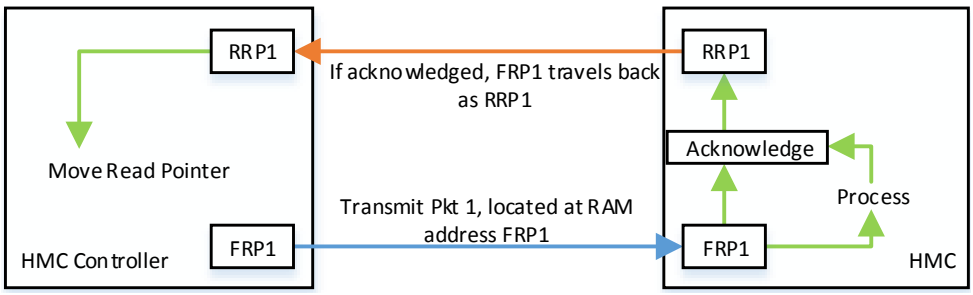

Figure 4.2: Pointer Flow

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 where 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.

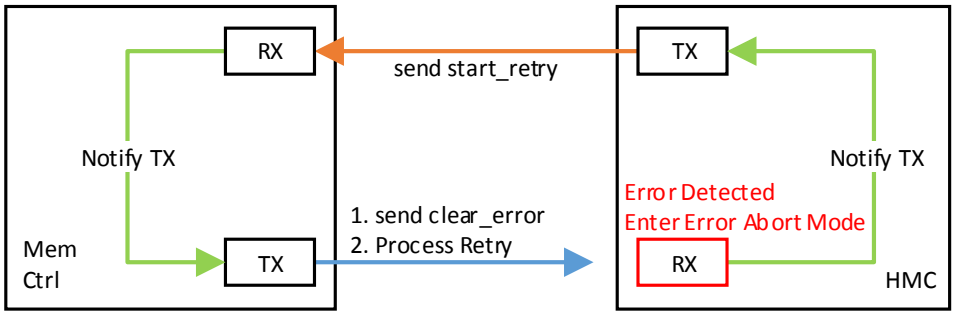

Figure 4.3: TX Link Retry

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 than 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 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.

Note: For a correct operation of the link retry mechanism, equal to or more irtry packets

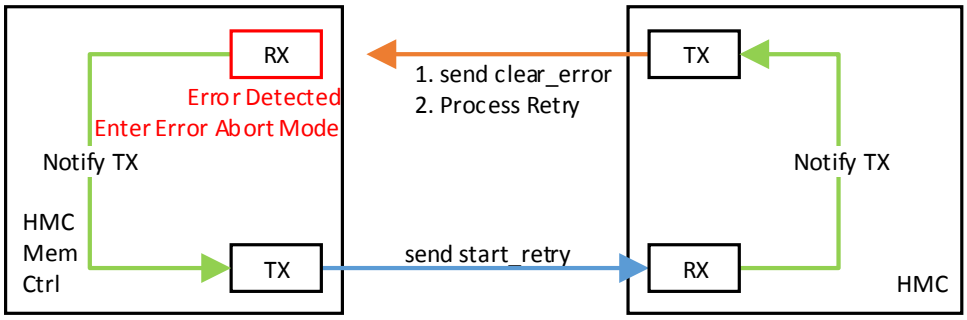

Figure 4.4: HMC Retry

(both types) must be issued than the reiver 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 Restrictions

The following restrictions apply to the current implementation of the memory controller

- The frequency of the clock clk_user must be equal to or higher than clk_hmc. A packet must be shifted in without interruption from the cycle containing the header throughout the cycle holding the tail, as the TX block passes any valid FLITs immediately.

4.7 Configuration

According the configuration of the data-width (DWIDTH), the number of HMC lanes (HMC_NUM_LANES) and their respective lane speed, the configurations in Table 4.2 can be applied, and Table 4.3 lists all valid parameter sets. The resulting clocking frequencies are calculated with:

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

4.8 Important Notes on HMC configuration

The following restrictions apply to the openHMC memory controller regarding the configuration of the HMC. The fields mentioned do not belong to the memory controller, but to the HMC's internal configuration register.

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

Table 4.4: Configuration Parameters

Link Input Buffer Max Token Count	
> 8'd+(3*FPW)	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*FPW valid FLITs may be transmitted in this cycle. Furthermore, up to 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*FPW). Otherwise the memory controller might stay IDLE under certain conditions

Other Useful Hints

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

5 | Implementation Results

5.1 Implemented Configurations

All of the features and configurations that were described in this document were verified in simulation using a bus functional HMC model by Micron as well as a verification environment developed by the CAG. The openHMC memory controller was successfully implemented with closed timing for the configurations listed in Table 5.1. The Xilinx Vivado Design Suite 2014.2 was used as implementation tool, with the Vivado Default Synthesis settings and the Performance_ExplorePostRoutePhysOpt strategy for implementation. LOG_MAX_RTC was set to 8 for each run. The run_length_limiter granularity was set to 4.

Table 5.1: Implementation Results

ID	FPW	NUM _LANES	LANE _SPEED [Gbit]	clk_hmc [MHz]	Target
1	4	8	10	156.25	Xilinx Virtex 7 XC7VX690TFFG1761-2
2	4	8	12.5	195.3125	Xilinx Virtex 7 XC7VX690TFFG1761-3
3	8	16	10	156.25	Xilinx Kintex Ultrascale XCKU-040-ffva1156-3-e-es1
4	8	16	12.5	195.3125	Xilinx Kintex Ultrascale XCKU-040-ffva1156-3-e-es1

5.2 Resource Utilization

Table 5.2 gives an overview over the 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)	Registers	BRAM B36/B18
1	15533	12956	16
2	15651	12670	9
3	59910	26892	32
4	60673	26918	32

A | Acronyms

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

B | Known Bugs

There are no known bugs at the moment.

C | Register File Contents

Legend

HW Hardware access rights

SW Software access rights

wo write-only

ro read-only

rw read-write

Table C.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 C.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 C.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 C.4: Control

Field	# Bits	Description & Encoding	Reset	HW	SW
p_rst_n	1	Active low HMC reset.	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
rx_token _count	1	Set the input buffer space in the RX block	RX_TOKEN_CNT	ro	rw
irtry_received _threshold	1	Set the amount of irtry packets to be received until an action is performed.	0x10	ro	rw
irtry_to _send	1	Set the amount 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	0x30	ro	rw
first _cube_ID	1	Set the Cube ID of the first HMC connected. Used in irtry packets	0	ro	rw

Table C.5: Other Counter

Field	# Bits	Description & Encoding	Reset	HW	SW
link_retries	32	Incremental 1-bit counter: How many link retries were 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

D | Directory Structure

```
openhmc
├── sim
├── rtl
│   ├── building_blocks
│   │   ├── counter
│   │   │   └── counter48.v
│   │   └── fifos
│   │       ├── async
│   │       │   └── async_fifo.v
│   │       ├── sync
│   │       │   ├── sync_fifo_reg_stage.v
│   │       │   ├── sync_fifo.v
│   │       │   └── sync_fifos.f
│   │       └── rams
│   │           └── ram.v
│   ├── hmc_controller
│   │   ├── crc
│   │   │   ├── crc_128bit_init.v
│   │   │   └── crc_accu.v
│   │   ├── register_file
│   │   │   └── hmc_controller_rf.v
│   │   ├── rx
│   │   │   ├── rx_descrambler.v
│   │   │   ├── rx_crc_compare.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_1
```

E | Revision History

1.0 First release

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

F List of Figures

G List of Tables

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 1.1. http://www.hybridmemorycube.org/. [last accessed 16-Aug-2014].

[3] ARM Limited. AMBA AXI4-Stream Protocol Specification v1.0. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0051a/index.html. [last accessed 16-Aug-2014].