Hybrid Memory Cube Specification 2.0

Hybrid Memory Cube with HMC-30G-VSR PHY

4GB 4H DRAM stack
8GB 8H DRAM stack

HMC Memory Features

- V_{DDM} = 1.1V ±0.033V
- V_{PP} = 1.8V ±0.125V
- 4GB configuration
  - 256 memory banks
- 8GB configuration
  - 512 memory banks
- Closed-bank memory architecture
- Built-in memory controller for each vault
  - Automatic refresh control over all temperatures
- Internal ECC data correction
- Advanced RAS features including data scrubbing
- Post-assembly repair capability
- In-field repair for ultimate reliability

HMC Interface Features

- V_{DD} = 0.9V ±0.027V
- AV_{DD} = 0.9V ±0.027V
- V_{DDK} = 1.8V (1.71–1.89V)
- 12.5 Gb/s, 15 Gb/s, 25 Gb/s, 28 Gb/s, or 30 Gb/s SerDes I/O interface
- Up to four 16-lane, full-duplex serialized links
  - Half-width link (8-lane) and quarter-width link (4-lane) configurations also supported
  - Up to 320 GB/s effective bandwidth
- Packet-based data/command interface
- Supports 16, 32, 48, 64, 80, 96, 112, 128, and 256 byte references per request
- Error detection (cyclic redundancy check [CRC]) for packets with automatic retry
- Power management supported per link
- Through-silicon via (TSV) technology
- Built-in self-test (BIST)
- JTAG interface (IEEE 1149.1-2001, 1149.6)
- I^2C interface up to 1 MHz
- SPI master interface

Options	Marking
• Configuration	
– 4GB cube (8Gb x 4H DRAM stack)	8G4
– 8GB cube (8Gb x 8H DRAM stack)	8G8
• BGA package (Pb-free)	
– 4-link (33mm x 34mm) 4GB	TBD
– 4-link (33mm x 34mm) 8GB	TBD
– 2-link (TBD) 4GB	TBD
– 2-link (TBD) 8GB	TBD
• Operating temperature	
– DRAM: 0°C ≤ T_J ≤ 105°C	None
– Logic: 0°C ≤ T_J ≤ 110°C	None
• PHY	
– HMC-30-VSR	-S30
• Logic revision	01
• DRAM revision	:A

Description

A hybrid memory cube (HMC) is a single package containing either four or eight DRAM die and one logic die, all stacked together using through-silicon via (TSV) technology.

Within each cube, memory is organized vertically; portions of each memory die are combined with the corresponding portions of the other memory die in the stack. Each grouping of memory partitions is combined with a corresponding controller within the logic die, forming what is referred to as a vault.

Contents

HMC Architecture 9
Logic Base Architecture 10
Pin Descriptions 12
Link Data Transmission 14
Logical Sub-Block of Physical Layer 15
Link Serialization 15
Scrambling and Descrambling 16
Training Sequence 17
Lane Run Length Limitation 17
Lane Reversal 18
Lane Polarity 18
Chaining 18
Power-On and Initialization 23
Power State Management 27
Link Layer 30
Transaction Layer 31
Memory Addressing 34
Memory Addressing Granularity 34
Memory Address-to-Link Mapping 35
Default Address Map Mode Tables 36
Address Mapping Mode Register 38
DRAM Addressing 38
Packet Length 38
Packet Flow Control 38
Tagging 39
Packet Integrity 40
Request Packets 41
Response Packets 42
Flow Packets 49
Poisoned Packets 49
Poisoned Packet Rules 49
Request Commands 50
READ and WRITE Request Commands 52
POSTED WRITE Request Commands 53
ATOMIC Request Commands 54
Addressing Atomic Operations 54
Arithmetic Atomics 55
Dual 8-Byte Signed Add Immediate 55
Single 16-Byte Signed Add Immediate 55
Dual 8-Byte Signed Add Immediate and Return 56
16-Byte Signed Add Immediate and Return 57
8-Byte Increment 58
Bitwise Atomics 59
8-Byte Bit Write 59
8-Byte Bit Write With Return 59
16-Byte Swap 60
Boolean Atomics 62
16-Byte AND 62
16-Byte NAND 63
16-Byte NOR 63

16-Byte OR 64
16-Byte XOR 64
Comparison Atomics 66
8-Byte Compare and Swap if Greater Than 66
8-Byte Compare and Swap if Less Than 67
8-Byte Equal To 67
16-Byte Compare and Swap if Greater Than 68
16-Byte Compare and Swap if Less Than 69
16-Byte Equal To 70
8-Byte Compare and Swap if Equal 71
16-Byte Compare and Swap if Zero 72
MODE READ and WRITE Request Commands 73
BIT WRITE Command 73
Response Commands 74
READ and MODE READ Response Commands 74
WRITE and MODE WRITE Response Commands 75
ERROR Response Command 75
Flow Commands 75
NULL Command 76
RETRY POINTER RETURN (PRET) Command 76
TOKEN RETURN (TRET) Command 76
INIT RETRY (IRTRY) Command 76
Valid Field Command Summary 76
Transaction Layer Initialization 77
Configuration and Status Registers 78
Link Retry 79
Retry Pointer Description 81
Link Master Retry Functions 82
Forward Retry Pointer Generation 83
Packet Sequence Number Generation 83
Forward Retry Pointer Reception and Embedding 83
Return Retry Pointer Reception 84
Link Master Sequences 84
StartRetry Sequence 84
LinkRetry Sequence 86
Link Slave Retry Functions 87
Packet CRC/Sequence Check 88
Error Abort Mode 88
IRTRY Packet Operation 88
Resumption of Normal Packet Stream after the Retry Sequence 90
Retry Pointer Loop Time 90
Link Flow Control During Retry 90
System Error Handling – Host Error Management 91
Retraining 92
Retraining a Link with High Error Rate 92
Warm Reset 93
Functional Characteristics 94
Packet Ordering and Data Consistency 94
Data Access Performance Considerations 95
Vault ECC and Reference Error Detection 96
Refresh 96
Scrubbing 96

Response Open Loop Mode 97
JTAG Interface 97
Disabling the JTAG Interface 97
I2C Interface 99
Serial Peripheral Interface 99
HMC-30G-VSR Electrical Specifications 99
Absolute Maximum Ratings 100
Supply Current 100
DC Operating Conditions 100
HMC-30G-VSR Side-Band Electrical Specifications 100
JTAG Electrical Parameters 100
I2C Electrical Parameters 101
SPI Electrical Parameters 102
HMC-30G-VSR Physical Link Specifications 103
Physical Link Electrical Interface 103
Equalization Schemes 103
Link Bit Rate and Link Bit Error Rate 104
High Speed Signaling Parameters 105
Supported Channels 110
Non-High Speed Link Parameters 114
Package Dimensions 116
Appendix A: Glossary of Terms 122

List of Figures

Figure 1: Example HMC Organization 9
Figure 2: HMC Block Diagram Example Implementation (4-link HMC configuration) 10
Figure 3: Link Data Transmission Implementation Example 14
Figure 4: Scrambler and Descrambler Paths from Requester to Responder 16
Figure 5: Example of a Chained Topology 19
Figure 6: Example of Star Topology 20
Figure 7: Example of Topology with Multiple Hosts 21
Figure 8: Example of Two-Host Expanded Star Topology 22
Figure 9: HMC Initialization Flowchart 26
Figure 10: Initialization Timing 27
Figure 11: Sleep Mode Entry and Exit (Single Link Only) 29
Figure 12: Simultaneous Transition of Four Host Links to Sleep Mode, Entry into Down Mode and Return to Active Mode (Single HMC, Four Link Example) 30
Figure 13: Packet Layouts 32
Figure 14: Request Packet Header Layout 41
Figure 15: Request Packet Tail Layout 41
Figure 16: Response Packet Header Layout 42
Figure 17: Response Packet Tail Layout 43
Figure 18: Configuration and Status Register Access Through ERI 79
Figure 19: Implementation Example of Link Retry Block Diagram 80
Figure 20: Implementation Example of a Retry Buffer 82
Figure 21: Host Recovery after Link Retry Fails 92
Figure 22: Warm Reset Flow Chart 94
Figure 23: SPI Application Circuit 99
Figure 24: TAP Timing 101
Figure 25: Digital Waveform with Pre-Tap (C(-1) and Post-Tap (C(+1)) On 104
Figure 26: TX and RX Return Loss 109
Figure 27: Receiver Sinusoidal Jitter Tolerance 110
Figure 28: Example of HMC-30G-VSR Implementation 111
Figure 29: Examples of Channel Insertion Loss and MIN/MAX Limits 111
Figure 30: Determining Insertion Loss Deviation 112
Figure 31: Channel Crosstalk Specification 112
Figure 32: Channel Return Loss 113
Figure 33: HMC Package Drawing (4-Link HMC-30G-VSR Device) — 1.0mm Ball-Pitch 116
Figure 34: HMC Package Drawing (2-Link HMC-30G-VSR Device) — 1.0mm Ball-Pitch 117
Figure 35: HMC Package Drawing (2-Link HMC-30G-VSR Device) — 0.65mm Ball-Pitch 118
Figure 36: Connector Map — 1.0mm Ball-Pitch 4-Link Package 119
Figure 37: Connector Map — 1.0mm Ball-Pitch 2-Link Package 120
Figure 38: Connector Map — 0.65mm Ball-Pitch 2-Link Package 121

List of Tables

Table 1: HMC Configurations 11
Table 2: Pin Descriptions 12
Table 3: Unit Interval FLIT Bit Positions for Full-Width Configuration (16 Lanes) 15
Table 4: Unit Interval FLIT Bit Positions for Half-Width Configuration (8 Lanes) 15
Table 5: Unit Interval FLIT Bit Positions for Quarter-Width Configuration (4 Lanes) 16
Table 6: HMC Scramble and Descramble Modes 16
Table 7: TS1 Definition 17
Table 8: Link Power States and Conditions 28
Table 9: Single FLIT Example: TRET Packet 33
Table 10: Mulit-FLIT Example: 2ADD8 Request Packet 33
Table 11: Addressing Definitions 35
Table 12: Default Address Map Mode Table – 4GB 36
Table 13: Default Address Map Mode Table – 8GB 37
Table 14: Request Packet Header Fields 41
Table 15: Request Packet Tail Fields 42
Table 16: Response Packet Header Fields 42
Table 17: Response Packet Tail Fields 43
Table 18: RTC Encoding 44
Table 19: ERRSTAT[6:0] Bit Definitions – No Errors 44
Table 20: ERRSTAT[6:0] Bit Definitions – Warnings 45
Table 21: ERRSTAT[6:0] Bit Definitions – DRAM Errors 45
Table 22: ERRSTAT[6:0] Bit Definitions – Link Errors 46
Table 23: ERRSTAT[6:0] Bit Definitions – Protocol Errors 46
Table 24: ERRSTAT[6:0] Bit Definitions – Vault Critical Errors 47
Table 25: ERRSTAT[6:0] Bit Definitions – Fatal Errors 48
Table 26: Transaction Layer – Request Commands 50
Table 27: Atomic Quick Reference 54
Table 28: Address Field in Request Header 54
Table 29: Dual 8-Byte Signed Add Immediate 55
Table 30: Operands from DRAM 55
Table 31: Immediate Operands Included in Atomic Request Data Payload 55
Table 32: Single 16-Byte Signed Add Immediate 55
Table 33: Operands from DRAM 56
Table 34: Immediate Operand Included in Atomic Request Data Payload 56
Table 35: Dual 8-Byte Signed Add Immediate and Return 56
Table 36: Operands from DRAM 56
Table 37: Immediate Operands Included in Atomic Request Data Payload 57
Table 38: Operands Included in Atomic Response Data Payload 57
Table 39: 16-Byte Signed Add Immediate and Return 57
Table 40: Operands from DRAM 57
Table 41: Immediate Operand Included in Atomic Request Data Payload 57
Table 42: Operand Included in Atomic Response Data Payload 58
Table 43: 8-Byte Increment 58
Table 44: Operands from DRAM 58
Table 45: 8-Byte Bit Write 59
Table 46: Operands from DRAM 59
Table 47: Immediate Operand Included in Atomic Request Data Payload 59
Table 48: 8-Byte Bit Write With Return 60
Table 49: Operands from DRAM 60
Table 50: Immediate Operand Included in Atomic Request Data Payload 60

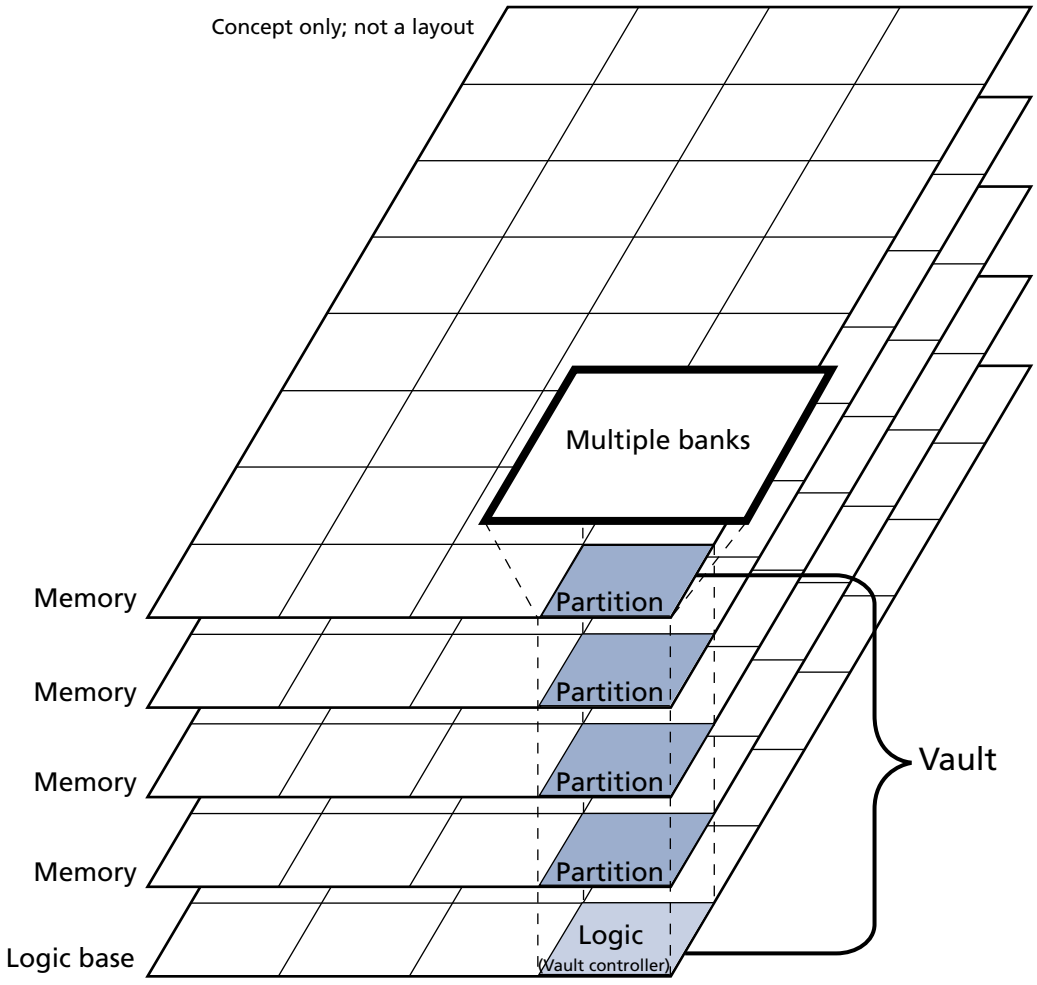
Table 51: Operand Included in Atomic Response Data Payload 60
Table 52: 16-Byte Swap 61
Table 53: Operands from DRAM 61
Table 54: Immediate Operand Included in Atomic Request Data Payload 61
Table 55: Operand Included in Atomic Response Data Payload 61
Table 56: Operands from DRAM 62
Table 57: Immediate Operand Included in Atomic Request Data Payload 62
Table 58: Operand Included in Atomic Response Data Payload 62
Table 59: 16-Byte AND 62
Table 60: AND Bitwise Truth Table 63
Table 61: 16-Byte NAND 63
Table 62: NAND Bitwise Truth Table 63
Table 63: 16-Byte NOR 63
Table 64: NOR Bitwise Truth Table 64
Table 65: 16-Byte OR 64
Table 66: OR Bitwise Truth Table 64
Table 67: 16-Byte XOR 64
Table 68: XOR Bitwise Truth Table 65
Table 69: 8-Byte Compare and Swap if Greater Than 66
Table 70: Operands from DRAM 66
Table 71: Immediate Operands Included in Atomic Request Data Payload 66
Table 72: Operands Included in Atomic Response Data Payload 66
Table 73: 8-Byte Compare and Swap if Less Than 67
Table 74: Operands from DRAM 67
Table 75: Immediate Operands Included in Atomic Request Data Payload 67
Table 76: Operands Included in Atomic Response Data Payload 67
Table 77: 8-Byte Equal To 68
Table 78: Operands from DRAM 68
Table 79: Immediate Operands Included in Atomic Request Data Payload 68
Table 80: 16-Byte Compare and Swap if Greater Than 68
Table 81: Operands from DRAM 69
Table 82: Immediate Operand Included in Atomic Request Data Payload 69
Table 83: Operand Included in Atomic Response Data Payload 69
Table 84: 16-Byte Compare and Swap if Less Than 69
Table 85: Operands from DRAM 70
Table 86: Immediate Operand Included in Atomic Request Data Payload 70
Table 87: Operand Included in Atomic Response Data Payload 70
Table 88: 16-Byte Equal To 70
Table 89: Operands from DRAM 70
Table 90: Immediate Operand Included in Atomic Request Data Payload 71
Table 91: 8-Byte Compare and Swap if Equal 71
Table 92: Operands from DRAM 71
Table 93: Immediate Operands Included in Atomic Request Data Payload 71
Table 94: Operands Included in Atomic Response Data Payload 72
Table 95: 16-Byte Compare and Swap if Zero 72
Table 96: Operands from DRAM 72
Table 97: Immediate Operand Included in Atomic Request Data Payload 72
Table 98: Operand Included in Atomic Response Data Payload 72
Table 99: Mode Request Addressing 73
Table 100: Valid Data Bytes 73
Table 101: Request Packet Bytes to be Written 74
Table 102: Transaction Layer – Response Commands 74

Table 103: Flow Commands 76
Table 104: Valid Field and Command Summary Table 76
Table 105: Examples of Host Error Management 91
Table 106: Instruction Codes 97
Table 107: DC Electrical Characteristics 100
Table 108: JTAG Voltage and Timing Parameters 100
Table 109: I^2C Voltage Parameters 101
Table 110: SPI Voltage Parameters 102
Table 111: Synchronous Link Bit Rate Specifications 105
Table 112: TX Signaling Parameters 105
Table 113: RX Signaling Parameters 108
Table 114: Informative Channel Parameters 113
Table 115: Initialization Timing Parameters 114
Table 116: Link Power Management Parameters 114
Table 117: Reference Clock Parameters 115
Table 118: Functional Pin List — 1.0mm Ball-Pitch 4-Link Package 119
Table 119: Functional Pin List — 1.0mm Ball-Pitch 2-Link Package 120
Table 120: Functional Pin List — 0.65mm Ball-Pitch 2-Link Package 121
Table 121: Glossary of Terms 122

HMC Architecture

An HMC consists of a single package containing multiple memory die and one logic die, stacked together, using through-silicon via (TSV) technology (see the example below). Within an HMC, memory is organized into vaults. Each vault is functionally and operationally independent.

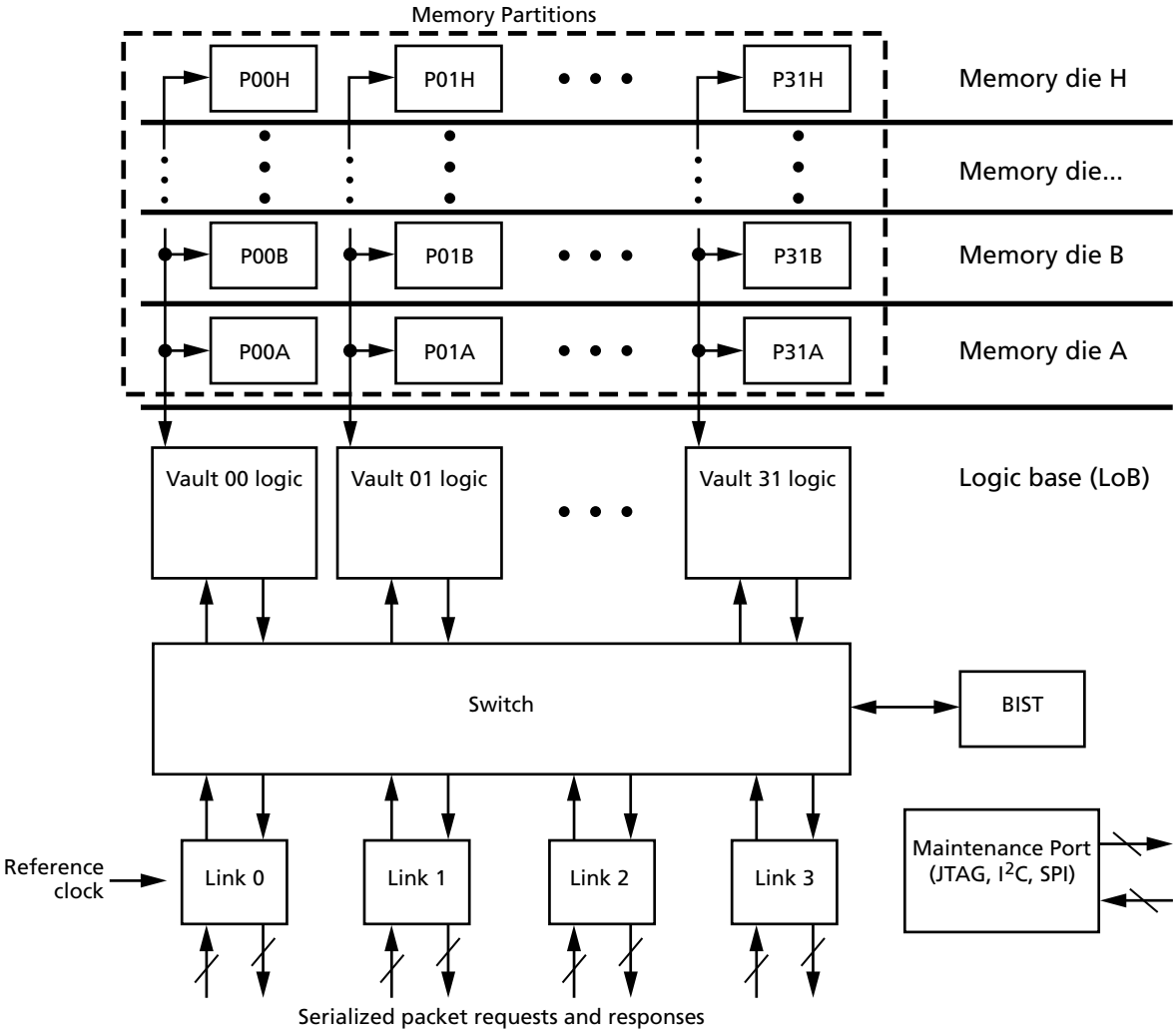
Figure 1: Example HMC Organization



Each vault has a memory controller (called a vault controller) in the logic base that manages all memory reference operations within that vault. Each vault controller determines its own timing requirements. Refresh operations are controlled by the vault controller, eliminating this function from the host memory controller.

Each vault controller may have a queue that is used to buffer references for that vault's memory. The vault controller may execute references within that queue based on need rather than order of arrival. Therefore, responses from vault operations back to the external serial I/O links will be out of order. However, requests from a single external serial link to the same vault/bank address are executed in order. Requests from different external serial links to the same vault/bank address are not guaranteed to be executed in a specific order and must be managed by the host controller.

Figure 2: HMC Block Diagram Example Implementation (4-link HMC configuration)



Logic Base Architecture

The logic base manages multiple functions for the HMC (see example implementation in the following figure):

- All HMC I/O, implemented as multiple serialized, full duplex links
- Memory control for each vault; Data routing and buffering between I/O links and vaults
- Consolidated functions removed from the memory die to the controller
- Mode and configuration registers
- BIST for the memory and logic layer
- Test access port compliant to JTAG IEEE 1149.1-2001, 1149.6
- Some spare resources enabling field recovery from some internal hard faults.

The collective internally available bandwidth from all of the vaults is made accessible to the I/O links. A crossbar switch is an example implementation of how this could be achieved. The external I/O links consist of multiple serial links, each with a default of 16

input lanes and 16 output lanes for full duplex operation. Also supported are a half-width configuration, comprised of 8 input lanes and 8 output lanes per link, and a quarter-width configuration, comprised of 4 input lanes and 4 output lanes per link. HMC-30G-VSR also supports asymmetric links: Non-matching or different TX versus RX link sizes within the boundaries of 1/4, 1/2, and full link options. Each lane direction in each link has additional power-down signals for power management. The following configurations are those supported within this specification.

Table 1: HMC Configurations

	Configurations
Number of links in package	2, 4
Link lane speed (Gb/s)	12.5,15, 25, 28, 30
Link width	Full, half, quarter
Memory density	4GB, 8GB
Number of vaults	32
Memory banks	4GB: 256 banks 8GB: 512 banks
Maximum aggregate link bandwidth[1]	480 GB/s (3.84 Tb/s)
Maximum DRAM data bandwidth	320 GB/s (2.56 Tb/s)
Maximum vault data bandwidth	10 GB/s (80 Gb/s)

Notes:
1. Link bandwidth includes data as well as packet header and tail. Full-width (16 input and 16 output lanes) configuration is assumed.
2. Asymmetric link interface is supported between TX and RX.

All in-band communication across a link is packetized. Packets specify single, complete operations. For example, a READ reference of 64 data bytes. There is no specific timing associated with memory requests, and responses can generally be returned in a different order than requests were made because there are multiple independent vaults, each executing independently of the others. The vaults generally reorder their internal requests to optimize bandwidths and to reduce average latencies.

Pin Descriptions

Table 2: Pin Descriptions

"P" pins are the true side of differential signals; "N" pins denote the complement side

Signal Pin Symbol	Type	Description
Link Interface[1]		
LxRXP[n:0]	Input	Receiving lanes
LxRXN[n:0]		
LxTXP[n:0]	Output	Transmitting lanes
LxTXN[n:0]		
LxRXPS	Input	Power-reduction input
LxTXPS	Output	Power-reduction output
FERR_N	Output	Fatal error indicator. HMC drives LOW if fatal error occurs, otherwise the pull-down is turned off and floats HIGH. FERR_N operates in V_{DDK} domain. Requires external pull-up resistor. FERR_N can be wire-OR'd among multiple HMC devices. $R_{ON} = 300\Omega$; R_{OFF} = 10kΩ
Clocks and Reset		
REFCLKP	Input	Reference clock for all links, logic core fabric, switch and vault controllers
REFCLKN		
REFCLKSEL	Input	Determines on-die termination for REFLCLKP/N. Set DC HIGH (≥V_{DDK} - 0.5V) if REFCLKP/N are AC-coupled. Set DC LOW (≤V_{SS} + 0.5V) if REFCLKP/N are DC-coupled.
P_RST_N	Input	System reset. Active LOW CMOS input referenced to V_{SS}. The P_RST_N is a rail-to-rail signal with DC HIGH ≥ (V_{DDK} - 0.5V) and DC LOW ≤ (V_{SS} + 0.5V).
JTAG Interface		
TMS	Input	JTAG test mode select
TCK	Input	JTAG test mode clock
TDI	Input	JTAG test data-in
TDO	Output	JTAG test data-out
TRST_N	Input	JTAG test reset (active LOW)
I^2C Interface		
SCL	Input	I^2C clock
SDA	Bidirectional	I^2C data
SPI Master Interface		
SCK	Output	Serial clock
SDO	Output	Serial data output
SDI	Input	Serial data input
SS_N	Output	Chip select or slave select (active LOW)
Bootstrapping Pins		

Table 2: Pin Descriptions (Continued)

"P" pins are the true side of differential signals; "N" pins denote the complement side

Signal Pin Symbol	Type	Description	
CUB[2:0]	Input	User-assigned HMC identification to enable the host to map the unique location of an HMC in a system. The I^2C slave address is equivalent to the state of the CUB[2:0] bits. This value will also be used as the default value for the CUB field of the Request packet header. DC HIGH ≥ (V_{DDK} - 0.5V) and DC LOW ≤ (V_{SS} + 0.5V).	
REFCLK_BOOT[1:0]	Input	Bootstrapping pins that enable autonomous PLL configuration. Tie pins DC HIGH or DC LOW to match reference clock frequency being used. DC HIGH ≥ (V_{DDK} - 0.5V) and DC LOW ≤ (V_{SS} + 0.5V)	
		REFCLK_BOOT[1:0]	**Reference Clock Frequency**
		00	125 MHz
		01	156.25 MHz
		10	166.67 MHz
		11	312.25 MHz
Reserved Pins			
DNU	–	Do not use; must not be connected on the board	
EXTRESP/EXTRESN	Input	Precision resistor pins used for termination impedance auto-calibration; vendor-specific.	
MCL	Input	Vendor-reserved for specific test, must connect LOW	
GPIO	Bidirectional	Vendor-reserved	
Pending	Bidirectional	Vendor-reserved (Pending definition for test and contingency.)	
Supply Pins			
V_{DD}	Supply	Logic core supply: 0.9V ±0.027V	
AV_{DD}	Supply	PHY core supply: 0.9V ±0.027V	
V_{DDM}	Supply	DRAM supply: 1.1V ±0.03V	
V_{PP}	Supply	DRAM wordline boost supply: 1.8V ±0.090V	
V_{DDK}	Supply	GPIO, side-band control supply: 1.8V ±0.090V	
AV_{SS}	Supply	Analog PHY Ground	
V_{SS}	Supply	Ground	

Note: 1. x represents specific link number and will be 0 to 3 depending upon configuration.

Link Data Transmission

Commands and data are transmitted in both directions across the link using a packet-based protocol where the packets consist of 128-bit flow units called “FLITs.” These FLITs are serialized, transmitted across the physical lanes of the link, then re-assembled at the receiving end of the link. Three conceptual layers handle packet transfers:

- The physical layer handles serialization, transmission, and deserialization.
- The link layer provides the low-level handling of the packets at each end of the link.
- The transaction layer provides the definition of the packets, the fields within the packets, and the packet verification and retry functions of the link.

Two logical blocks exist within the link layer and transaction layer:

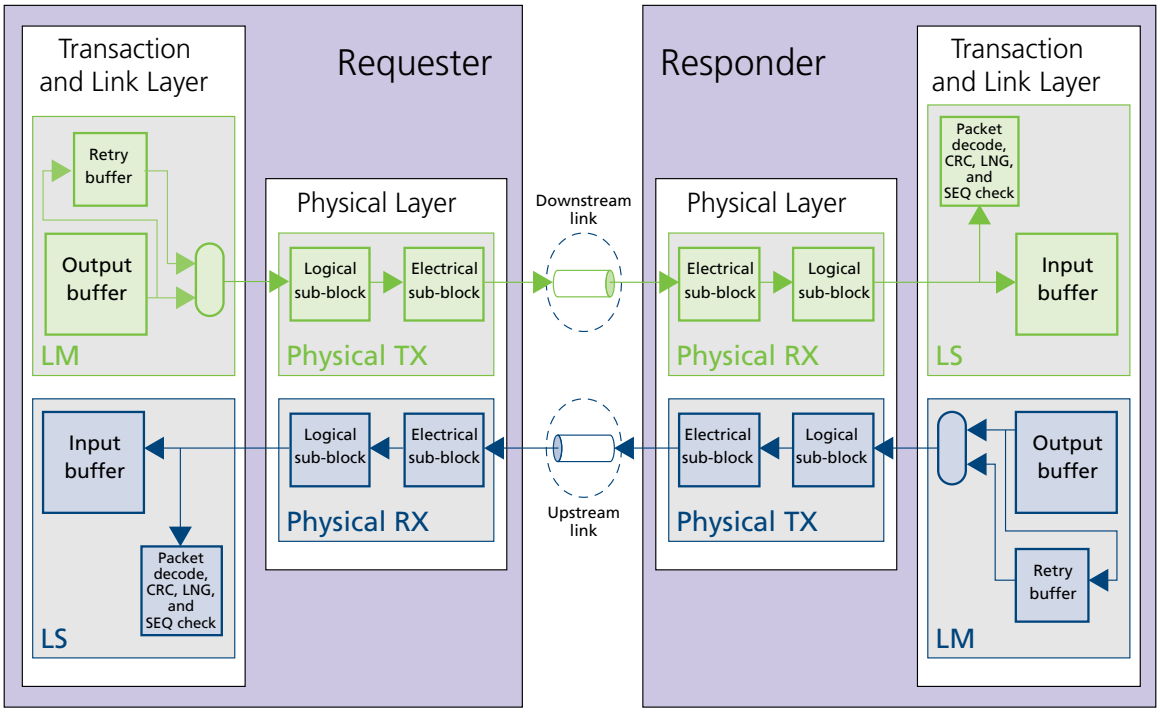
- The link master, is the logical source of the link where the packets are generated and the transmission of the FLITs is initiated.
- The link slave, is the logical destination of the link where the FLITS of the packets are received, parsed, evaluated, and then forwarded internally.

The nomenclature below is used throughout the specification to distinguish the direction of transmission between devices on opposite ends of a link. These terms are applicable to both host-to-cube and cube-to-cube configurations.

Requester: Represents either a host processor or an HMC link configured as a pass-thru link. A requester transmits packets downstream to the responder.

Responder: Represents an HMC link configured as a host link. A responder transmits packets upstream to the requester.

Figure 3: Link Data Transmission Implementation Example



Logical Sub-Block of Physical Layer

The transfer of information across the links consists of 128-bit FLITs. Each FLIT takes the following path through the serial link:

1. 128-bit FLITs are generated by the link master and sent in parallel to the transmitting logical sub-block in the physical layer.
2. The transmitting logical sub-block serializes each FLIT and drives it across the link interface in a bit-serial form on each of the lanes.
3. The receiving logical sub-block deserializes each lane and recreates the 128-bit parallel FLIT. Receive deserializer FLIT alignment is achieved during link initialization.
4. The 128-bit parallel FLIT is sent to the link layer link slave section.

Link Serialization

Link serialization occurs with the least-significant portion of the FLIT traversing across the lanes of the link first. During one unit interval (UI) a single bit is transferred across each lane of the link. For the full-width configuration, 16 bits are transferred simultaneously during the UI, so it takes 8UIs to transfer the entire 128-bit FLIT. For the half-width configuration, 8 bits are transferred simultaneously, taking 16UIs to transfer a single FLIT. The following table shows the relationship of the FLIT bit positions to the lanes during each UI for full-width, half-width, and quarter-width configurations.

Table 3: Unit Interval FLIT Bit Positions for Full-Width Configuration (16 Lanes)

	Lane															
UI	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
2	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
7	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

Table 4: Unit Interval FLIT Bit Positions for Half-Width Configuration (8 Lanes)

	Lane							
UI	7	6	5	4	3	2	1	0
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
…	…	…	…	…	…	…	…	…
15	127	126	125	124	123	122	121	120

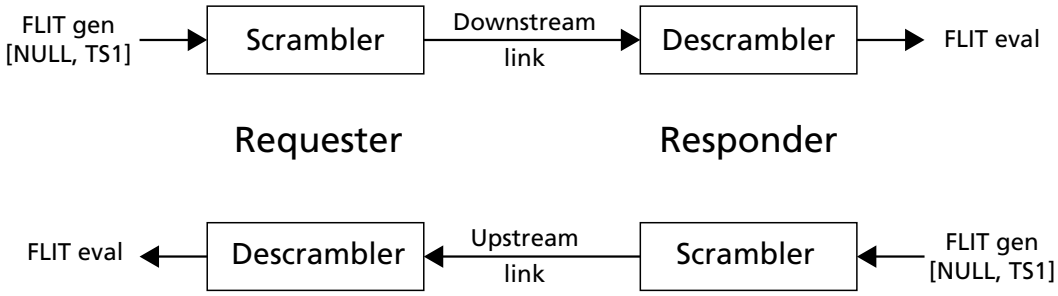
Table 5: Unit Interval FLIT Bit Positions for Quarter-Width Configuration (4 Lanes)

UI	Lane			
	3	2	1	0
0	3	2	1	0
1	7	6	5	4
2	11	10	9	8
…	…	…	…	…
31	127	126	125	124

Scrambling and Descrambling

HMC-30G-VSR implements a scrambler to provide serial data with adequate edge density for AC coupling and data driven clock recovery. HMC-30G-VSR will use PRBS31, but because of the option for backward compatibility, 30G-VSR will also support PRBS15 when using revision 1.x protocol.

Figure 4: Scrambler and Descrambler Paths from Requester to Responder



HMC-30G-VSR supports two scramble and decramble modes. When in revision 1.0 protocol compatibility mode, the protocol will use revision 1.0 PRBS15 scramble/descramble. When using the revision 2.0 or later protocol, the protocol will use the revision 2.x PRBS31 scrambler/descrambler format. For HMC-compatible operation with protocol revision 1.x, refer to the revision 1.x HMC specification; this document will focus on the 2.x specification.

Use the following polynomial for the scrambler and descrambler logic: 1 + x -28 + x -31. To scramble the data, XOR the LSB of the linear feedback shift register (LFSR) with the data, as shown in . Seeding values vary per lane, as shown in . Designers may wish to include a mode register bit to bypass scrambling for debug purposes.

Table 6: HMC Scramble and Descramble Modes

Descriptor	Scrambler Equation	Protocol Revision	Device
PRBS15	$1 + x^{-14} + x^{-15}$	1.x	HMC-15G-SR or HMC-30G-VSR
PRBS31 (default)	$1 + x^{-28} + x^{-31}$	2.x	HMC-30G-VSR

Training Sequence

When a training sequence is initiated, a TS1 16-bit character is transmitted which includes lane and sequence field. Please refer to the table below for the 16 bit TS1 character definition. This occurs in both link directions: from requester-to-responder (host-to-HMC) and from responder-to-requester (HMC-to-host). The sequence field should increment with each subsequent 16-bit TS1 character transmitted. The HMC-30G-VSR device requires this behavior when training the link from host-to-HMC. With the HMC-30G-VSR device there are new programmable options that allow it to increment the sequence field on every TS1 transmit as well as every other or every fourth TS1 character transmitted from the HMC to the host. This provides a longer TS1 pattern repetition sequence for hosts that have receiver designs that require greater lane de-skew capability.

Table 7: TS1 Definition

TS1 Bit Position															
(Sent Last) 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	(Sent First) 0
1	1	1	1	0	0	0	0	L	L	L	L	S	S	S	S
0xF				0x0				Dependant on lane[1]				Sequence[2]			

Notes:
1. Lane field = 0xC for lane 15 (or lane 7 in half-width configuration or lane 3 in quarter-width configuration; 0x3 for lane 0; and 0x5 for all other lanes.
2. Sequence[3:0] is a 4-bit counter that increments from 0 to 15 with each successive TS1 sent on a lane. Sequence value will roll back to 0 after 15 is reached.
3. The half-width link configuration uses lanes 0–7 or 8–15.
4. The quarter-width link configuration uses lanes 0–3, 4–7, 8–11, or 12–15.
5. The 2-byte TS1 character establishes the framing that identifies byte pairs that belong to the same FLIT after training completes in half-width mode. The transition from sending TS1s to sending NULL FLITs can occur on any byte boundary within the entire 32-byte TS1 sequence, including mid-TS1 character, so this transition boundary to NULL bytes should NOT be used to infer the FLIT boundary in half-width configuration.
6. Consecutive TS1 characters that share the same Sequence[3:1] value identify post-training FLIT boundaries for quarter-width mode.
7. The transition from sending TS1s to sending NULL FLITs can occur on any byte boundary within the entire 32-byte TS1 sequence, including mid-TS1 character; therefore, this transition boundary to NULL bytes should NOT be used to infer the FLIT boundary in half-width configuration.

Lane Run Length Limitation

For each HMC RX lane, the scrambled data pattern must meet a run length limit of 100UI. This is the maximum number of consecutive identical digits allowed. A run of 101 or more consecutive ones (or zeroes) must not be generated by the transmitter on any of the downstream lanes at any time. This restriction is in place so that the clock recovery circuit at each HMC RX lane meets its minimum required transition density to assure correct data alignment. This restriction does not require a particular implementation of the transmitter logic, as long as the run length limit is met at the HMC RX lane inputs.

Although a given host may not have a run length limit on the scrambled upstream data, the HMC is designed to meet run length limitations of 100UI on each lane at all times. This is implemented through the HMC's ability to monitor the scrambled data at each TX lane. If the scrambled TX data on any lane exceeds 100 consecutive digits, the TX scramble logic forces at least one transition. Forcing a transition corrupts the upstream response and results in a link retry (see Link Retry). The probability of such an event is approximately 2^{-96} for random payload data. The HMC run length limitation feature may be disabled within the mode register.

Lane Reversal

In order to accommodate different external link routing topologies, the RX logical sub-block has the capability to reverse the lane bit number assignment. Lane reversal at the HMC is detected during initialization when receiving the TS1 training sequence and is automatically compensated for by logic in each receiving logical sub-block that reassigns external Lane 0 to connect to Lane 15 internally, Lane 1 to connect to Lane 14 internally, and so on. Once lane ordering is detected and set by the HMC logical sub-block, it remains unchanged thereafter. The lane reversal mode does not have to be the same for both directions of the link. An HMC component may have one or more links with lane reversal enabled at their respective receivers. The host is responsible for any implementation of lane reversal from the upstream lanes.

Lane Polarity

In order to accommodate different external Link routing topologies, the RX logical sub-block has the capability to logically invert the received data on a lane-by-lane basis. The polarity is determined during link initialization and remains unchanged thereafter. The training sequence is used to infer polarity at the HMC receiver. The requester is responsible for any implementation of lane polarity inversion from upstream lanes.

Chaining

Multiple HMC devices may be chained together to increase the total memory capacity available to a host. A network of up to eight HMC devices and 4 host source links is supported. Each HMC in the network is identified through the value in its CUB field, located within the request packet header. The host processor must load routing configuration information into each HMC. This routing information enables each HMC to use the CUB field to route request packets to their destination.

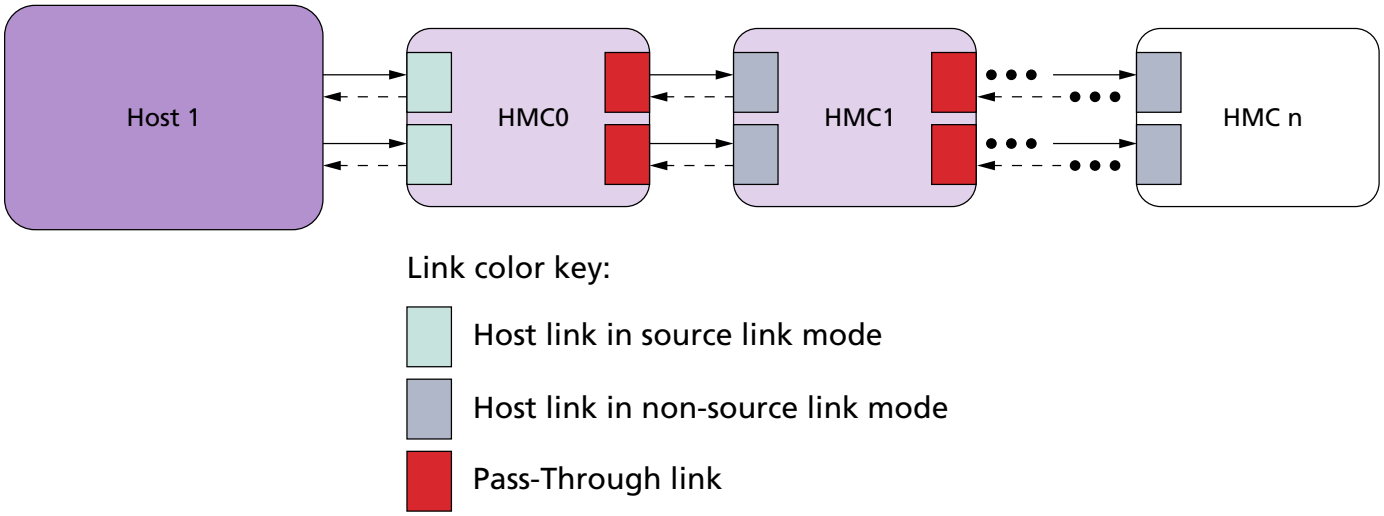
Each HMC link in the cube network is configured as either a host link or a pass-through link, depending upon its position within the topology. See Figure 5 and Figure 6 for illustration. A host link uses its link slave to receive request packets and its link master to transmit response packets. After receiving a request packet, the host link will either propagate the packet to its own internal vault destination (if the value in the CUB field matches its programmed cube ID) or forward it towards its destination in another HMC via a link configured as a pass-through link. In the case of a malformed request packet whereby the CUB field of the packet does not indicate an existing CUBE ID number in the chain, the request will not be executed, and a response will be returned (if not posted) indicating an error.

A pass-through link uses its link master to transmit the request packet towards its destination cube, and its link slave to receive response packets destined for the host processor.

An HMC link connected directly to the host processor must be configured as a host link in source mode. The link slave of the host link in source mode has the responsibility to generate and insert a unique value into the source link identifier (SLID) field within the tail of each request packet. The unique SLID value is used to identify the source link for response routing. The SLID value does not serve any function within the request packet other than to traverse the cube network to its destination vault where it is then inserted into the header of the corresponding response packet. The host processor must load routing configuration information into each HMC. This routing information enables each HMC to use the SLID value to route response packets to their destination. Only a host link in source mode will generate a SLID for each request packet. On the opposite side of a pass-through link is a host link that is NOT in source mode. This host link operates with the same characteristics as the host link in source mode except that it does not generate and insert a new value into the SLID field within a request packet. All link slaves in pass-through mode use the SLID value generated by the host link in source mode for response routing purposes only. The SLID fields within the request packet tail and the response packet header are considered "Don't Care" by the host processor. See Figure 5 for supported multi-cube topologies. Contact Micron for guidance regarding feasibility of all other topologies.

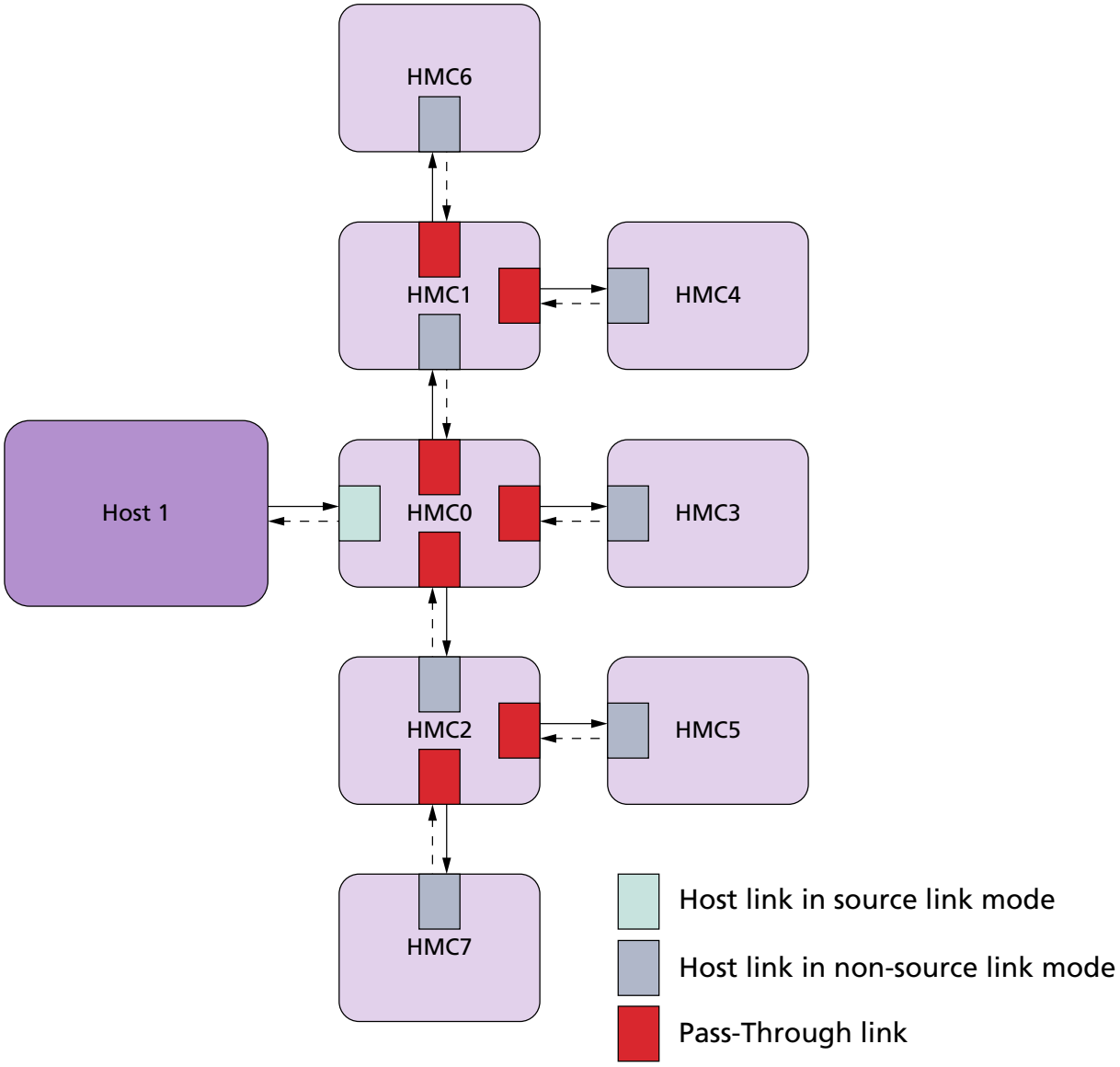
In the following figures, the link arrows show the direction of requests from the host(s). Responses will travel in the opposite direction on the same link.

Figure 5: Example of a Chained Topology



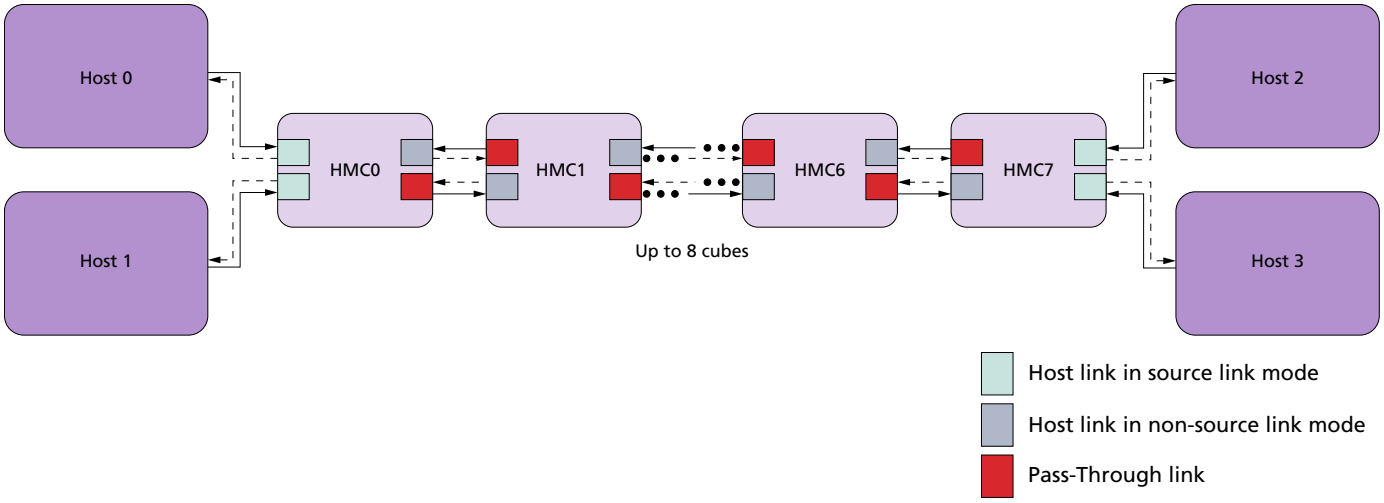
Notes:
1. For this topology, n ≤7.
2. The solid arrow indicates the command request flow; the dashed arrow indicates the response flow.

Figure 6: Example of Star Topology



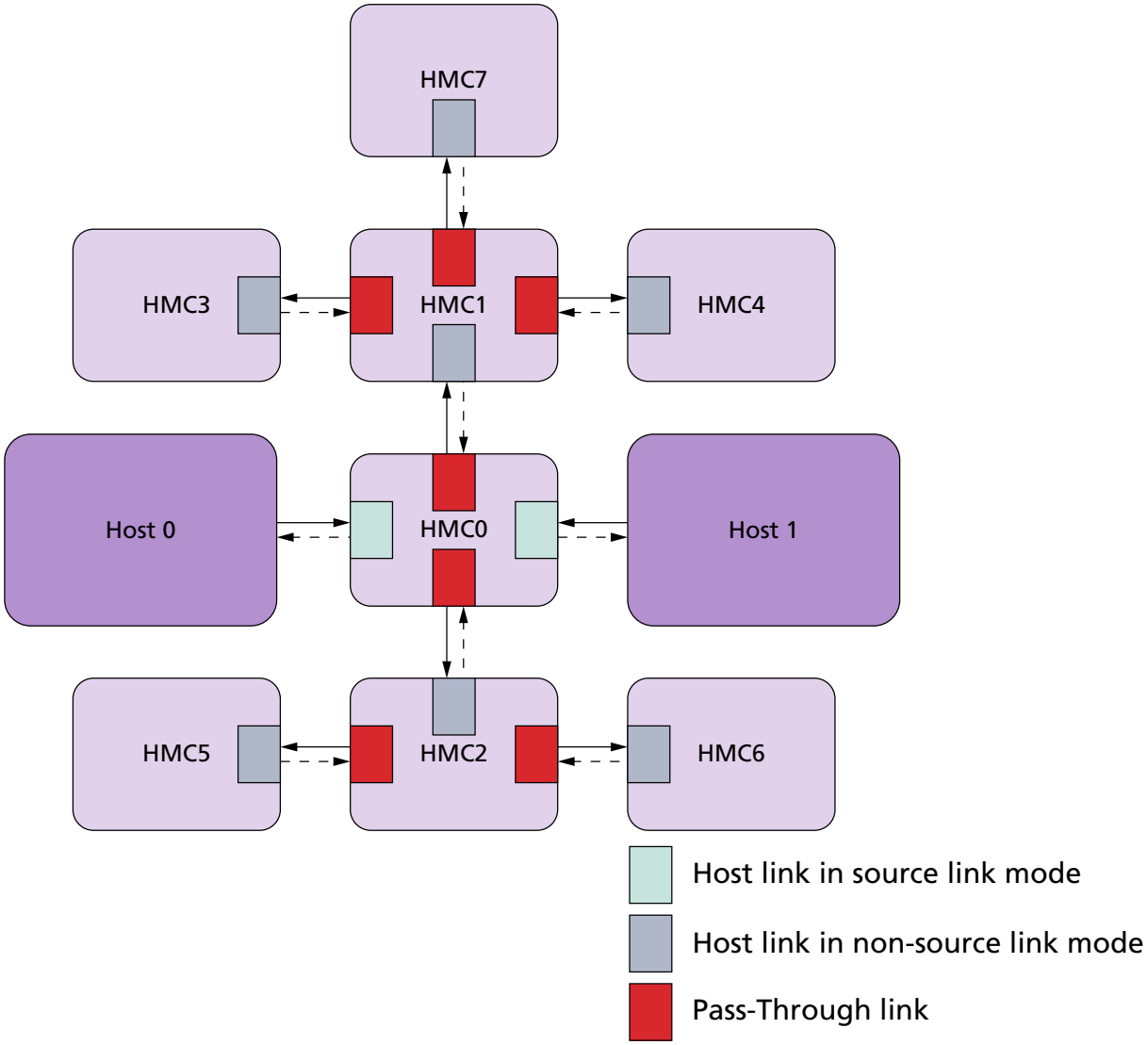
Note: 1. The solid arrow indicates the command request flow; the dashed arrow indicates the response flow.

Figure 7: Example of Topology with Multiple Hosts



Note: 1. The solid arrow indicates the command request flow; the dashed arrow indicates the response flow.

Figure 8: Example of Two-Host Expanded Star Topology



Note: 1. The solid arrow indicates the command request flow; the dashed arrow indicates the response flow.

Power-On and Initialization

The HMC must be powered on and initialized in a predefined manner. All timing parameters specified in the following sequence can be found in the Initialization Timing Parameters table. Registers indicating the status of the HMC during initialization can be accessed via the I^2C or JTAG bus. All links on an HMC may be initialized in parallel or independently. The steps in this section referring to the initialization of a single link are applicable to all links of an HMC being initialized at the same time. The HMC Initialization Flowchart that follows identifies the steps from an unpowered HMC device to a fully functional link/protocol communication between the host and HMC. This includes protocol and links in chained configurations.

The following are the major steps in the power-on and initialization sequence. Each step will be more fully detailed below.

- Power-On
- Reset/Clear Initialization
- Load HMC Device Global Settings
- Load Phy Configuration
- Link/Protocol Initialization (Phy and Protocol Training)

The following sequence is required for power-on and initialization of the host and HMC:

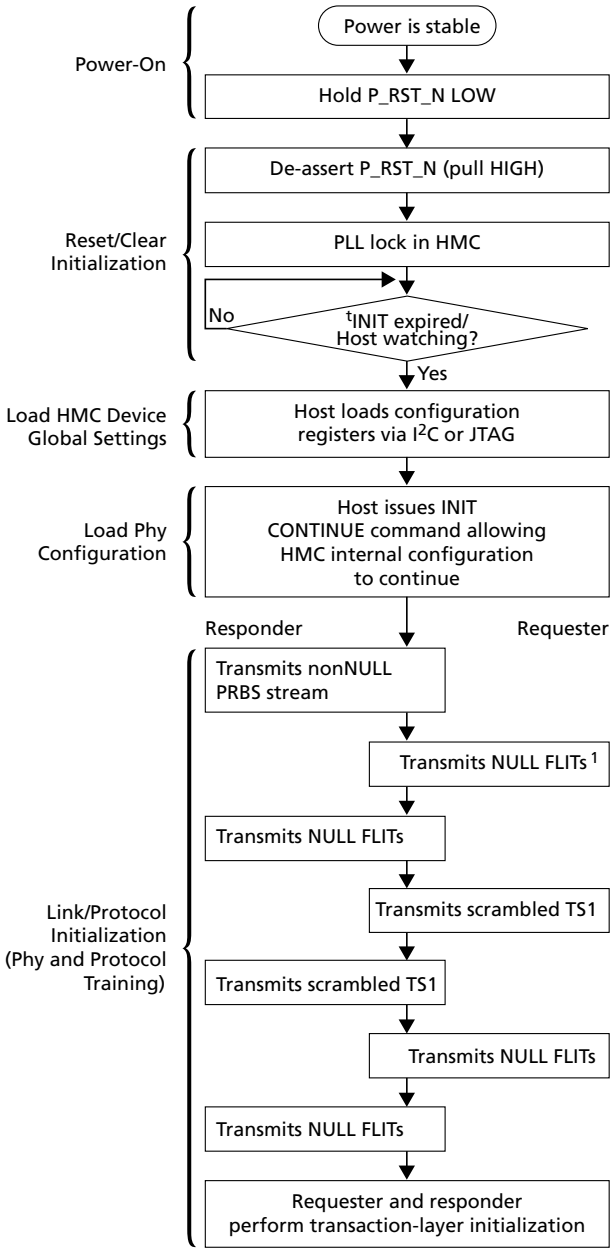
1. Power-On
    a. Apply power (all supplies). V_{PP} must ramp to full rail before the V_{DDM} ramp begins. All supplies must ramp to their respective minimum DC levels within ^{t}DD, but supply slew rates must not exceed V_{DVDT}.
    b. Maintain P_RST_N LOW, $0.2 \times V_{DDK}$, during power ramp to ensure that outputs remain disabled (pulled HIGH or pulled LOW) and on-die termination (ODT) is off. Link transmit signals (LxTXP[n:0] and LxTXN[n:0]) will remain pulled HIGH or pulled LOW until Step 5 below. All other HMC inputs can be undefined during the power ramp with the exception of TRST_N, which must be held LOW. TRST_N will track the subsequent transition of P_RST_N if using the JTAG port.
2. Reset/Clear Initialization
    a. After the voltage supplies and reference clocks (REFCLKP and REFCLKN) are within their specified operating tolerance, P_RST_N must be LOW for at least ^{t}RST to begin the initialization process. The LxRXPS signal for each link must be set for at least ^{t}IS prior to the de-assertion of the P_RST_N signal. If the link is active, the corresponding LxRXPS signal should be set HIGH ($\geq V_{DDK} - 0.5V$). If a link is unused or disabled, the corresponding LxRXPS signal should be set LOW ($\leq V_{SS} + 0.5V$).
    b. After P_RST_N goes HIGH ($\geq 0.5 \times V_{DDK}$), HMC initialization begins. P_RST_N must meet MIN slew rate of $V_{RSTDVDT} = 0.1V/ns$ as it transitions from LOW to HIGH. The following initialization steps are applicable for active links only.
3. Load HMC Device Global Settings
    a. Global HMC PLL lock occurs within ^{t}INIT. After ^{t}INIT has passed, the I^2C or JTAG bus is used to load the selected configuration registers. When the command is complete, the INIT CONTINUE command must be issued to allow internal configuration to continue. The requester must transmit a scrambled

PRBS stream (may be NULL or non-NULL) at or before the INIT CONTINUE command is issued. While there is no requirement on when the requester begins transmission of NULL FLITs, any delay in doing so after the INIT CONTINUE command is issued will delay the training process.

4. Load Phy Configuration
    a. After the INIT CONTINUE command is issued, the HMC continues with internal initialization, including link PLL lock, receiver equalization, and CDR clock recovery (which depends upon scrambled data supplied by the requester). The responder enters its start state and the transmit signals (LxTXP[n:0] and LxTXN[n:0]) begin to transmit a nonNULL PRBS stream to the requester. This stream is intended to allow the requester to achieve equalization and CDR clock recovery, and also prohibit the requester from acquiring descrambler sync, which is not desired until step 8.
5. Link/Protocol Initialization (Phy and Protocol Training)
    a. To initialize the responder's descramblers, the requester enters the idle state and issues continuous scrambled NULL FLITs to the responder (see NULL Command for NULL description). The responder descrambler sync should occur within ^{t}RESP1 of the PLL locking. Upon descrambler sync, the responder will also enter the IDLE state and begin to transmit scrambled NULL FLITs.
    b. The responder issues continuous scrambled NULL FLITs, and the requester waits for these at its descrambler outputs. The purpose of this step is to achieve synchronization of the requester's descramblers.
    c. Once the requester has synchronized its descramblers, it enters the TS1 state and issues the scrambled TS1 training sequence continuously to the responder to achieve FLIT synchrony (see the TS1 Definition table for TS1 sequence). Responder link lock should occur within ^{t}RESP2.
    d. After responder link lock occurs, it will enter the TS1 state and issue a series of scrambled TS1 training sequence back to the requester. The requester waits for one or more valid TS1 training sequence(s) at the local descrambler output. The requester must then align per-lane receiver timing to achieve host FLIT synchronization.
    e. The requester stops sending TS1 training sequences to the responder after it has achieved link lock, whereby it then starts sending scrambled NULL FLITs. The requester must send a minimum of 32 NULL FLITs before sending the first nonNULL flow packet. Upon detecting the first NULL FLIT, the responder must enter the active state within a period of 32 FLITs, so that it can accept nonNULL flow packets after the requester has transmitted at least 32 NULL FLITs.
    f. Upon detection of the first NULL FLIT after the TS1 sequence, the responder stops sending TS1 sequences and starts sending scrambled NULL FLITs back to the requester. The responder will transmit a minimum of 32 NULL FLITs before sending the first nonNULL flow packet. Upon detecting the first NULL FLIT, the requester must enter the active state within a period of 32 FLITs, so that it can accept nonNULL flow packets after the responder has transmitted at least 32 NULL FLITs. CRC errors on both ends of the link may be disregarded prior to becoming active. After becoming active, IRTRY packets must be held off until after sending 32 NULL FLITs. If the requester is running in response open loop mode, it should not set the RSOPENLOOP bit in the link configuration register until it is ready to accept an unsolicited error response packet that may occur before the first request is transmitted

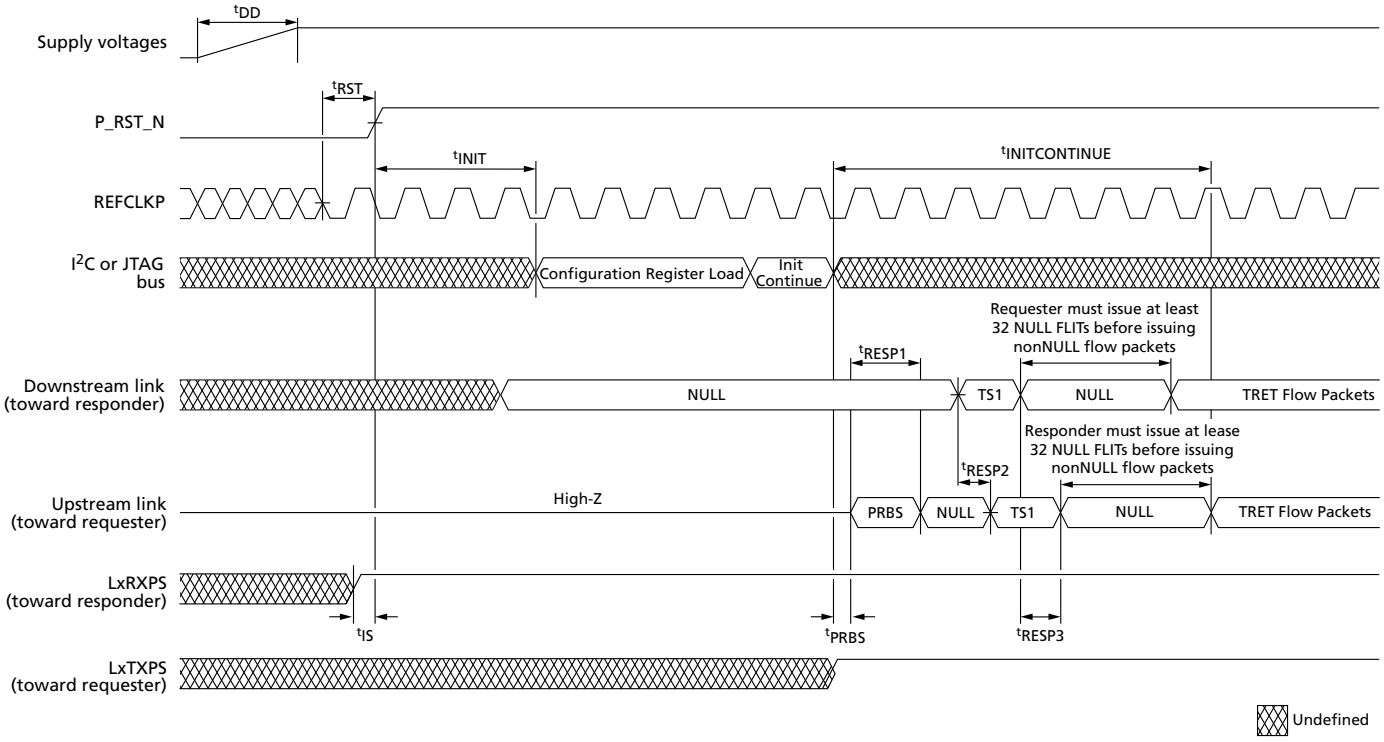
g. The link-layer initialization is complete. Both sides of the link will perform transaction layer initialization. The first TRET sent from the HMC during the transaction layer initialization will have FRP = 1 and RRP = 0. Host commanded power-state transitions are supported at this point. The requester should not transmit TRET packets until it is able to receive transaction layer responses, including error response packets. If the requester is running in response open loop mode, it should not set the RSOPENLOOP bit in the link configuration register until it is ready to accept an unsolicited error response packet that may occur before the first request is transmitted.

Figure 9: HMC Initialization Flowchart



Notes:
1. The requester may begin transmission of NULL FLITs at any point prior to this step.
2. Because configurations allow chaining and two-way communication between host and links, the requester and responders can be host devices or HMCs.

Figure 10: Initialization Timing



Note: 1. Data on links is scrambled.

Power State Management

Each link can independently be set into a lower power state through the usage of the power state management pins, LxRXPS and LxTXPS. Each of the links can be set into a low-power state, sleep mode, as well as a minimum power state called down mode. The LxTXPS control is also used to initiate a link re-initialization sequence.

Power is reduced when sleep mode is entered from normal operation (active mode) through the disablement of the link's high-speed SerDes circuitry. After a link has been initialized and is in active mode, the requester can transition its power state management pin, LxTXPS, from HIGH to LOW to put the opposite side of the link (responder) into sleep mode. As it begins to enter sleep mode, the responder will transition its LxTXPS from HIGH to LOW within the ^{t}PST specified timing. Values for ^{t}PST and all other power state management related timing can be found in the Link Power Management Parameters table. Transition of the responder's LxTXPS will initiate the requester's Rx and Tx lanes to enter into sleep mode as well. It is the responsibility of the host to quiesce traffic (ensure closure on all in-flight transactions) prior to exiting active mode link state. The responder's link logic is not responsible for in-flight transactions when LxRXPS goes from 1 to 0.

Down mode allows a link to go to an even lower power state than sleep mode by disabling both the high-speed SerDes circuitry as well as the link's PLLs. A link enters down mode if its corresponding LxRXPS signal is LOW when the P_RST_N signal transitions from LOW to HIGH either during initialization or a reset event. The HMC can be config-

ured so that all of its links will enter down mode after the last link in active mode transitions to sleep mode. In this setting, any link already in sleep mode when the last link exits active mode will be transitioned to down mode to further reduce power consumption. Transitioning any links from sleep mode to down mode will take up to 150µs (represented by ^{t}SD specification).

When all links exit active mode and transition to down mode, the HMC enters a self refresh state. Although not accessible from the links, data stored within the memory is still maintained. Upon entering self refresh the HMC must stay in this state for a minimum of 1ms (^{t}SREF). This is measured from the time the HMC's last link transitions its LxRXPS LOW (entering self refresh) to its first LxRXPS to transition HIGH (entering active mode).

All links will independently respond to power state control; however, in the event of simultaneous power state transitions, the HMC will stagger transitions between active mode and sleep modes in order to avoid supply voltage shifts. The amount of stagger time incurred is defined by the ^{t}SS specification. Multiply ^{t}SS by n - 1 to determine when the final link will transition states, where n equals the number of simultaneous link transitions. For example, if the requester sets its L0TXPS, L1TXPS, L2TXPS, and L3TXPS to 0 at the same time, it will take 1.5µs for the HMC to transition all four links into sleep mode (^{t}SS × 3). This is also true for links exiting sleep mode and entering active mode. As an example, if L0RXPS and L1RXPS transition from LOW to HIGH at the same time, the second links will not start the transition to active mode until up to 500ns later (^{t}SS × 1).

The transition of a link to active mode from either sleep mode or down mode requires a re-initialization sequence as illustrated in Figure 11 (page 29).

Bringing a link from down mode into active mode requires additional time to complete the SerDes PLL self-calibration as specified by ^{t}PSC (see Note 3 of Figure 11 (page 29)).

The transition of a pass-through link's LxTXPS signal is dependent on the power state transition of the host links within the same HMC:

- A pass-through link's LxTXPS will transition LOW after the last host link in the same cube enter sleep mode.
- A pass-through link's LxTXPS will transition HIGH as soon as the LxTXPS of any host link in the same cube transitions HIGH.

Table 8: Link Power States and Conditions

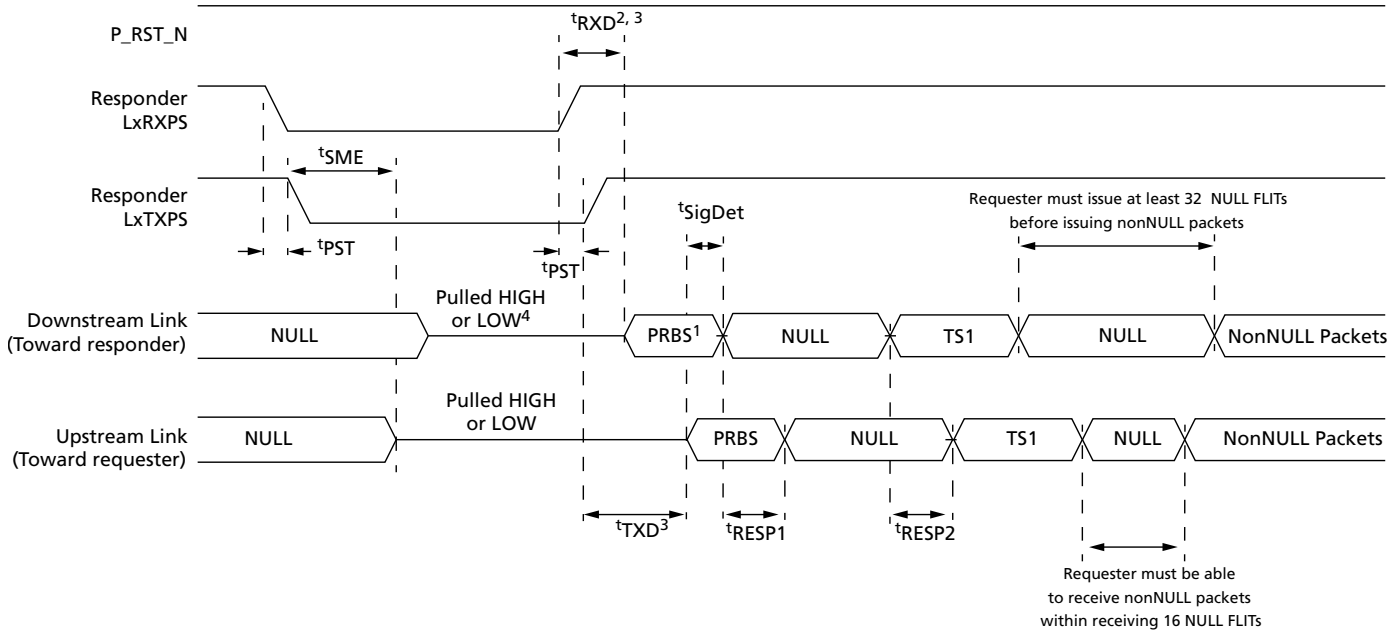
Mode	Method of Entry	HMC Tx Lane Status	HMC Rx Lane Status	Link PLL Status
Active	Standard initialization	Enabled	Enabled	Enabled
Link retraining	LxRXPS asserted LOW while in active mode Held LOW for a minimum of <TBD> and released back to HIGH prior to 1ms	Pulled HIGH or LOW	Disabled	Enabled
Sleep	LxRXPS asserted LOW while in active mode Held LOW for a minimum of 1ms (^{t}SREF[3])	Pulled HIGH or LOW	Disabled	Enabled

Table 8: Link Power States and Conditions (Continued)

Mode	Method of Entry	HMC Tx Lane Status	HMC Rx Lane Status	Link PLL Status
Down	1) LxRXPS asserted LOW when P_RST_N transitions from LOW to HIGH; or 2) After the last link in active mode enters sleep mode (all other links already in sleep or down mode)	Pulled HIGH or LOW	Disabled	Disabled

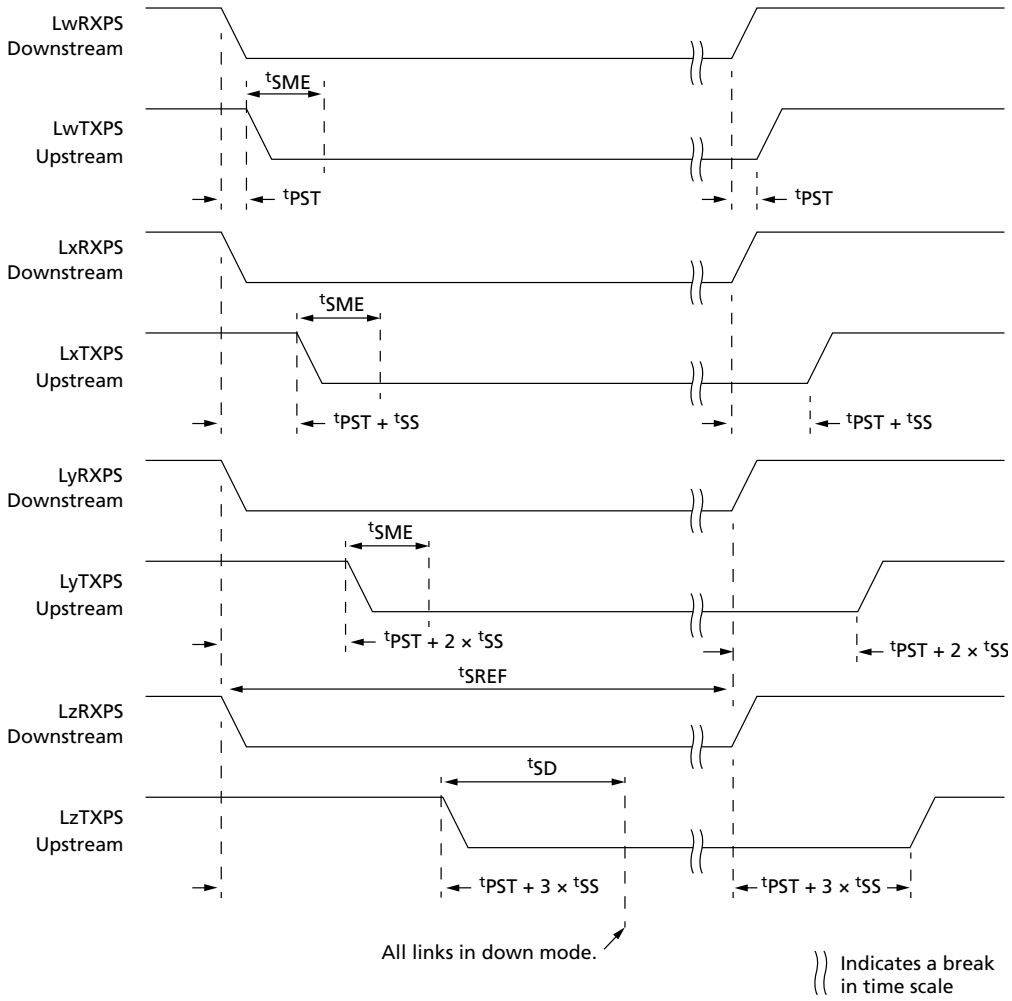
Notes:
1. Data from all vaults available through any link in active mode, even if other links are in sleep or down mode.
2. Self refresh is entered when all links exit active mode whereby data in HMC memory is retained but unavailable.
3. ^{t}SREFL is the time it takes for link initialization and is included in ^{t}SREF, which puts the device into sleep mode or warm reset. Both parameters are listed in the Link Power Management Parameters table. ^{t}SREF is shown in Figure 12 (page 30).

Figure 11: Sleep Mode Entry and Exit (Single Link Only)



Notes:
1. PRBS transmitted by HMC pass-through link; NULLs or PRBS may be transmitted from host.
2. ^{t}RXD is only applicable in a pass-through link (the case of a cube-to-cube link).
3. ^{t}PSC must be added to this delay when exiting down mode.
4. The host may issue NULL FLITs instead of pulled HIGH or LOW.

Figure 12: Simultaneous Transition of Four Host Links to Sleep Mode, Entry into Down Mode and Return to Active Mode (Single HMC, Four Link Example)



Link Layer

The link layer handles communication across the link not associated with the transaction layer. This consists of the transmission of NULL FLITs and flow packets.

All FLITs of a packet must be transmitted sequentially, without interruption, across the link. NULL FLITs are generated at the link layer when no other packets are being transmitted. A NULL FLIT is an all-zeros pattern that (in common with all FLITs) is scrambled prior to transmission. Any number of packets can be streamed back-to-back across the link, or they can be separated by NULL FLITs, depending upon system traffic and transaction layer flow control. There is no minimum or maximum requirement associated with the number of NULL FLITs transmitted between packets. NULL FLITs are not subject to flow control. The first nonzero FLIT following a NULL FLIT is considered to be necessarily the first FLIT of a packet. Data FLITs within a packet may be all zeros, but these lie between a header FLIT and a tail FLIT and therefore cannot be misinterpreted as NULL FLITs.

Flow packets are generated by the link master to pass flow control and retry control commands to the opposite side of the link. Flow packets are sent when no other link traffic is occurring or when a link retry sequence is to be initiated. Flow packets are single-FLIT packets with no data payload and are not subject to flow control. Flow packets utilize the same header and tail format as request packets, but are not considered requests, and do not have corresponding response commands. See Flow Commands for specifics on the commands transmitted with flow packets.

Transaction Layer

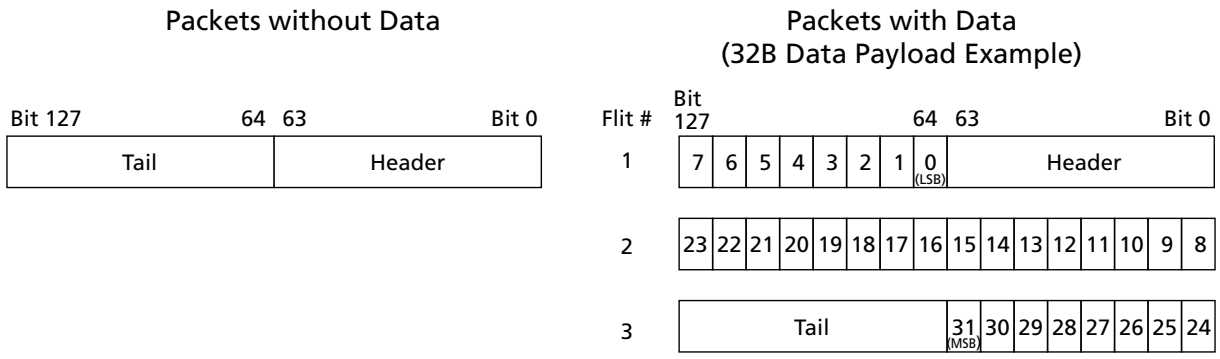
As noted previously, all information transmitted across the links is included in packets that consist of one or more FLITs. A packet always includes an 8-byte header at the beginning of the packet and an 8-byte tail at the end of the packet. The header includes the command field that identifies the packet type, other control fields, and when required, the addressing information. The tail includes flow and link-retry control fields along with the CRC field. There are three different packet types. The first two, request and response packets, are used at the transaction layer. The third type, the flow packet, is only used at the link layer, however it is included below so that all packet types can be described within a single section.

1. Request packets are issued by the requester (host or HMC configured as a pass-thru link). The request packet header includes address information to perform a READ or WRITE operation. Write request packets include data FLITs.
2. Response packets are issued by the responder (HMC configured as a host link). Response headers do not include address information. READ responses include data FLITs. Write response packets do not include data FLITs.
3. Flow packets are not part of the transaction layer protocol and are not subject to flow control. They are only used at the link to pass flow control and retry control information back to the link master when normal transaction layer packets are not flowing in the return direction. Flow packets are generated at the link master and are decoded and then dropped at the link slave; they are not forwarded and do not pass through the HMC.

Each packet type has its own defined header and tail formats, described in the Request Packets, Response Packets, and Flow Packets sections.

Packets that consist of multiple FLITs are transmitted across the link with the least significant FLIT number first. The first FLIT includes the header and the least significant bits (LSBs) of the data. Subsequent data FLITs progress with more significant data following. The figure below illustrates the layout for packets with and without data.

Figure 13: Packet Layouts



Note: 1. Each numbered data field represents a byte with bit positions [7(MSB): 0(LSB)].

A CRC field is included in the tail of every packet that covers the entire packet, including the header, all data, and the nonCRC tail bits. The link retry logic retransmits packets across the link if link errors are detected, as described in the Link Retry section. Upon successful transmission of packets across the link, the packets are transmitted through the logic base all the way to the destination vault controller. CRC provides command and data through-checking to the destination vault controller. ECC provides error coverage of the data from the vault controller to and from the DRAM arrays. Thus, data is protected along the entire path to and from the memory device. If the vault controller detects a correctable error, error correction is executed before the data is returned. If an uncorrectable error is detected, an error status is returned in the response packet back to the requester.

The CRC algorithm used on the HMC is the Koopman CRC-32K. This algorithm was chosen for the HMC because of its balance of coverage and ease of implementation. The polynomial for this algorithm is:

x32 + x30 + x29 + x28 + x26 + x20 + x19 + x17 + x16 + x15 + x11 + x10 + x7 + x6 + x4 + x2 + x + 1

The CRC calculation operates on the LSB of the packet first. The packet CRC calculation must insert 0s in place of the 32-bits representing the CRC field before generating or checking the CRC. For example, when generating CRC for a packet, bits [63: 32] of the Tail presented to the CRC generator should be all zeros. The output of the CRC generator will have a 32-bit CRC value that will then be inserted in bits [63:32] of the Tail before forwarding that FLIT of the packet. When checking CRC for a packet, the CRC field should be removed from bits [63:32] of the Tail and replaced with 32-bits of zeros, then presented to the CRC checker. The output of the CRC checker will have a 32-bit CRC value that can be compared with the CRC value that was removed from the tail. If the two compare, the CRC check indicates no bit failures within the packet.

The two tables that follow represent a single FLIT packet and multi-FLIT packet, respectively. The purpose of these examples is to illustrate how CRC is generated.

Table 9: Single FLIT Example: TRET Packet

UI	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Hex Value
0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0x0882
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0x0100
5	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0xf801
6	1	0	1	0	0	1	1	1	1	0	0	0	1	0	1	0	0xa78a
7	0	1	0	1	1	1	1	0	1	0	0	1	0	0	0	1	0x5e91

Table 10: Mulit-FLIT Example: 2ADD8 Request Packet

UI	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Hex Value
0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0x1112
1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0x801D
2	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1	1	0x282B
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
4	1	1	1	0	0	1	1	0	0	1	1	0	0	1	0	0	0xE664
5	0	0	1	1	0	1	0	0	1	0	1	0	1	1	1	0	0x34AE
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
8	1	0	1	1	1	1	1	1	1	0	0	1	0	0	0	1	0xBF91
9	0	0	1	1	0	0	1	1	1	0	0	0	1	1	0	0	0x338C
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
12	0	1	0	0	1	0	1	1	1	1	0	0	0	0	1	0	0x4BC2
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0x0002
14	1	1	1	1	1	0	0	0	1	1	0	1	0	1	0	0	0xF8D4
15	0	1	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0x6418

Overall packet lengths are from 1 to 17 FLITs. Write request packets and read response packets will contain from 16B to 256B of data between the header and the tail.

All requests are operationally closed as follows:

- Request packets are uniquely identified with a tag field embedded in the packet.
- All read request packets have a corresponding read response packet returned to the requesting link that includes the corresponding request tag and requested data.
- Normal write request packets are acknowledged with an explicit write response packet that includes the write request's tag.
  - This positive acknowledgment provides an indication to the requesting processor that the request has been executed without error.

- As the HMC does not use the TAG field of posted write requests, the host can populate this field with any value.
- If unrecoverable errors occur in the request path, they are captured in the error status that is included in the response packet.
- If unrecoverable errors occur in the request path for posted write requests, an error response packet is returned to the requester indicating an error.

Flow packets are not considered a request, they do not have a valid tag, and they do not have a corresponding response packet.

The flow of request packets and response packets are managed with tokens, described in the Packet Flow Control section. Flow packets have no flow control and can be sent at any time by the link master.

Memory Addressing

A request packet header includes an address field of 34 bits for internal memory addressing within the HMC. This includes vault, bank, and DRAM address bits. The configurations currently defined will provide a total of up to 8GB of addressable memory locations within one HMC.

The user can select a specific address mapping scheme so that access of the HMC vaults and banks is optimized for the characteristics of the request address stream. If the request address stream is either random or generally sequential from the low-order address bits, the host can choose to use one of the default address map modes offered in the HMC (See the Default Address Map Mode Table). The default address mapping is based on the maximum block size chosen in the address map mode register. It maps the vault address toward the less significant address bits, followed by the bank address just above the vault address. This address mapping algorithm is referred to as “low interleave,” and forces sequential addressing to be spread across different vaults and then across different banks within a vault, thus avoiding bank conflicts. A request stream that has a truly random address pattern is not sensitive to the specific method of address mapping.

Memory Addressing Granularity

HMC supports READ and WRITE data block accesses with 16-byte starting-address granularity up to the maximum block size setting (32B, 64B, 128B, or 256B). Data length granularity of 16-bytes is supported up to 128-bytes. The maximum block size is set with the Address Configuration register. See Data Access Performance Considerations for information on how to select maximum block size such that it optimizes performance in a given system. The maximum block size specified in the address map mode register determines the number of bits within the byte address field, of which the four LSBs are ignored due to the 16-byte granularity (an exception is the BIT WRITE command). The 4 LSBs of the byte address may be used for future density expansion if 16B block aligned access is preserved. Details for address expansion are outside the scope of this specification version. The remaining upper bits of the byte address indicate the starting 16-byte boundary when accessing a portion or all of the block. The vault controller uses these upper bits of the byte address as part of its DRAM column addressing. If the starting 16- byte address in conjunction with the data access length of the request causes the access to go past the end of the maximum block size boundary, the DRAM column addressing will wrap within the maximum block size and continue. For example, if the maximum block size is 64 bytes and a 48-byte READ command is received

starting at byte 32, the DRAM access will occur starting at byte 32 up to byte 63, then continue at byte 0 up to byte 15.

Memory Address-to-Link Mapping

Each link is associated with eight local vaults that are referred to as a quadrant. Access from a link to a local quadrant may have lower latency than to a link outside the quadrant. Bits within the vault address designate both the specific quadrant and the vaults within that quadrant.

Table 11: Addressing Definitions

Address	Description	Comments
Byte address	Bytes within the maximum supported block size	The four LSBs of the byte address are ignored for READ and WRITE requests (with the exception of BIT WRITE command; See BIT WRITE Command for details)
Vault address	Addresses vaults within the HMC	Lower three bits of the vault address specifies 1 of 8 vaults within the logic chip quadrant
		Upper two bits of the vault address specifies 1 of 4 quadrants
Bank address	Addresses banks within a vault	4GB HMC: Addresses 1 of 8 banks in the vault
		8GB HMC: Addresses 1 of 16 banks in the vault
DRAM address	Addresses DRAM rows and column within a bank	The vault controller breaks the DRAM address into row and column addresses, addressing 1Mb blocks of 16 bytes each

Default Address Map Mode Tables

Table 12: Default Address Map Mode Table – 4GB

Revision 2.x protocol

Request Address Bit	4GB			
	32-Byte Max Block Size	64-Byte Max Block Size	128-Byte Max Block Size	256-Byte Max Block Size
33	Ignored	Ignored	Ignored	Ignored
32	Ignored	Ignored	Ignored	Ignored
31	DRAM[19]	DRAM[19]	DRAM[19]	DRAM[19]
30	DRAM[18]	DRAM[18]	DRAM[18]	DRAM[18]
29	DRAM[17]	DRAM[17]	DRAM[17]	DRAM[17]
28	DRAM[16]	DRAM[16]	DRAM[16]	DRAM[16]
27	DRAM[15]	DRAM[15]	DRAM[15]	DRAM[15]
26	DRAM[14]	DRAM[14]	DRAM[14]	DRAM[14]
25	DRAM[13]	DRAM[13]	DRAM[13]	DRAM[13]
24	DRAM[12]	DRAM[12]	DRAM[12]	DRAM[12]
23	DRAM[11]	DRAM[11]	DRAM[11]	DRAM[11]
22	DRAM[10]	DRAM[10]	DRAM[10]	DRAM[10]
21	DRAM[9]	DRAM[9]	DRAM[9]	DRAM[9]
20	DRAM[8]	DRAM[8]	DRAM[8]	DRAM[8]
19	DRAM[7]	DRAM[7]	DRAM[7]	DRAM[7]
18	DRAM[6]	DRAM[6]	DRAM[6]	DRAM[6]
17	DRAM[5]	DRAM[5]	DRAM[5]	DRAM[5]
16	DRAM[4]	DRAM[4]	DRAM[4]	DRAM[4]
15	DRAM[3]	DRAM[3]	DRAM[3]	Bank[2]
14	DRAM[2]	DRAM[2]	Bank[2]	Bank[1]
13	DRAM[1]	Bank[2]	Bank[1]	Bank[0]
12	Bank[2]	Bank[1]	Bank[0]	Vault[4]
11	Bank[1]	Bank[0]	Vault[4]	Vault[3]
10	Bank[0]	Vault[4]	Vault[3]	Vault[2]
9	Vault[4]	Vault[3]	Vault[2]	Vault[1]
8	Vault[3]	Vault[2]	Vault[1]	Vault[0]
7	Vault[2]	Vault[1]	Vault[0]	Byte[7] = DRAM[3]
6	Vault[1]	Vault[0]	Byte[6] = DRAM[2]	Byte[6] = DRAM[2]
5	Vault[0]	Byte[5] = DRAM[1]	Byte[5] = DRAM[1]	Byte[5] = DRAM[1]
4	Byte[4] = DRAM[0]	Byte[4] = DRAM[0]	Byte[4] = DRAM[0]	Byte[4] = DRAM[0]
3	Byte[3] = ignored	Byte[3] = ignored	Byte[3] = ignored	Byte[3] = ignored
2	Byte[2] = ignored	Byte[2] = ignored	Byte[2] = ignored	Byte[2] = ignored
1	Byte[1] = ignored	Byte[1] = ignored	Byte[1] = ignored	Byte[1] = ignored
0	Byte[0] = ignored	Byte[0] = ignored	Byte[0] = ignored	Byte[0] = ignored

Table 13: Default Address Map Mode Table – 8GB

Revision 2.x protocol

Request Address Bit	8GB			
	32-Byte Max Block Size	64-Byte Max Block Size	128-Byte Max Block Size	256-Byte Max Block Size
33	Ignored	Ignored	Ignored	Ignored
32	DRAM[19]	DRAM[19]	DRAM[19]	DRAM[19]
31	DRAM[18]	DRAM[18]	DRAM[18]	DRAM[18]
30	DRAM[17]	DRAM[17]	DRAM[17]	DRAM[17]
29	DRAM[16]	DRAM[16]	DRAM[16]	DRAM[16]
28	DRAM[15]	DRAM[15]	DRAM[15]	DRAM[15]
27	DRAM[14]	DRAM[14]	DRAM[14]	DRAM[14]
26	DRAM[13]	DRAM[13]	DRAM[13]	DRAM[13]
25	DRAM[12]	DRAM[12]	DRAM[12]	DRAM[12]
24	DRAM[11]	DRAM[11]	DRAM[11]	DRAM[11]
23	DRAM[10]	DRAM[10]	DRAM[10]	DRAM[10]
22	DRAM[9]	DRAM[9]	DRAM[9]	DRAM[9]
21	DRAM[8]	DRAM[8]	DRAM[8]	DRAM[8]
20	DRAM[7]	DRAM[7]	DRAM[7]	DRAM[7]
19	DRAM[6]	DRAM[6]	DRAM[6]	DRAM[6]
18	DRAM[5]	DRAM[5]	DRAM[5]	DRAM[5]
17	DRAM[4]	DRAM[4]	DRAM[4]	DRAM[4]
16	DRAM[3]	DRAM[3]	DRAM[3]	Bank[3]
15	DRAM[2]	DRAM[2]	Bank[3]	Bank[2]
14	DRAM[1]	Bank[3]	Bank[2]	Bank[1]
13	Bank[3]	Bank[2]	Bank[1]	Bank[0]
12	Bank[2]	Bank[1]	Bank[0]	Vault[4]
11	Bank[1]	Bank[0]	Vault[4]	Vault[3]
10	Bank[0]	Vault[4]	Vault[3]	Vault[2]
9	Vault[4]	Vault[3]	Vault[2]	Vault[1]
8	Vault[3]	Vault[2]	Vault[1]	Vault[0]
7	Vault[2]	Vault[1]	Vault[0]	Byte[7] = DRAM[3]
6	Vault[1]	Vault[0]	Byte[6] = DRAM[2]	Byte[6] = DRAM[2]
5	Vault[0]	Byte[5] = DRAM[1]	Byte[5] = DRAM[1]	Byte[5] = DRAM[1]
4	Byte[4] = DRAM[0]	Byte[4] = DRAM[0]	Byte[4] = DRAM[0]	Byte[4] = DRAM[0]
3	Byte[3] = ignored	Byte[3] = ignored	Byte[3] = ignored	Byte[3] = ignored
2	Byte[2] = ignored	Byte[2] = ignored	Byte[2] = ignored	Byte[2] = ignored
1	Byte[1] = ignored	Byte[1] = ignored	Byte[1] = ignored	Byte[1] = ignored
0	Byte[0] = ignored	Byte[0] = ignored	Byte[0] = ignored	Byte[0] = ignored

Address Mapping Mode Register

The address mapping mode register provides user control of mapping portions of the ADRS field within the header of request packets to the vault and bank addresses. When using default address-mapping, the address mapping mode field should be set based on specific system requirements. See Data Access Performance Considerations for details.

In addition to the default address mapping algorithms, user-defined fields are provided for the user to specify a unique mapping algorithm. There is a user-defined field for the vault address and the bank address. Each field holds a 5-bit encoded value pointing to the corresponding bit position in the ADRS field of the request packet header. This represents the LSB that will be used for the vault or bank address, with the more significant bits of the subaddress coming from the bit positions immediately above it in the request header ADRS field. When operating with the user-defined modes, it is the user's responsibility to set the user-defined encoded fields so that the vault and bank addresses do not overlap each other or overlap the byte address within the request header ADRS field. The byte address is always fixed at the LSBs of the request header ADRS field. The user must also be aware that the number of bits in the bank address is dependent upon the size of the HMC. The remaining bits of the request ADRS field not specified as the vault and bank addresses become the DRAM address.

The address map mode register is writable via the MODE WRITE command or via the maintenance interface (I^2C or JTAG bus). Because the MAX block size is defaulted to 64 bytes, the host initialization code must either be compatible (issue requests of 64 bytes or smaller) or issue a MODE WRITE command to change the MAX block size.

DRAM Addressing

Each bank in the HMC contains 16,777,216 memory bytes (16MB). A WRITE or READ command to a bank accesses 32 bytes of data for each column fetch. The bank configuration is identical in all specified HMC configurations.

Packet Length

As noted previously, packets are always multiples of 16-byte FLITs. Within each packet header there is a length (LNG) field that indicates the total number of FLITs in the packet. A packet that contains no data FLITs would have LNG = 1; this represents the single FLIT holding the 8-byte header and 8-byte tail. When generating response packets, the necessary data size is determined from the CMD field in the corresponding request packet. Specific packet lengths are provided in the following tables: Transaction Layer – Request Commands, Transaction Layer – Response Commands, and Flow Commands.

Packet Flow Control

Flow control occurs in both directions on each link. The flow of transaction layer packets is managed with token counts. Each token represents the storage to hold one FLIT (16 bytes). The link slave input buffer temporarily stores transaction layer packets as they are received from the link. (Flow packets are not stored in the input buffer, and therefore, are not subject to flow control.) The minimum size of this buffer must be equal to or greater than the number of FLITs in a single packet of the largest supported length, but in general can hold many FLITS to enable a constant flow across the link. The available space in this input buffer is represented in a token count register at the link master. This gives the link master knowledge of the available buffering at the other

end of the link and the ability to evaluate whether there is enough buffer space to receive every FLIT of the next packet to be transmitted.

The token count register is loaded during the initialization sequence with the maximum number of tokens representing the available buffer space at the link slave when it is empty. As the link master sends each transaction layer packet across the link, it must decrement the token count register by the number of FLITs in the transmitted packet. As the packet is received and stored in the input buffer, its FLITs use up the space represented by the decremented value of token count register at the link master. As each packet is read out of the input buffer, it is forwarded to a destination, and its FLIT location is freed up. This requires a mechanism to return packet tokens to the link transmitter, using the opposite link direction. The input buffer control logic sends a count (representing the number of FLITs that were read out of the input buffer when the packet was forwarded) to the local (adjacent) link master. This count is returned in the return token count (RTC) field of the next possible packet traveling in the opposite direction on the link. At the link slave, the RTC field is extracted from the incoming packet and sent back to the original link transmitter where its value is added to the current value of the token count register. If no transaction layer packet traffic is occurring in the return direction, the link master must create a flow packet known as a token return (TRET) to return the token. Not returning tokens can lead to performance degradation or stalling. TRET is described in TOKEN RETURN (TRET) Command.

Appropriate sizing of the link slave input buffer requires weighing the trade-offs between latency, throughput, silicon real estate, and routability. If there is a temporary pause in the packet forwarding priority at the output of the input buffer (internal switch conflicts, or destination vault flow control), a packet might not be emptied from the input buffer. This would cause a pause in the return of the tokens. As long as this is infrequent and the input buffer is sized to accommodate it, the packet flow will not be disrupted unless the link is running at 100% busy. If the input buffer is empty, a specific implementation may choose to bypass the input buffer and forward a packet immediately to its destination. In this case, the tokens for the packet would be immediately returned to the link master.

As noted previously, the link master must not transmit a packet if there is insufficient space in the input buffer at the other end of the link. The LNG field within the header of each outgoing packet must be compared to the token count register to determine whether the packet should be transmitted or the packet stream should be paused.

Tagging

Operational closure for requests is accomplished using the tag fields in request and response packets. The tag value remains associated with the request until a response is received indicating that the request has been executed. Tags in READ requests are returned with the respective read data in the read response packet header. Write response tags are returned within a write response packet. In the case of posted write requests, because no response is returned and the HMC does not use the TAG field of posted write requests, the host can populate this field with any value. When multiple host links are connected to the HMC, each response packet will be returned on the same link as its associated request. This keeps the tag range independent for each host link.

Tag fields in packets are eleven bits long, which is enough space for 2048 tags. All tags are available for use. Tag assignment and reassignment are managed by the host. There is no required algorithm to assign tag values to requests. HMC does not use the tag for

internal control or identification, only for copying the tag from the request packet to the response packet.

Tags are assigned by control logic at the host link master and must not be used in another request packet until a response tag with the same tag number is returned to that host link. Other host links will use the same tag range of 2048 tags, but they are uniquely identified by their association with each different host link. Additionally, in a multi-HMC topology, the host could choose to expand the number of usable tags by associating each tag set (based on host link) with the target cube of the request. This is enabled by the inclusion of the cube number in each response packet. Thus, the total maximum number of unique tags is 2048 times the number of host links of one HMC, times the number of cubes in the chain. For example, if four links of one HMC are connected from an HMC to the host, there are 8192 usable tags for requests to the HMC. As an implementation example, a host control logic might decide to provide a time-out capability for every transaction to determine whether a request or response packet has been lost or corrupted, preventing the tag from being returned. The timer in this implementation example would consider link retry periods that occur to account for response packets that are delayed, but still returned. Host timeout values must be determined from performance modeling of the HMC using specific host request streams, with its address patterns and transaction mix, to determine expected maximum latency of transactions. The transaction timeout period must be guard-banded to allow for possible significant variability in latency doe to host request stream variability.

Packet Integrity

The integrity of the packet contents is maintained with the CRC field in the tail of every packet. Because the entire packet (including header and tail) is transmitted from the source link all of the way to the destination vault, CRC is used to detect failures that occur not only on transmission across the link, but along the entire path. CRC may be regenerated along the path if flow control fields within the header or tail change. CRC regeneration is done in an overlapped fashion with respect to a CRC check to ensure that no single point of failure will go undetected.

There may be cases when the first FLITs of a packet are forwarded before the tail is received and the CRC is checked. This occurs in the HMC's link slave after a request packet crosses a link and is done to avoid adding latency in the packet path. If the packet is found to have a CRC error as it passes through the link slave, the packet is "poisoned," meaning that a destination, in this case a vault controller, will recognize it as nonfunctional and will not use it. The link slave poisons the packet by inverting a recalculated value of the CRC and inserting that into the tail in place of the errored CRC. A successful link retry will result in the original packet being resent, thus replacing the poisoned packet. Another example of a forwarded poisoned packet is the case in which DRAM errors occur and are internally retried. If a parity error occurs on the command or address from the vault controller to the DRAM, a response packet may be generated and transmission back to the requester could be started before the parity error is detected. In this case, the CRC in the tail of the response packet will be poisoned, meaning the vault controller will invert the CRC and insert it into the tail of the packet. Consequently, the requester will receive the poisoned packet and must drop it. The vault controller will retry the DRAM request, and upon a successful DRAM access, another response packet will be generated to replace the poisoned one.

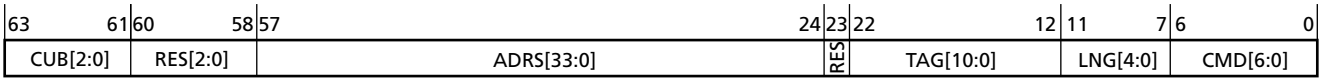
If a packet travels across a link after it is poisoned, a link master will still be able to embed flow control fields in the packet by recalculating the CRC with the embedded val-

ues, then inverting the recalculated CRC so that the poisoned state will be maintained. Because the flow control fields are valid in a poisoned packet, it is stored in the retry buffer. In this case, the link slave on the other end of the link will recognize the poisoned CRC and will still extract the flow control fields. Whenever a packet is poisoned, the CMD, ADRS, TAG, and ERRSTAT fields are not valid, but the flow control fields (FRP, RRP, RTC) are valid.

Request Packets

Request packets carry request commands from the requester (host or HMC link configured as pass-thru link) to the responder (HMC link configured as host link). All request packets are subject to normal packet flow control.

Figure 14: Request Packet Header Layout



Request packet headers for request commands and flow commands contain the fields shown in the table below. Specific commands may require some of the fields to be 0. These are provided in the Valid Field and Command Summary Table.

Table 14: Request Packet Header Fields

Name	Field Label	Bit Count	Bit Range	Function
Cube ID	CUB	3	[63:61]	CUB field used to match request with target cube. The internal Cube ID Register defaults to the value read on external CUB pins of each HMC device.
Reserved	RES	3	[60:58]	Reserved: These bits are reserved for future address or Cube ID expansion. The responder will ignore bits in this field from the requester except for including them in the CRC calculation. The HMC can use portions of this field range internally.
Address	ADRS	34	[57:24]	Request address. For some commands, control fields are included within this range.
Reserved	RES	1	[23]	
Tag	TAG	11	[22:12]	Tag number uniquely identifying this request.
Packet length	LNG	5	[11:7]	Length of packet in FLITs (1 FLIT is 128 bits). Includes header, any data payload, and tail.
Command	CMD	7	[6:0]	Packet command. (See the Transaction Layer – Request Commands table for the list of request commands.)

Figure 15: Request Packet Tail Layout

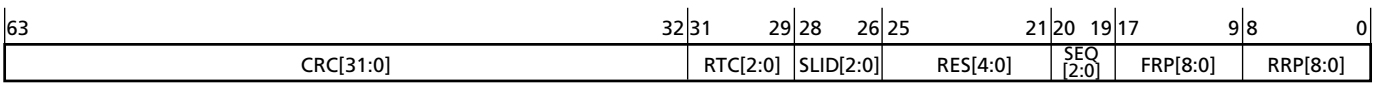


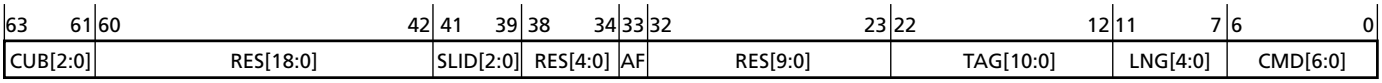
Table 15: Request Packet Tail Fields

Name	Field Label	Bit Count	Bit Range	Function
Cyclic redundancy check	CRC	32	[63:32]	The error-detecting code field that covers the entire packet.
Return token count	RTC	3	[31:29]	Return token count for transaction-layer flow control. In the request packet tail, the RTC contains the encoded value for tokens that represent available space in the requester's input buffer.
Source Link ID	SLID	3	[28:26]	Used to identify the source link for response routing. The incoming value of this field is ignored by HMC. Internally, HMC overwrites this field and uses the value for response routing; refer to the description of the SLID field in the response header.
Reserved	RES	5	[25:21]	Reserved: The responder will ignore bits in this field from the requester except for including them in the CRC calculation. The HMC can use portions of this field range internally.
Sequence number	SEQ	3	[20:18]	Incrementing value for each packet transmitted, except for PRET and IRTRY packets (see Packet Sequence Number Generation).
Forward retry pointer	FRP	9	[17:9]	Retry pointer representing this packet's position in the retry buffer (see Retry Pointer Description).
Return retry pointer	RRP	9	[8:0]	Retry pointer being returned for other side of link (see Retry Pointer Description).

Response Packets

Response packets carry response commands from the responder (HMC link configured as host link) to the requester (host or HMC link configured as pass-thru link). All response packets are subject to normal packet flow control.

Figure 16: Response Packet Header Layout



Response packet headers for response commands contain the fields shown in the table below. Specific commands may require some of the fields to be 0. These are provided in the Valid Field and Command Summary Table.

Table 16: Response Packet Header Fields

Name	Field Label	Bit Count	Bit Range	Function
CUB ID	CUB	3	[63:61]	The target cube inserts its Cube ID number into this field. The requester can use this field for verification and for identifying unique tags per the target cube.

Table 16: Response Packet Header Fields (Continued)

Name	Field Label	Bit Count	Bit Range	Function
Reserved	RES	19	[60:42]	Reserved: The host will ignore bits in this field from the HMC except for including them in the CRC calculation.
Source Link ID	SLID	3	[41:39]	Used to identify the source link for response routing. This value is copied from the corresponding Request header and used for response routing purposes. The host can ignore these bits (except for including them in the CRC calculation).
Reserved	RES	5	[38:34]	Reserved: The host will ignore bits in this field from the HMC except for including them in the CRC calculation.
Atomic flag	AF	1	[33]	Atomic flag
Reserved	RES	10	[32:23]	Reserved: The host will ignore bits in this field from the HMC except for including them in the CRC calculation.
Tag	TAG	11	[22:12]	Tag number uniquely associating this response to a request.
Packet length	LNG	5	[11:7]	Length of packet in 128-bit FLITs
Command	CMD	7	[6:0]	Packet command (see the Transaction Layer – Response Commands table for response commands)

Figure 17: Response Packet Tail Layout

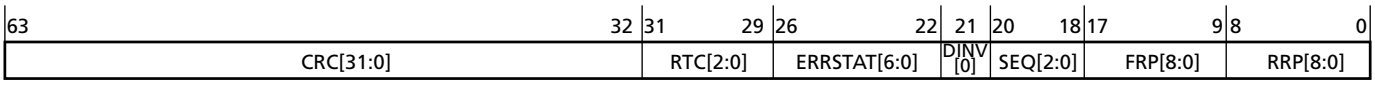


Table 17: Response Packet Tail Fields

Name	Field Label	Bit Count	Bit Range	Function
Cyclic redundancy check	CRC	32	[63:32]	Error-detecting code field that covers the entire packet.
Return token counts	RTC	3	[31:29]	Return token count for transaction-layer flow control. In the response packet tail, the RTC contains an encoded value equaling the returned tokens. The tokens represent incremental available space in the HMC input buffer. See the table below for the RTC encoding key.
Error status	ERRSTAT	7	[28:22]	Error status bits (see the ERRSTAT[6:0] bit definitions tables)
Data Invalid	DINV	1	[21]	Indicates validity of packet payload. Data in packet is valid if DINV = 0 and invalid if DINV = 1.
Sequence number	SEQ	3	[20:18]	Incrementing value for each packet transmitted. See Packet Sequence Number Generation).
Forward retry pointer	FRP	9	[17:9]	Retry pointer representing this packet's position in the retry buffer (see Retry Pointer Description).

Table 17: Response Packet Tail Fields (Continued)

Name	Field Label	Bit Count	Bit Range	Function
Return retry pointer	RRP	9	[8:0]	Retry pointer being returned for the other side of link (see Retry Pointer Description).

Table 18: RTC Encoding

Token Count Returned	Token Bit Encoding		
	RTC[2:0]		
0	0	0	0
1	0	0	1
2	0	1	0
4	0	1	1
8	1	0	0
16	1	0	1
32	1	1	0
64	1	1	1

The ERRSTAT field included in the response tail is valid for all read, write, and error responses. All bits in the ERRSTAT field being zero indicates a normal response with no errors or flags. There are 6 different error and flag categories defined by ERRSTAT[6:4] bits. The unique error or flag descriptions within each category indicate a specific error or item that requires attention. Additional status on errors or flags can be obtained by accessing internal status registers using the MODE READ command or sideband (I^2C or JTAG). The table that follows shows the error categories and descriptions along with the method of response (read/write response packet or error response packet).

Table 19: ERRSTAT[6:0] Bit Definitions – No Errors

ERRSTAT Bit							Description	Response Packet Type
6	5	4	3	2	1	0		
0	0	0	0	0	0	0	No errors/conditions	

Table 20: ERRSTAT[6:0] Bit Definitions – Warnings

ERRSTAT Bit							Description	Response Packet Type
6	5	4	3	2	1	0		
0	0	0	0	0	0	1	Operational temperature threshold: The internal temperature sensors in the HMC have reached the upper limits of operational temperature.	Error response packet (TAG = CUB number)
0	0	0	0	0	1	0	Host token overflow warning: The host sent more than 1023 tokens to the HMC on a link.	
0	0	0	0	1	0	1	Repair warning: There are one or more regions of memory that have no available repair resources left.	
0	0	0	0	1	1	0	Repair alarm: A request has encountered a hard SBE or MUE, and there are no repair resources available.	

Table 21: ERRSTAT[6:0] Bit Definitions – DRAM Errors

ERRSTAT Bit							Description	Response Packet Type	Notes
6	5	4	3	2	1	0			
0	0	1	0	0	0	0	SBE occurred (this status is normally disabled) - Data is corrected before being returned to the requester. The SBE is scrubbed and tested to determine if it is a hard error, in which case it is dynamically repaired if dynamic repair is enabled and sufficient repair resources are available.	Read or write response packet (TAG = tag of corresponding request)	1
0	0	1	1	1	1	1	MUE has occurred - this indicates that a multiple uncorrectable error has occurred in the DRAM for read data. The DINV (data invalid) flag will also be active. Can also be unsolicited if MUE encountered during patrol scrub.	Read or write response packet (TAG = tag of corresponding request)	1

Note: 1. The response packet type varies depending upon what type of packet the DRAM error occurs in.
Read Response packet used when SBE or MUE occurs during read response.
Write Response packet used when SBE or MUE occurs during atomic operation.
Error Response packet used when SBE or MUE occurs during posted atomic operation or patrol scrub.

Table 22: ERRSTAT[6:0] Bit Definitions – Link Errors

ERRSTAT Bit							Description	Response Packet Type
6	5	4	3	2	1	0		
0	1	0	0	0	0	0	Link 0 retry in progress: Link 0 has successfully initiated recovery from a link error and is executing one or more link retries (within the retry attempt limit).	Error response packet (TAG = CUB number)
0	1	0	0	0	0	1	Link 1 retry in progress: Link 1 has successfully initiated recovery from a link error and is executing one or more link retries (within the retry attempt limit).	
0	1	0	0	0	1	0	Link 2 retry in progress: Link 2 has successfully initiated recovery from a link error and is executing one or more link retries (within the retry attempt limit).	
0	1	0	0	0	1	1	Link 3 retry in progress: Link 3 has successfully initiated recovery from a link error and is executing one or more link retries (within the retry attempt limit).	

Table 23: ERRSTAT[6:0] Bit Definitions – Protocol Errors

ERRSTAT Bit							Description	Response Packet Type	Notes
6	5	4	3	2	1	0			
0	1	1	0	0	0	0	Invalid command: An unsupported command existed in a request packet.	Write response packet (TAG = tag of corresponding request)	
0	1	1	0	0	0	1	Invalid length: An invalid length was specified for the given command in a request packet.	Write response packet (TAG = tag of corresponding request)	1
0	1	1	0	0	1	0	Non-existent cube: CUB field within Request Packet Header does not exist.	Write response packet (TAG = tag of corresponding request)	

Note: 1. LNG ≤17
If LNG >17, a fatal error occurs.

Table 24: ERRSTAT[6:0] Bit Definitions – Vault Critical Errors

ERRSTAT Bit							Description	Response Packet Type
6	5	4	3	2	1	0		
1	1	0	0	0	0	0	Vault 0 critical error: Command/Address parity error has not cleared after the command/address retry count threshold has been met. Vault 0 command execution is halted until a reset is performed.	Error response packet (TAG = CUB number)
1	1	0	0	0	0	1	Vault 1 critical error	
1	1	0	0	0	1	0	Vault 2 critical error	
1	1	0	0	0	1	1	Vault 3 critical error	
1	1	0	0	1	0	0	Vault 4 critical error	
1	1	0	0	1	0	1	Vault 5 critical error	
1	1	0	0	1	1	0	Vault 6 critical error	
1	1	0	0	1	1	1	Vault 7 critical error	
1	1	0	1	0	0	0	Vault 8 critical error	
1	1	0	1	0	0	1	Vault 9 critical error	
1	1	0	1	0	1	0	Vault 10 critical error	
1	1	0	1	0	1	1	Vault 11 critical error	
1	1	0	1	1	0	0	Vault 12 critical error	
1	1	0	1	1	0	1	Vault 13 critical error	
1	1	0	1	1	1	0	Vault 14 critical error	
1	1	0	1	1	1	1	Vault 15 critical error	
1	0	1	0	0	0	0	Vault 16 critical error	
1	0	1	0	0	0	1	Vault 17 critical error	
1	0	1	0	0	1	0	Vault 18 critical error	
1	0	1	0	0	1	1	Vault 19 critical error	
1	0	1	0	1	0	0	Vault 20 critical error	
1	0	1	0	1	0	1	Vault 21 critical error	
1	0	1	0	1	1	0	Vault 22 critical error	
1	0	1	0	1	1	1	Vault 23 critical error	
1	0	1	1	0	0	0	Vault 24 critical error	
1	0	1	1	0	0	1	Vault 25 critical error	
1	0	1	1	0	1	0	Vault 26 critical error	
1	0	1	1	0	1	1	Vault 27 critical error	
1	0	1	1	1	0	0	Vault 28 critical error	
1	0	1	1	1	0	1	Vault 29 critical error	
1	0	1	1	1	1	0	Vault 30 critical error	
1	0	1	1	1	1	1	Vault 31 critical error	

Table 25: ERRSTAT[6:0] Bit Definitions – Fatal Errors

ERRSTAT Bit							Description	Response Packet Type	Notes
6	5	4	3	2	1	0			
1	1	1	0	0	0	0	Link 0 retry failed and was unsuccessful after the retry limit was reached.	Error response packet (TAG = CUB number)	
1	1	1	0	0	0	1	Link 1 retry failed and was unsuccessful after the retry limit was reached.		
1	1	1	0	0	1	0	Link 2 retry failed and was unsuccessful after the retry limit was reached.		
1	1	1	0	0	1	1	Link 3 retry failed and was unsuccessful after the retry limit was reached.		
1	1	1	1	0	0	0	Input Buffer Overrun Link 0: The Host has issued too many FLITs and has overrun the Link Input Buffer, or has issued a MODE CMD before the previous MODE CMD has returned a response packet.		
1	1	1	1	0	0	1	Input Buffer Overrun Link 1: The Host has issued too many FLITs and has overrun the Link Input Buffer, or has issued a MODE CMD before the previous MODE CMD has returned a response packet.		
1	1	1	1	0	1	0	Input Buffer Overrun Link 2: The Host has issued too many FLITs and has overrun the Link Input Buffer, or has issued a MODE CMD before the previous MODE CMD has returned a response packet.		
1	1	1	1	0	1	1	Input Buffer Overrun Link 3: The Host has issued too many FLITs and has overrun the Link Input Buffer, or has issued a MODE CMD before the previous MODE CMD has returned a response packet.		
1	1	1	1	1	0	1	Reliability temperature threshold: The internal temperature for reliable operation has been reached. Any additional temperature increase may be harmful.		1
1	1	1	1	1	1	0	Packet length fatal error: Packet length larger than the MAX supported size (LNG > 17 FLITs).		1
1	1	1	1	1	1	1	Internal fatal error: This group includes, but is not limited to, CRC error at vault controller, internal state machine errors, and internal buffer underrun/overruns.		1

Note: 1. The input buffer overrun fatal error only happens when the HMC input buffer actually overruns, not when the host sends more FLITs than it has tokens for.

Flow Packets

Flow packets are generated at the link master to convey information for link flow control and link retry control to the link slave on the other end of the link. They are not part of the transaction layer. The link slave extracts the control information from the flow packets and then removes the flow packets from the packet stream without forwarding them to the link input buffer. For this reason flow packets are not subject to normal packet flow control. The link master can generate and transmit flow packets without requiring tokens, and it will not decrement the token count when it transmits flow packets.

Flow packets use the request packet header and tail layouts described in Request Packets. The flow packet commands are described in Flow Commands.

Some header fields and tail fields within the flow packets are not used and should be set to 0, as specified in the Valid Field and Command Summary Table.

Poisoned Packets

A poisoned packet is uniquely identified by the fact that its CRC is inverted from the correct CRC value. Packet CRC verification logic of both the requester and the responder should include checking for an inverted value of good CRC. A poisoned packet starts out as a good packet (as identified by a good CRC) but for various reasons can become poisoned. The most common cause of a poisoned packet is a link error, as described in Link Retry. Other examples of how a packet becomes poisoned are described in Packet Integrity. There are both valid and invalid fields within the header and tail of poisoned packets as is described in the Valid Field and Command Summary Table. Receiving a poisoned packet does not trigger a link retry.

Poisoned Packet Rules

- A packet of any valid length can be poisoned.
- NULL FLITs will not be poisoned.
- If a link slave (LS) in the requester or responder receives a previously poisoned packet (the packet was poisoned before it was transmitted on the link) of any valid length, the LS must extract the FRP, RRP, and RTC fields as in any other received packet. After extraction, the LS has the option of dropping the packet or forwarding the packet if the link slave is using cut-through routing, based on the packet routing algorithm that the link slave normally executes (evaluating the CUB and ADRS field of requests or evaluating the SLID field of responses). CUB and ADRS fields are not valid in poisoned packets, and could lead to routing the poison packet to a non-intended destination. This unintended routing must not cause a routing error to be reported.
  - Previously poisoned packets traveling across the link use tokens like any other valid packet. The link slave should return the correct number of tokens when a previously poisoned packet is pulled out of the Link Input Buffer. If the link slave drops a previously poisoned packet, it must still return the correct number of tokens back to the other side of the link.
- The routing fabric and all packet destinations must be able to tolerate stray poisoned packets of any valid length. At any point in the routing fabric the packet could be dropped, but if cut-through routing applies (packet routing begins before the tail arrives indicating that the packet is poisoned), the packet may continue to propagate. All normal credit and token handling applies to the propagating poisoned packet.

Credits or tokens for any previously poisoned packet that has been dropped must also be returned.

- If the previously poisoned packet is routed to the link master of an outgoing link:
  - The link master (LM) treats it like any other valid packet, where the LM embeds FRP, RRP, and RTC values into the tail of the poisoned packet. As the LM recalculates the CRC, which now includes these embedded values, it must result in another poisoned CRC state (being an inverted value of the newly generated CRC value). The poisoned packet will also be temporarily stored in the Link Retry Buffer when it is transmitted on the link.
- A packet destination (including a host receiving packets from a cube) must be able to tolerate receiving a poisoned packet of any valid length and silently dropping it. Note that a poisoned packet could have corrupted CUB, ADRS, ERRSTAT, and/or TAG fields, along with corrupted data FLITs if present.

Request Commands

All request commands are issued by the requester (host or HMC link configured as pass-thru link) to the responder (HMC link configured as host link), using request packets described in Request Packets. Every request must have a unique tag included in the request packet header. A corresponding response packet (one that includes the same tag) will be issued by the responder.

All request packets are saved in the link retry buffer as they cross a link. This means that these packets are retried if an error occurs on their transfer.

Table 26: Transaction Layer – Request Commands

Command Description	Symbol	CMD Bit							Request Packet Length in FLITs	Response Returned	Response Packet Length in FLITs	Atomic FLAG
		6	5	4	3	2	1	0				
WRITE Requests												
16-byte WRITE request	WR16	0	0	0	1	0	0	0	2	WR_RS	1	0
32-byte WRITE request	WR32	0	0	0	1	0	0	1	3	WR_RS	1	0
48-byte WRITE request	WR48	0	0	0	1	0	1	0	4	WR_RS	1	0
64-byte WRITE request	WR64	0	0	0	1	0	1	1	5	WR_RS	1	0
80-byte WRITE request	WR80	0	0	0	1	1	0	0	6	WR_RS	1	0
96-byte WRITE request	WR96	0	0	0	1	1	0	1	7	WR_RS	1	0
112-byte WRITE request	WR112	0	0	0	1	1	1	0	8	WR_RS	1	0
128-byte WRITE request	WR128	0	0	0	1	1	1	1	9	WR_RS	1	0
256-byte WRITE request	WR256	1	0	0	1	1	1	1	17	WR_RS	1	0
MODE WRITE request	MD_WR	0	0	1	0	0	0	0	2	MD_WR_RS	1	0
POSTED WRITE Requests												
16-byte POSTED WRITE request	P_WR16	0	0	1	1	0	0	0	2	None	None	None
32-byte POSTED WRITE request	P_WR32	0	0	1	1	0	0	1	3	None	None	None

Table 26: Transaction Layer – Request Commands (Continued)

Command Description	Symbol	CMD Bit							Request Packet Length in FLITs	Response Returned	Response Packet Length in FLITs	Atomic FLAG
		6	5	4	3	2	1	0				
48-byte POSTED WRITE request	P_WR48	0	0	1	1	0	1	0	4	None	None	None
64-byte POSTED WRITE request	P_WR64	0	0	1	1	0	1	1	5	None	None	None
80-byte POSTED WRITE request	P_WR80	0	0	1	1	1	0	0	6	None	None	None
96-byte POSTED WRITE request	P_WR96	0	0	1	1	1	0	1	7	None	None	None
112-byte POSTED WRITE request	P_WR112	0	0	1	1	1	1	0	8	None	None	None
128-byte POSTED WRITE request	P_WR128	0	0	1	1	1	1	1	9	None	None	None
256-byte POSTED WRITE request	P_WR256	1	0	1	1	1	1	1	17	None	None	None
READ Requests												
16-byte READ request	RD16	0	1	1	0	0	0	0	1	RD_RS	2	0
32-byte READ request	RD32	0	1	1	0	0	0	1	1	RD_RS	3	0
48-byte READ request	RD48	0	1	1	0	0	1	0	1	RD_RS	4	0
64-byte READ request	RD64	0	1	1	0	0	1	1	1	RD_RS	5	0
80-byte READ request	RD80	0	1	1	0	1	0	0	1	RD_RS	6	0
96-byte READ request	RD96	0	1	1	0	1	0	1	1	RD_RS	7	0
112-byte READ request	RD112	0	1	1	0	1	1	0	1	RD_RS	8	0
128-byte READ request	RD128	0	1	1	0	1	1	1	1	RD_RS	9	0
256-byte READ request	RD256	1	1	1	0	1	1	1	1	RD_RS	17	0
MODE READ request	MD_RD	0	1	0	1	0	0	0	1	MD_RD_RS	2	0
ARITHMETIC ATOMICS												
Dual 8-byte signed add immediate	2ADD8	0	0	1	0	0	1	0	2	WR_RS	1	No
Single 16-byte signed add immediate	ADD16	0	0	1	0	0	1	1	2	WR_RS	1	No
Posted dual 8-byte signed add immediate	P_2ADD8	0	1	0	0	0	1	0	2	None	None	None
Posted single 16-byte signed add immediate	P_ADD16	0	1	0	0	0	1	1	2	None	None	None
Dual 8-byte signed add immediate and return	2ADDS8R	1	0	1	0	0	1	0	2	RD_RS	2	Yes
Single 16-byte signed add immediate and return	ADDS16R	1	0	1	0	0	1	1	2	RD_RS	2	Yes

Table 26: Transaction Layer – Request Commands (Continued)

Command Description	Symbol	CMD Bit							Request Packet Length in FLITs	Response Returned	Response Packet Length in FLITs	Atomic FLAG
		6	5	4	3	2	1	0				
8-byte increment	INC8	1	0	1	0	0	0	0	1	WR_RS	1	Yes
Posted 8-byte increment	P_INC8	1	0	1	0	1	0	0	1	None	–	No
BOOLEAN ATOMICS												
16-byte XOR	XOR16	1	0	0	0	0	0	0	2	RD_RS	2	Yes
16-byte OR	OR16	1	0	0	0	0	0	1	2	RD_RS	2	Yes
16-byte NOR	NOR16	1	0	0	0	0	1	0	2	RD_RS	2	Yes
16-byte AND, bitwise clear, read and clear	AND16	1	0	0	0	0	1	1	2	RD_RS	2	Yes
16-Byte NAND	NAND16	1	0	0	0	1	0	0	2	RD_RS	2	Yes
COMPARISON ATOMICS												
8-byte compare and swap if greater than	CASGT8	1	1	0	0	0	0	0	2	RD_RS	2	Yes
16-byte compare and swap if greater than	CASGT16	1	1	0	0	0	1	0	2	RD_RS	2	Yes
8-byte compare and swap if less than	CASLT8	1	1	0	0	0	0	1	2	RD_RS	2	Yes
16-byte compare and swap less than	CASLT16	1	1	0	0	0	1	1	2	RD_RS	2	Yes
8-byte compare and swap if equal	CASEQ8	1	1	0	0	1	0	0	2	RD_RS	2	Yes
16-byte compare and swap if zero	CAS-ZERO16	1	1	0	0	1	0	1	2	RD_RS	2	Yes
8-byte equal	EQ8	1	1	0	1	0	0	1	2	WR_RS	1	Yes
16-byte equal	EQ16	1	1	0	1	0	0	0	2	WR_RS	1	Yes
BITWISE ATOMICS												
8-byte bit write	BWR	0	0	1	0	0	0	1	2	WR_RS	1	No
Posted 8-byte bit write	P_BWR	0	1	0	0	0	0	1	2	None	–	No
8-byte bit write with return	BWR8R	1	0	1	0	0	0	1	2	RD_RS	2	Yes
16-byte swap or ex-change	SWAP16	1	1	0	1	0	1	0	2	RD_RS	2	No

Note: 1. All CMD bit encode values not defined in this table are reserved.

READ and WRITE Request Commands

READ and WRITE request commands are issued by the requester to access the DRAM memory. They are block commands that access a contiguous number of memory-addressable bytes with 16-byte granularity, meaning they can begin and end on any 16-byte boundary up to a maximum length of 256 bytes. Due to the 16-byte granularity of

READ and WRITE request commands, the four LSBs of the request address are ignored. As shown in Table 19, there are unique commands for each supported memory access length.

READ and WRITE operations will not cross a row (page) boundary within the DRAM array. The maximum block size selected in the address map mode register determines the boundary at which the addressing will wrap if the block access goes beyond the end of the maximum block size. All read requests are acknowledged with a read response packet that includes the tag of the request and the data payload. All write requests (nonposted) are acknowledged with an explicit write response packet that includes the tag of the write request. If a READ or WRITE request command specifies a data payload length that is longer than the configured maximum block size, an error will be detected but the request will still be executed.

POSTED WRITE Request Commands

POSTED WRITE request commands operate similarly to standard write requests, except that there is no response returned to the requester. As the HMC does not use the TAG field of posted write requests, the host can populate this field with any value. If an error occurs during the execution of the write request, an Error Response packet will be generated indicating the occurrence of the error to the host. In this case, the write request's tag will not be included in the Error Response packet.

ATOMIC Request Commands

Atomic requests involve reading 16 bytes of data from DRAM (as determined by the request ADRS field), performing an operation on the data through the use of a 16-byte operand (either included in the request packet or pre-defined), and then writing the results back to the same location in DRAM. The read-update-write sequence occurs atomically, meaning that subsequent requests to the same bank are scheduled after the write of the atomic request is complete. Many atomic requests are similar to a 16-byte write request in that they have a 16-byte data payload in the request packet, and a response may or may not be returned (dependent on the specific atomic command). The resulting data from the ATOMIC operation is not returned in a response command. Depending on the command, original memory operand maybe returned. The following two sections describe the currently defined ATOMIC operations supported by the HMC.

Table 27: Atomic Quick Reference

Arithmetic	Bitwise	Boolean	Comparison
Dual 8-byte signed add immediate[1]	16-byte swap (exchange)	16-byte AND (read and clear, bitwise clear)	8-byte CAS if equal
Dual 8-byte signed add immediate with return	8-byte bit write[1]	16-byte NAND	16-byte CAS if zero
Single 16-byte signed add immediate[1]	8-byte bit write with return (8-byte swap)	16-byte OR (bitwise SET)	CAS if greater than (8-byte; 16-byte)
Single 16-byte signed add immediate with return	–	16-byte NOR	CAS if less than (8-byte; 16-byte)
8-byte increment	–	16-byte XOR (bitwise TOGGLE)	Compare if equal (8-byte; 16-byte)

Note: 1. Implementation is identical to HMC-15G-SR revision 1.x of Protocol specification.

Addressing Atomic Operations

For 16-Byte atomic operations, the 4 LSB (bits 3-0) of the ADRS field within the Atomic request header are ignored as described in the Default Address Map Mode Table.

For 8-Byte atomic operations, the 3 LSB (2-0) of the ADRS field are ignored, and ADRS bit 3 is used to determine whether the 8-Byte block is the LSB 8-Byte block (bit 3 = 0) or the MSB 8-Byte block (bit 3 = 1) of the 16-Byte memory operand.

Table 28: Address Field in Request Header

Bits	34:4	3	2	1	0
Value	x	**x**	**0**	**0**	**0**

Arithmetic Atomics

Dual 8-Byte Signed Add Immediate

This atomic request performs two parallel add immediate operations. Each add operation uses an 8-byte memory operand from DRAM and a 4-byte two's complement signed integer immediate operand from the atomic request data payload. If the result exceeds 2^{64}-1 the carry-out of the most significant bit is dropped and the result rolls over. The results are written back to DRAM, overwriting the original memory operands.

Table 29: Dual 8-Byte Signed Add Immediate

Operation is identical to revision 1.x Protocol [HMC-15G-SR]

Symbol	Request Data Payload	Atomic Flag	CMD Bits
2ADD8	Two 8-byte immediate operands	No	0010010
P_2ADD8	Two 8-byte immediate operands	No	0100010

Table 30: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 2								Memory operand 1							

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 31: Immediate Operands Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3 (MSB)	2	1	0 (LSB)
Field	4 bytes of zeros				Immediate operand 2				4 bytes of zeros				Immediate operand 1			

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

Single 16-Byte Signed Add Immediate

This request performs a single add immediate operation. It uses a 16-byte memory operand from DRAM and an 8-byte two's complement signed integer immediate operand from the atomic request data payload. If the result exceeds 2^{128}-1 the carry-out of the most significant bit is dropped and the result rolls over. The result is written back to the DRAM, overwriting the original memory operand.

Table 32: Single 16-Byte Signed Add Immediate

Operation is identical to revision 1.x Protocol [HMC-15G-SR]

Symbol	Request Data Payload	Atomic Flag	CMD Bits
ADD16	16-byte immediate operand	No	0010011
P_ADD16	16-byte immediate operand	No	0100011

Table 33: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1															

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 34: Immediate Operand Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	8 bytes of zeros								Immediate operand 1							

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

Dual 8-Byte Signed Add Immediate and Return

This atomic request performs two parallel add immediate operations. Each add operation uses an 8-byte memory operand from DRAM and a 8-byte two's complement signed integer immediate operand from the atomic request data payload. If the result exceeds 2^{64}-1, the carry-out of the most significant bit is dropped, the result rolls over, and an overflow occurs. The overflow is indicated by the atomic flag equaling 1. The results are written back to DRAM, overwriting the original memory operands.

Table 35: Dual 8-Byte Signed Add Immediate and Return

Symbol	Request Data Payload	Response Data Payload	Atomic Flag	CMD Bits
2ADD8SR	Two 8-byte immediate operands	Two 8-byte original memory operands	Overflow = 1	1010010

Table 36: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 2								Memory operand 1							

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 37: Immediate Operands Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Immediate operand 2								Immediate operand 1							

Note: 1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 38: Operands Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Original memory operand 2								Original memory operand 1							

Note: 1. Each byte field contains bit positions [7(MSB):0(LSB)].

16-Byte Signed Add Immediate and Return

This atomic request performs one 16-byte add immediate operations. This operation adds 16-byte memory operand in DRAM with the 16-byte two's complement signed integer immediate operand from the atomic request data payload. If the result exceeds 2^{128} -1, the carry-out of the most significant bit is dropped, the result rolls over, and an overflow occurs. The overflow is indicated by the atomic flag equaling 1. The results are written back to DRAM, overwriting the original memory operands.

Table 39: 16-Byte Signed Add Immediate and Return

Symbol	Request Data Payload	Response Data Payload	Atomic Flag	CMD Bits
ADD16SR	16-byte immediate operand	16-byte original memory operand	Overflow = 1	1010011

Table 40: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)[1]	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1															

Notes: 1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 41: Immediate Operand Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)[1]	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Immediate operand 1															

Note: 1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 42: Operand Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)[1]	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Original memory operand 1															

Note: 1. Each byte field contains bit positions [7(MSB):0(LSB)].

8-Byte Increment

8-byte memory operand 1 is read from DRAM, incremented by 1, and written back to the same location in DRAM. (Immediate operand is not used). If the result exceeds 2^{64}-1, the carry-out of the most significant bit is dropped and an overflow occurs, indicated in the atomic flag.

Table 43: 8-Byte Increment

Symbol	Request Data Payload	Response Data Payload	Atomic Flag	CMD Bits
INC8 (Add 1)	None	None	Overflow = 1	1010000
P_ INC8	None	None	No	1010100

Table 44: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Memory operand 1 (when ADRS 3 = 1)								Memory operand 1 (when ADRS 3 = 0)							

Note: 1. Each byte field contains bit positions [7(MSB):0(LSB)].

Bitwise Atomics

8-Byte Bit Write

The bit write request is a 2-FLIT request packet that provides a 0- to 8-byte bit-write capability as indicated by write data mask bits included in the data payload. The specific bits to be written within the 8-byte block are identified by the data mask bits in bytes[15:8] of the 16-byte data payload within the request packet as shown in the following table.

The bit(s) to be written must be positioned in the correct bit position within the write data field, as identified with the cleared bit(s) in the data mask field. For example, if the data mask = 0x00FFFFFFFFFF00FF, then bytes 7 and 1 of the write data field will be written into the same bytes in the memory. If the data mask = 0xFFFFFFFFFFFFFFFF, the READ-UPDATE-WRITE operation will be performed with the original value of the data being rewritten into the memory.

Table 45: 8-Byte Bit Write

Operation is identical to revision 1.x Protocol [HMC-15G-SR]

Symbol	Request Data Payload	Response Data Payload	Atomic Flag	CMD Bits
BWR (Bit write)	8-byte bit mask, 8-byte write data	None	No	0010001
P_ BWR	8-byte bit mask, 8-byte write data	None	No	0100001

Table 46: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1 (when ADRS 3 = 1)								Memory operand 1 (when ADRS 3 = 0)							

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 47: Immediate Operand Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Bit mask								Write data							

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

8-Byte Bit Write With Return

The bit write request is a 2-FLIT request packet that provides a 0- to 8-byte bit-write capability as indicated by write data mask bits included in the data payload. The specific bits to be written within the 8-byte block are identified by the data mask bits in bytes[15:8] of the 16-byte data payload within the request packet as shown in the following table.

The bit(s) to be written must be positioned in the correct bit position within the write data field, as identified with the cleared bit(s) in the data mask field. For example, if the data mask = 0x00FFFFFFFFFF00FF, then bytes 7 and 1 of the write data field will be written into the same bytes in the memory. If the data mask = 0xFFFFFFFFFFFFFFFF, the READ-UPDATE-WRITE operation will be performed with the original value of the data being rewritten into the memory.

Table 48: 8-Byte Bit Write With Return

Operation is identical to revision 1.x Protocol [HMC-15G-SR]

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
BWR8R (Bit write, bit mask, 8-byte swap)	8-byte bit mask, 8-byte write data	8-byte original memory operand	If any bit is changed, then AF = 1	1010001

Table 49: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1 (when ADRS 3 = 1)								Memory operand 1 (when ADRS 3 = 0)							

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 50: Immediate Operand Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Bit mask								Write data							

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 51: Operand Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	8 bytes of zeros (in the upper 8 bytes)								Original memory operand 1							

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

16-Byte Swap

This request performs an 16-byte swap writing the 16-byte immediate operand 1 in the request packet in to the memory location while returning the original 16-byte memory operand 1 at the memory location in the response.

Table 52: 16-Byte Swap

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
SWAP16 (Exchange)	16-byte immediate operand	16-byte original memory operand	No	1010001

Table 53: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1															

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 54: Immediate Operand Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Immediate operand 1															

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 55: Operand Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Original memory operand 1															

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

Boolean Atomics

For each of the following boolean atomic requests, a Boolean operation is performed using the memory operand 1 (from DRAM) and the immediate operand 1 (included in the atomic request packet).

Table 56: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1															

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 57: Immediate Operand Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Immediate operand 1															

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 58: Operand Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Memory operand 1															

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

16-Byte AND

For this atomic request, execute a bitwise AND on memory operand 1 using immediate operand as the AND mask, write the AND result back into the memory location, and return original memory operand 1 in response packet. Bitwise clear is the AND command where immediate operand selects which bits to clear by indicating 0 in the bit locations to be cleared. Read and clear is implemented when immediate operand is all 0s. The atomic flag is set if any bit has been changed.

Table 59: 16-Byte AND

Symbol	Request Data Payload	Response	Atomic Flag	CMD Bits
AND16 (Bitwise clear, read and clear)	16-byte immediate operand	16-byte original memory operand	If any bit is changed, then AF = 1	1000011

Table 60: AND Bitwise Truth Table

Immediate Operand	Original Memory Operand	Result	Atomic Flag
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

16-Byte NAND

For this atomic request, execute a bitwise NAND on memory operand 1 using immediate operand as the NAND mask, write the NAND result back into the memory location, and return original memory operand 1 in response packet. The atomic flag is set if any bit has been changed.

Table 61: 16-Byte NAND

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
NAND16	16-byte immediate operand	16-byte original memory operand	If any bit is changed, then AF = 1	1000100

Table 62: NAND Bitwise Truth Table

Immediate Operand	Original Memory Operand	Result	Atomic Flag
0	0	1	1
0	1	1	0
1	0	1	1
1	1	0	1

16-Byte NOR

For this atomic request, execute a bitwise NOR on memory operand 1 using immediate operand as the NOR mask, write the NOR result back into the memory location, and return original memory operand 1 in response packet. The atomic flag is set if any bit has been changed.

Table 63: 16-Byte NOR

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
NOR16	16-byte immediate operand	16-byte original memory operand	If any bit is changed, then AF = 1	1000010

Table 64: NOR Bitwise Truth Table

Immediate Operand	Original Memory Operand	Result	Atomic Flag
0	0	1	1
0	1	0	1
1	0	0	0
1	1	0	1

16-Byte OR

For this atomic request, execute a bitwise OR on memory operand 1 using immediate operand as the OR mask, write the OR result back into the memory location, and return original memory operand 1 in response packet. The atomic flag is set if any bit has been changed.

Table 65: 16-Byte OR

Symbol	Request Data	Response Data	Atomic Flag	CMD Bits
OR16 (TEST and SET)	16-byte immediate operand	16-byte original memory operand	If any bit is changed, then AF = 1	1000001

Table 66: OR Bitwise Truth Table

Immediate Operand	Original Memory Operand	Result	Atomic Flag
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	0

16-Byte XOR

For this atomic request, execute a bitwise XOR on memory operand 1 using immediate operand as the XOR mask, write the XOR value back into the memory location, and return original memory operand 1 in response packet.

Table 67: 16-Byte XOR

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
XOR16 (Bitwise toggle)	16-byte immediate operand	16-byte original memory operand	If any bit is changed, then AF = 1	1000000

Table 68: XOR Bitwise Truth Table

Immediate Operand	Original Memory Operand	Result	Atomic Flag
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Comparison Atomics

Comparison Atomics assume unsigned values where ever applicable.

8-Byte Compare and Swap if Greater Than

For this atomic request, the request immediate operand in the request data packet payload is compared with the memory operand (the comparison assumes unsigned numbers), and the greater of the two is written back into the memory location. The original memory operand value is returned with the response data payload. If the immediate operand is greater than the memory operand, the atomic flag is set.

The comparison assumes unsigned values.

Table 69: 8-Byte Compare and Swap if Greater Than

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
CASGT8 (Maximum)	8-byte immediate operand	8-byte original memory operand	If immediate operand > original memory operand, then AF = 1	1100000

Table 70: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1 (when ADRS 3 = 1)								Memory operand 1 (when ADRS 3 = 0)							

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 71: Immediate Operands Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field									Immediate operand 1							

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 72: Operands Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	8 bytes of zeros (in the upper 8 bytes)								Original memory operand 1							

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

8-Byte Compare and Swap if Less Than

For this atomic request, the memory operand is compared with the immediate operand in the request (the comparison assumes unsigned numbers), and the smaller of the two is written back into the memory location. The original memory operand value is returned with the response data payload.

The comparison assumes unsigned values.

Table 73: 8-Byte Compare and Swap if Less Than

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
CASLT8 (Minimum)	8-byte immediate operand	8-byte original memory operand	If immediate operand < memory operand, then AF = 1	1100001

Table 74: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1 (when ADRS 3 = 1)								Memory operand 1 (when ADRS 3 = 0)							

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 75: Immediate Operands Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field									Immediate operand 1							

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 76: Operands Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	8 bytes of zeros (in the upper 8 bytes)								Original memory operand 1							

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

8-Byte Equal To

For this atomic request, the memory operand is compared with the immediate operand in the request to determine if they are equal. The operation only results in determining the atomic flag; the DRAM location is not modified. If the immediate operand is equal to the memory operand, the atomic flag is set.

Table 77: 8-Byte Equal To

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
EQ8 (Same)	8-byte immediate operand	None	If immediate operand = memory operand, then AF = 1	1101001

Table 78: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1 (when ADRS 3 = 1)								Memory operand 1 (when ADRS 3 = 0)							

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 79: Immediate Operands Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field									Immediate operand 1							

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

16-Byte Compare and Swap if Greater Than

For this atomic request, the request immediate operand in the request data packet payload is compared with the memory operand (the comparison assumes unsigned numbers), and the greater of the two is written back into the memory location. The original memory operand value is returned with the response data payload. If the immediate operand is greater than the memory operand, the atomic flag is set.

The comparison assumes unsigned values.

Table 80: 16-Byte Compare and Swap if Greater Than

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
CASGT16 (Maximum)	16-byte immediate operand	16-byte original memory operand	If immediate operand > memory operand, then AF = 1	1100010

Table 81: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1															

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 82: Immediate Operand Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Immediate operand 1															

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 83: Operand Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Original memory operand 1															

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

16-Byte Compare and Swap if Less Than

For this atomic request, the memory operand is compared with the immediate operand in the request (the comparison assumes unsigned numbers), and the smaller of the two is written back into the memory location. The original memory operand value is returned with the response data payload. If the immediate operand is less than the memory operand, the atomic flag is set.

The comparison assumes unsigned values.

Table 84: 16-Byte Compare and Swap if Less Than

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
CASLT16 (Minimum)	16-byte immediate operand	16-byte original memory operand	If immediate operand < original memory operand, then AF = 1	1100011

Table 85: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1															

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 86: Immediate Operand Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Immediate operand 1															

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 87: Operand Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Original memory operand 1															

Note:
1. Each byte field contains bit positions [7(MSB):0(LSB)].

16-Byte Equal To

For this atomic request, the memory operand is compared with the immediate operand in the request to determine if they are equal. The operation only results in determining the atomic flag; the DRAM location is not modified. If the immediate operand is equal to the memory operand, the atomic flag is set.

Table 88: 16-Byte Equal To

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
EQ16	16-byte immediate operand	None	If immediate operand = memory operand, then AF = 1	1101000

Table 89: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1															

Notes:
1. Corresponds to the start of the 16-byte aligned address provided in the request address field.

2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 90: Immediate Operand Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Immediate operand 1															

Note: 1. Each byte field contains bit positions [7(MSB):0(LSB)].

8-Byte Compare and Swap if Equal

This atomic request contains an 8-byte comparison field and an 8-byte immediate operand 1 in a 16B packet payload. The 8-byte comparison field is compared to memory operand 1 at the address location indicated in the request command. If the 8-byte compare field is equal to that of memory operand 1, then the immediate operand is written into memory operand 1, and the atomic flag is set.

This atomic request can also be used for 8-byte compare and swap if zero by setting comparison field to zero.

Table 91: 8-Byte Compare and Swap if Equal

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
CASEQ8	8-byte compare field 8-byte immediate operand	8-byte original memory operand	If compare field = memory operand, then AF = 1	1100100

Table 92: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1 (when ADRS 3 = 1)								Memory operand 1 (when ADRS 3 = 0)							

Notes: 1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 93: Immediate Operands Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Compare field								Immediate operand 1							

Note: 1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 94: Operands Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	8 bytes of zeros (in the upper 8 bytes)								Original memory operand 1							

Note: 1. Each byte field contains bit positions [7(MSB):0(LSB)].

16-Byte Compare and Swap if Zero

This atomic request contains a 16-byte data field and a 16B packet payload. The request reads memory operand and if it is zero, writes the immediate operand into the 16-byte memory operand address. If the memory operand is equal to zero, the atomic flag is set.

Table 95: 16-Byte Compare and Swap if Zero

Symbol	Request Data Payload	Response Data	Atomic Flag	CMD Bits
CASZERO16	16-byte immediate operand	16-byte original memory operand	If memory operand = 0, then AF = 1	1100101

Table 96: Operands from DRAM

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)[1]
Field	Memory operand 1															

Notes: 1. Corresponds to the start of the 16-byte aligned address provided in the request address field.
2. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 97: Immediate Operand Included in Atomic Request Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Immediate operand 1															

Note: 1. Each byte field contains bit positions [7(MSB):0(LSB)].

Table 98: Operand Included in Atomic Response Data Payload

Byte	15 (MSB)	14	13	12	11	10	9	8 (LSB)	7 (MSB)	6	5	4	3	2	1	0 (LSB)
Field	Original memory operand 1															

Note: 1. Each byte field contains bit positions [7(MSB):0(LSB)].

MODE READ and WRITE Request Commands

MODE READ and WRITE request commands access the internal hardware mode and status registers within the logic layer of the HMC. Each mode request accesses up to 32 contiguous bits of the register.

The address field (ADRS) within the mode request packet is redefined to provide a register address, a start field, and a size field as shown in the table below. MODE READ and WRITE request commands must include address values that reference the registers defined in the Register Addendum. A MODE READ request command that contains an address that references a non-existent register results in a MODE READ response that includes all zeros within the data payload. A MODE WRITE request that references a non-existent register will not be performed. There will be no error indication.

Table 99: Mode Request Addressing

Field	Bits	Number of Bits	Description
Unused	33:32	2	Unused bits
Start	31:27	5	**Start field:** Encoded to provide a value from 0 to 31 that points to the starting bit position within the 32 bits of data, 0 being the LSB position and 31 being the most significant bit (MSB) position.
Size	26:22	5	**Size field:** Indicates the number of contiguous bits to read or write (from 1 to 32), and is encoded as follows: 0x0 = 32 bits, 0x1–0x1F = 1–31 bits. The minimum size is 1 bit.
Register address	21:0	22	**Register address field:** References a specific mode or status register. All registers are 32 bits or less in size.

Note: 1. A full 32-bit access would be 0 in both the start and size fields.

The valid data bytes within the data payload in the mode write request packet and the mode read response packet are right-justified at the four least significant bytes as shown in the following table.

Table 100: Valid Data Bytes

Byte	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	Unused												Data			

If less than 32 bits are being accessed, as determined by the start and size fields within the address, the valid data bits are right-justified to the LSBs within the 4-byte data field.

The HMC can only process one mode request at a time. After a mode request is issued, the host(s) must wait for the corresponding response before issuing another mode request on any link. Because of this requirement, the HMC does not support posted MODE WRITE requests.

BIT WRITE Command

The BIT WRITE request is a 2-FLIT request packet that provides a 0- to 8-byte write capability as indicated by write data mask bits included in the data payload. The vault

controller performs a READ-UPDATE-WRITE operation to the DRAM to merge the write data bytes with the existing bytes in the memory. The three LSBs of the byte address field within the address in the request packet header must be 000 when addressing an 8-byte block. The specific bits to be written within the 8-byte block are identified by the data mask bits in bytes[15:8] of the 16-byte data payload within the request packet as shown in the following table.

Table 101: Request Packet Bytes to be Written

Byte	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	Data mask								Write data							

Notes:
1. Data mask: Each bit inhibits writing the corresponding write data bit when 1.
2. Write data: Up to eight bytes of byte-aligned write data.

The bit(s) to be written must be positioned in the correct bit position within the write data field, as identified with the cleared bit(s) in the data mask field. For example, if the data mask = 0x00FFFFFFFFFF00FF, then bytes 7 and 1 of the write data field will be written into the same bytes in the memory. If the data mask = 0xFFFFFFFFFFFFFFFF, the READ-UPDATE-WRITE operation will be performed with the original value of the data being rewritten into the memory.

Response Commands

All response commands are issued by the responder (HMC link configured as host link) to the requester (host or HMC link configured as pass-thru link), using response packets described in Response Packets. Every response other than the error response includes a valid tag that matches the tag in its corresponding request. All response packets are saved in the link retry buffer as they cross a link, which means they are retried if an error occurs on the link.

Table 102: Transaction Layer – Response Commands

Command Description	Symbol	CMD Bit							Packet Length in FLITs
		6	5	4	3	2	1	0	
READ response	RD_RS	0	1	1	1	0	0	0	1 + data FLITs
WRITE response	WR_RS	0	1	1	1	0	0	1	1
MODE READ response	MD_RD_RS	0	1	1	1	0	1	0	2
MODE WRITE response	MD_WR_RS	0	1	1	1	0	1	1	1
ERROR response	ERROR	0	1	1	1	1	1	0	1
Reserved	–	0	1	1	1	1	1	1	–

READ and MODE READ Response Commands

A READ or MODE READ response command is issued by the responder to return requested data to the requester. A read response packet always includes data payload along with the header and tail, and it includes the same tag as its corresponding read request. If an error occurred such that the data is not valid, as indicated in the DINV field in the response tail, the data payload FLITs are still present in the response packet

but they hold invalid data. For this reason, the response packet lengths are always consistent with the corresponding request.

A MODE READ command must include address values that reference the registers defined in the Request Commands section. A MODE READ request command that contains an address that points to a non-existent register will result in a MODE READ response that includes all zeros within the data payload, and the DINV bit will not be set.

WRITE and MODE WRITE Response Commands

A WRITE or MODE WRITE response command provides an explicit write response packet that returns the tag for the original write request. If errors occurred during the transfer or execution of the WRITE request command, status will be included in the ERRSTAT field of the write response packet. A lack of error status bits indicates that the write was successful.

A MODE WRITE command must include address values that reference the registers defined in the Request Commands section. A MODE WRITE request command that contains an address that references a non-existent register will result in a MODE WRITE response; however, the request will not be performed. There will be no error indication.

ERROR Response Command

HMC uses the READ, MODE READ, WRITE, and MODE WRITE response commands to report errors or flags that are associated with a specific request and tag. These error reports are included in the ERRSTAT field of the response packet.

The ERROR response command is used by the HMC to report a critical event when the request type and/or the original request tag are not known. The CUB field within the Error Response packet header contains the value of the cube number where the ERROR RESPONSE command originated. This allows the host to determine from which HMC the Error Response packet originated in a topology with multiple HMCs. To support backward compatibility with revision 1.0 Protocol, the three LSBs of the Tag field within the Error Response packet header also contain the cube number. The ERROR response command is unsolicited and can be configured to transmit ERROR response commands on any or all links except for those in chained topologies, where the error response command must only be routed to a single source link. The following situations cause the HMC to transmit an ERROR response command packet:

- The responder receives a request packet from the requester that is successfully transferred across a link but is internally corrupted before it reaches its destination. In this case the original command and tag are lost and the responder cannot generate a proper response packet or the necessary tag to close the transaction.
- The HMC has an internal error not associated with a transaction.
- The HMC reports a condition or status to the host.

Flow Commands

Flow commands facilitate retry and flow control on the link and are not part of the transaction layer. They are generated at the link master to send information to the other end of the link when no other link traffic is occurring or when a link retry sequence is to be initiated. Flow commands are not forwarded from that point. Flow commands use the request packet header and tail layouts described in the Request Packets section, but are not considered requests, and do not have corresponding response commands. Flow

commands are not subject to token flow control. The link master is allowed to send a flow command even if there are no tokens available.

Table 103: Flow Commands

Command Description	Symbol	CMD Bit						Packet Length in FLITs
		5	4	3	2	1	0	
Null	NULL	0	0	0	0	0	0	1
Retry pointer return	PRET	0	0	0	0	0	1	1
Token return	TRET	0	0	0	0	1	0	1
Init retry	IRTRY	0	0	0	0	1	1	1

NULL Command

A NULL command field of all zeros, along with all zeros in the rest of the FLIT, define a NULL FLIT. NULL FLITs are driven on the link when there are no packets to be transmitted. (Null FLITs are also used during link initialization.) Note that a FLIT of all zeros will have a correct CRC. A poisoned version of the NULL FLIT is not supported.

RETRY POINTER RETURN (PRET) Command

This command is issued by the link master to return retry pointers when there is no other link traffic flowing at the time the pointer is to be returned. It is a single-FLIT packet with no data payload. PRET packets are not saved in the retry buffer, so their SEQ and FRP fields should be set to 0. Tokens should not be returned in these packets, so the RTC field should be set to 0.

TOKEN RETURN (TRET) Command

This command is issued by the link master to return tokens when there is no other link traffic flowing at the time the token is to be returned. It is a single-FLIT packet with no data payload. Token return packets are saved in the retry buffer, so their SEQ and FRP fields are valid.

INIT RETRY (IRTRY) Command

A programmable stream of INIT RETRY (IRTRY) command packets is issued by the link master when it is in the Retry_Init state. This is a single-FLIT packet with no data payload. IRTRY command packets are not saved in the retry buffer, so their SEQ field should be set to 0. Tokens should not be returned in these packets, so the RTC field should also be set to 0 (see the LinkRetry_Init State section).

Valid Field Command Summary

Table 104: Valid Field and Command Summary Table

Symbol	Request		Response			Flow				Poisoned[2]
	READ	WRITE	READ	WRITE	ERROR	NULL	PRET	TRET	IRTRY	All
ADRS	V	V	N/A	N/A	V	0	0	0	0	NV
CMD	V	V	V	V	V	0	V	V	V	NV

Table 104: Valid Field and Command Summary Table (Continued)

Symbol	Request		Response			Flow				Poisoned[2]
	READ	WRITE	READ	WRITE	ERROR	NULL	PRET	TRET	IRTRY	All
CRC	V	V	V	V	V	V	V	V	V	V[2]
CUB	V	V	V	V	V	0	0	0	0	NV
ERRSTAT	N/A	N/A	V	V	V	N/A	N/A	N/A	N/A	NV
FRP	V	V	V	V	V	0	0	V	V[3]	V
LNG	V	V	V	V	V	0	V	V	V	V
RRP	V	V	V	V	V	0	V	V	V	V
RTC	V[4]	V[4]	V	V	V	0	0	V	0	V
SEQ	V	V	V	V	V	0	0	V	0	V
TAG	V	V[5]	V	V	V[6]	0	0	0	0	NV
DINV	N/A	N/A	V	V	V	0	N/A	N/A	N/A	NV
AF	N/A	N/A	V	V	V	0	N/A	N/A	N/A	NV

Notes:
1. V: Valid field
   NV: Invalid field – Do not use
   N/A: Not applicable, field doesn't exist
   0: Field must be 0
2. A poisoned packet is defined by an inverted CRC.
3. FRP[0] = Start Retry Flag
   FRP[1] = Clear Error Abort Flag
4. If using response open loop mode, the value in the RTC field is "Don't Care."
5. The TAG field is not used within POSTED WRITE requests, whereby the host can use any value for TAG, including zero.
6. TAG[2:0] = CUB[2:0] and TAG[10:3] = 0

Transaction Layer Initialization

During link initialization, after the responder stops sending TS1 patterns on the link, the link layer is considered active. Both ends of the link are in the active state, sending NULL FLITs, as described in Step 11 in the Power-On and Initialization section. At this point, the HMC will continue internal initialization.

Transaction layer initialization continues with the following steps:

1. After internal initialization is complete, the responder will send one or more TRET packets that will transfer the number of its link input buffer tokens to the requester. Each TRET packet can transmit a count of up to 64 tokens, therefore there will be multiple TRETs required to transfer the entire number of tokens representing the link input buffer's available space.
2. Once the requester receives at least one TRET packet from the responder, the requester should transmit a series of TRET packets back to the responder carrying the tokens representing the available space in the requester's link input buffer. The HMC can support up to 1023 tokens from the host on each link.
3. After the TRET packets are transmitted from the requester to the responder, the requester can now start sending transaction packets. There is no required time frame when the transaction packets can start.

Within an HMC, all links that are configured as pass-thru links must complete their transaction layer initialization before the links configured as host links begin to send TRET packets upstream. This ensures that all links within a multi-cube topology are initialized once the link configured as a host link in source mode has finished its transaction layer initialization.

Configuration and Status Registers

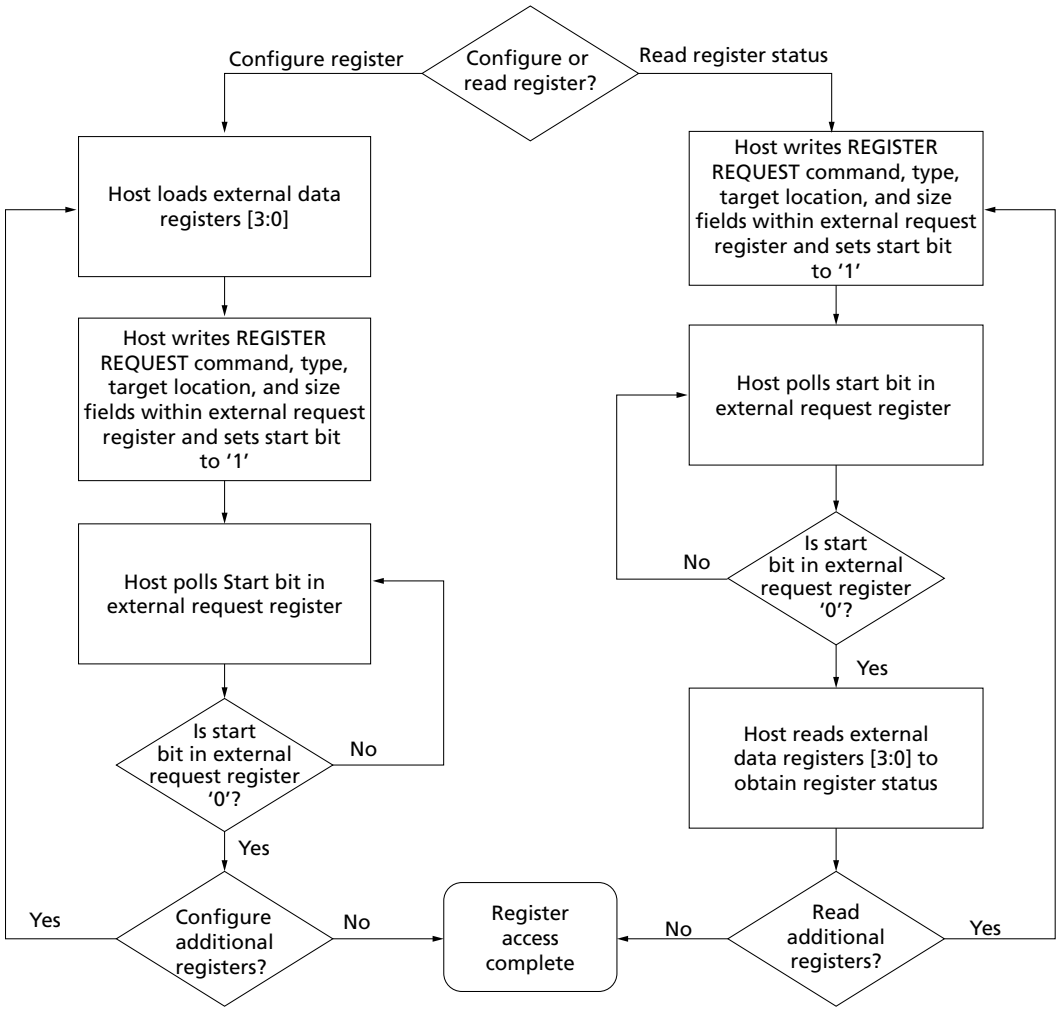
There are two methods supported for accessing the HMC configuration and status registers:

Direct access: This type of access uses the in-band MODE WRITE and MODE READ request commands or the sideband (JTAG/I^2C) to directly read or write the configuration and status registers. See HMC Register Addendum for specific direct access configuration and status registers.

Indirect access through the external data interface (ERI): This type of access utilizes four external data registers and an external request register to indirectly access the HMC configuration and status registers. The external data and request registers are accessible through MODE WRITE and MODE READ requests or sideband (JTAG/I^2C). Specific configuration and status registers accessible through the ERI are not contained within this specification, only the framework to enable this type of register access. See HMC Register Addendum for specific indirect access configuration and status registers. See the following figure for the steps required to access the configuration and status registers through the ERI.

For specific register functionality, including descriptions, addresses, and ERI sequences, please see the HMC Register Addendum.

Figure 18: Configuration and Status Register Access Through ERI



Link Retry

The link transmits data at a very high speed. The data transmission bit failure rate requires a mechanism for correcting failures to increase system reliability and availability. Link retry is a fault-tolerant feature that recovers the link when link errors occur. Link packet integrity is insured with a CRC code that resides in the tail of every packet. It covers all bits in the packet, including the header, any data payload, and the tail. CRC is generated at the link master before the packet bits are serialized and transmitted, and is checked at the link slave after the packet bits are received and deserialized. A mismatch indicates that some of the data within the packet has changed since the original CRC code was generated at the transmitting end of the link. An additional check is provided using an incrementing sequence number included in every packet. If the CRC check is successful upon receipt of the packet, the link slave will verify that the sequence number has a +1 value compared to the last successful packet received.

The link master saves a copy of each packet transmitted across the link. Each packet is held until an acknowledge occurs indicating that the packets have been received without errors. This acknowledgment comes from a retry pointer (included in every trans-

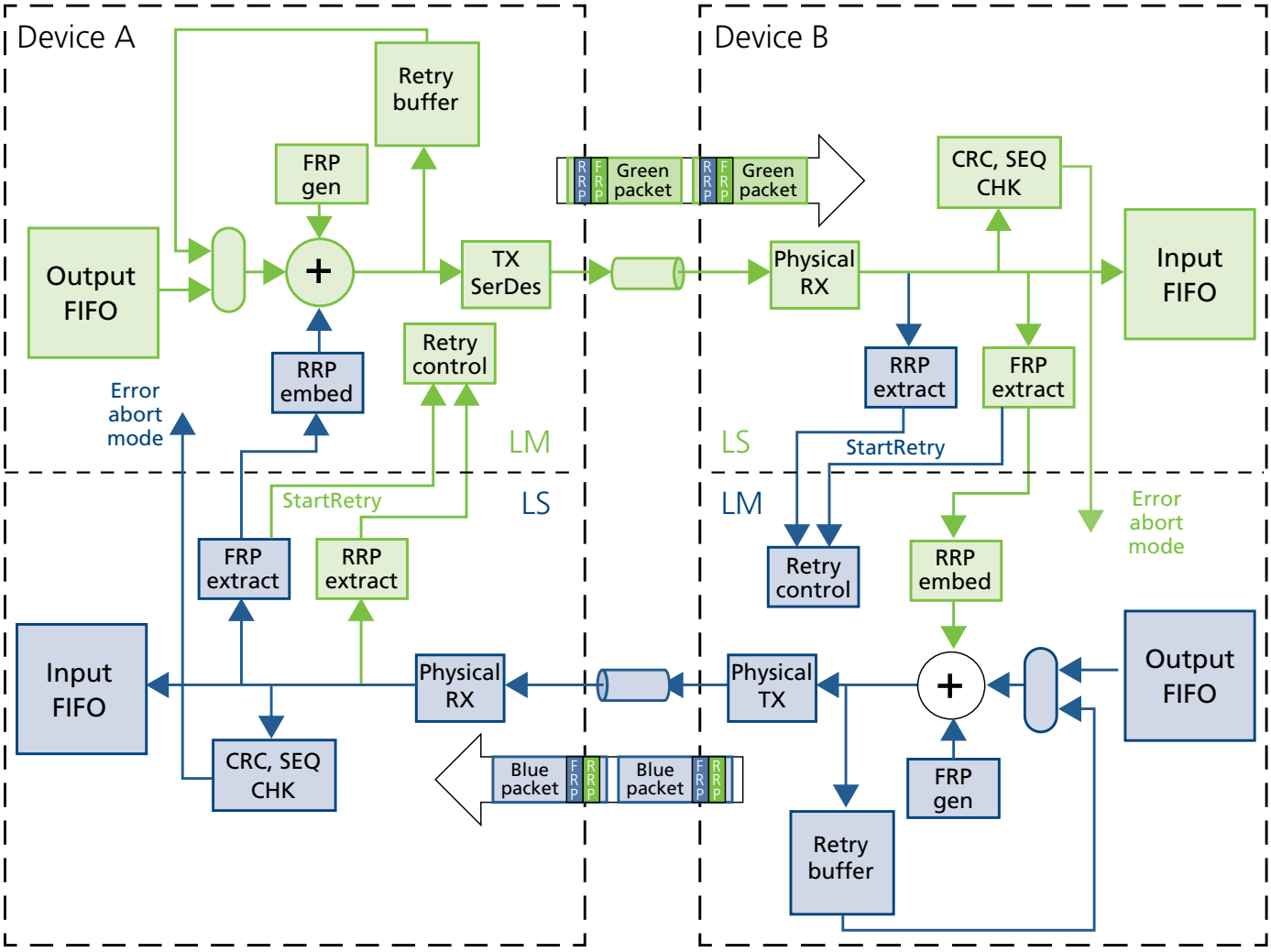
mitted packet); it is returned to the link master embedded in packets traveling in the opposite direction on that link.

If the link slave detects a packet error, it enters error abort mode. This results in the failed packet being poisoned or dropped and all subsequent packets being dropped. Error abort mode triggers the local link master to send a StartRetry flag back to the transmitting end using the other side of the link. The poisoned and dropped packets are then retransmitted by the link master during the retry sequence.

The following figure illustrates an example implementation of the link retry blocks on both ends of a link. The forward retry pointer (FRP) and return retry pointer (RRP) extract and embed blocks are color coded to show the path the retry pointer takes for each link direction.

The link is symmetrical and in the following discussions the requester can be either device A or device B. Similarly, the responder can be either device A or device B.

Figure 19: Implementation Example of Link Retry Block Diagram



Retry Pointer Description

The retry pointer represents the packet's position in the retry buffer. The retry pointer transmitted with the packet points to the retry buffer location + 1 (to which the tail is being written). This is the starting address of the next packet to be transmitted.

There are two retry pointer versions:

- The FRP as it travels in the forward direction on the link
- The RRP, as the same retry pointer travels back on the other side of the link, completing the round trip and eventually getting back to the retry buffer control logic in the link master

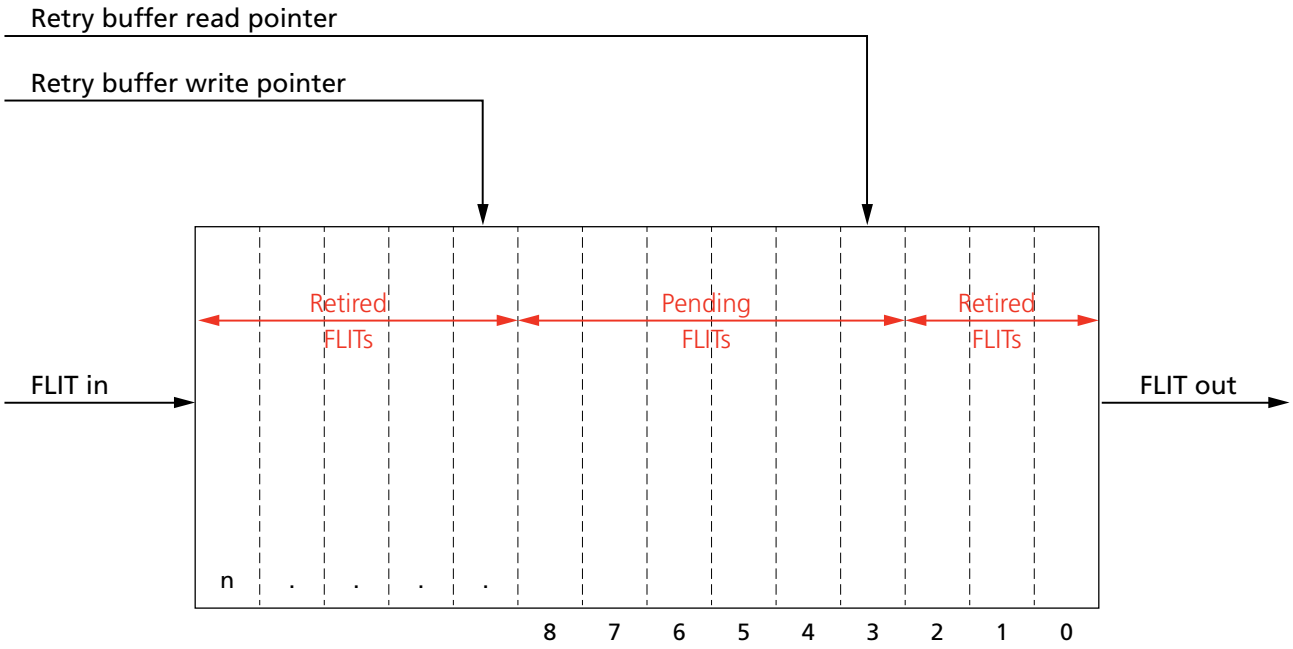
The FRP is sent with every packet that is part of the normal packet flow (other than IRTRY packets and PRET packets described later). The retry buffer is addressed using FLIT addresses, and the retry pointer represents a FLIT position in the retry buffer. Even though a packet may be multiple FLITs, the retry pointer will always point to the header FLIT position of a packet.

There are two pointer fields in the tail of a packet:

- FRP field: This is the forward retry pointer as the packet is traveling in the forward direction on the link. The FRP field is ignored in PRET packets and is redefined to contain two retry control flags in IRTRY packets.
- RRP field: This is the return retry pointer embedded in any packet traveling in the return direction of the link. If the return direction has no normal packet traffic, when an FRP is extracted in the local link slave and sent to the link master, a PRET packet is then created in the link master and the FRP is loaded into the RRP field. The RRP is not associated with the packet in which it is embedded, other than to “ride along” to get back to the other end of the link. The link is symmetrical in both directions, so RRPs are embedded in packet traffic moving in both directions. The RRP field of the packet is always valid, including IRTRY packets and PRET packets.

Link Master Retry Functions

Figure 20: Implementation Example of a Retry Buffer



In this implementation example, a retry buffer is used to save packets for potential re-transmission if a link error occurs. The retry buffer relies on a write pointer and read pointer to track the status of the FLITs stored within it. The write pointer is advanced with valid outgoing FLITs as determined by the packet stream traveling out of the link master. As FLITs are written into the retry buffer, they are considered to be pending FLITs, because they may need to be re-transmitted if an error occurs. They remain pending FLITs until the read pointer is advanced past them, indicating that one or more returned retry pointers were received, representing a successful packet transmission.

The read pointer in this example is used to read data from the retry buffer during a retry sequence. If no errors occur, retry pointers are returned (RRPs) and used to advance the read pointer. As RRPs are received, the read pointer will rotate around the retry buffer address space, following the write pointer, and turn pending FLITs into retired FLITs.

Retired FLITs are unneeded residue FLITs that will eventually be overwritten by new pending FLITs as the write pointer advances and wraps around the retry buffer address space.

During a retry sequence, the read pointer is incremented until it equals the write pointer, indicating that all pending FLITs have been pulled out of the retry buffer and retransmitted across the link. All RRPs received during this period are saved in a read pointer hold register until all pending FLITs have been read out of the retry buffer. At the end of the LinkRetry_FLIT state (see LinkRetry_FLIT State), as the retry control switches back to a flow of new packets, the latest value of the read pointer hold register is copied to the read pointer. This action may retire many of the pending FLITs. After the copy is made, any newly arrived RRPs are used to advance the read pointer directly, thus retiring more FLITs.

A threshold is implemented in this example to detect a near-full condition of the retry buffer. The near-full condition is detected by comparing the value of the write pointer to the read pointer. This threshold leaves enough room remaining in the retry buffer for the current packet in transmission to be completely saved in the retry buffer, without overflow, when the near-full detection occurs. At this point, the link master will stop the packet flow to the link.

Forward Retry Pointer Generation

The link master generates the retry pointer and includes it in the FRP field in the tail of the packet being transmitted. The retry pointer is created by using the value where the tail FLIT of the packet is to be saved for potential retransmission, incrementing by +1, and loading it into the FRP field within the tail of the packet. This value points to the location where the header of the next packet to be transmitted will be saved. The first packet written into the HMC's retry buffer (a TRET packet during transaction layer initialization) will have an FRP value of 1. Until a link master receives a forward retry pointer from the local link slave, a value of zero will be embedded into the RRP field of outgoing packets. The FRP value must be included in the CRC generation, which is also loaded into the tail of the packet being transmitted. The FRP field is valid for all packets that are saved for potential retransmission. All packets that are transmitted other than IRTRY, NULL, and PRET packets are saved until the success of their transmission has been acknowledged. If the original packet stream on the link included NULL FLITs between some of the packets, those NULL FLITs are not saved for potential retransmission. However, during a retry sequence, as the retried packet sequence is transmitted, NULL FLITs may be transmitted between packets.

Packet Sequence Number Generation

The link master generates a sequence number for every outgoing packet that is part of the normal packet flow and loads it into the SEQ field within the tail of the packet. The sequence number should be incremented by + 1 for every normal packet. The sequence number should increment by +1 for every packet that is saved for retransmission. It will not be generated for PRET or IRTRY packets, as these are not saved for retransmission. Any packets replayed during a retry sequence already have their sequence number embedded and are re-sent without updating the SEQ field. After the last retried packet is transmitted, the sequence number generator should increment for the next new packet that follows the retried packet stream. This accommodates a sequence checking across a retry sequence, verifying that all packets are sequential before, during, and after the retry sequence. The sequence number register in the link slave should be set to zero upon a cold or warm reset. The first packet sent from the link master after a cold or warm reset should contain a sequence number of 1.

Forward Retry Pointer Reception and Embedding

One link master function is to return retry pointers that represent packets traveling in the opposite direction on the other side of the link. The link master captures the FRP from the link slave and then embeds it in the RRP field in the tail of the current or next packet being transmitted. At that point it becomes the return retry pointer. Transmitted packets may consist of multiple FLITs that span multiple core cycles, so one or more new FRPs could be received from the local link slave before the previous one has been embedded. In this case, the latest FRP overwrites the previous one. The link master does not have to embed every retry pointer received from the link slave; only the latest one, which supersedes any previous pointer.

If there is no forward traffic being transmitted by the link master when a retry pointer arrives from the link slave, the link master generates a single-FLIT retry pointer return (PRET) packet where it embeds the latest value of the retry pointer into the RRP field. The PRET packet generated for this purpose is not saved in the retry buffer.

If there are no FRPs being sent from the link slave to the link master, the last valid FRP value received from the link slave will be embedded in subsequent packets sent from the link master. However, in this case, no explicit PRET packets need to be generated if the last valid FRP value has been embedded in at least one packet.

The link master must always embed the current value of the retry pointer collection register into the RRP field of every non-NULL packet transmitted, including:

- Normal transactional packets from the link output buffer
- IRTRY packets transmitted during the StartRetry stream
- IRTRY packets transmitted during the LinkRetry sequence
- Pending packets transmitted during the LinkRetry_FLIT state of the LinkRetry sequence. In this case, the old pointer values of the RRP field (previously transmitted) are overwritten by the latest pointer. The CRC must be recalculated for each packet due to the RRP change.

Return Retry Pointer Reception

The link master not only receives the FRPs for packets traveling in the opposite direction from the local link slave, as described in the previous section, but also receives RRPs from the local link slave. Receipt of an RRP completes the loop that the retry pointer travels and acts as an acknowledgment of the successful transmission of the associated packet.

Link Master Sequences

The link master provides one of two different sequences, depending on where the link error occurs, that interrupts the normal packet flow being transmitted from the link master. When only one link error occurs (the most likely case), the local link master adjacent to the link slave that detected the error will execute the StartRetry sequence. The link master transmitting in the direction where the link error occurred will execute the LinkRetry sequence.

StartRetry Sequence

When a link slave detects an error, it sets “Error Abort Mode,” which is monitored by the local link master. When this signal activates, it triggers the local link master to initiate its StartRetry sequence, which interrupts the normal transaction packet flow and transmits a programmable number of single-FLIT IRTRY packets. A packet currently being transmitted must be allowed to complete before starting the IRTRY stream. Within each of these IRTRY packets, the StartRetry flag is set with FRP[0] = 1. This stream of IRTRY packets must be contiguous, meaning each IRTRY FLIT must be followed by the next IRTRY FLIT without the insertion of any other types of FLITs/packets, including NULL FLITs, for the duration of the stream. The transmission of the IRTRY stream follows these rules:

- The link master should always embed the latest FRP it received into the RRP field of each IRTRY packet as it does for any outgoing packet.
- None of the IRTRY packets should be loaded into the Retry Buffer.

The execution of the StartRetry sequence also starts a Retry Timer. This is used to monitor the success of the link retry. If the link retry is unsuccessful and Error Abort Mode does not clear, the retry timer will time-out which will initiate the execution of another StartRetry sequence. This will continue until the number of retry attempts equals the retry attempt limit specified in the link retry control register.

StartRetry Sequence States

StartRetry_Begin State

This state enables the completion of an in-flight packet transmission, after which the normal packet flow stops. The in-flight packet must also be saved for retransmission.

StartRetry_Begin State Control
- Enter: When the link master detects the rising edge of the Error Abort Mode signal from the local link slave, OR when the retry timer expires and Error Abort Mode is set.
- Actions:
  1. Allow any packet currently being transmitted to complete.
  2. Stop normal packet flow and enter StartRetry_Init state. This will force the packet stream to be queued-up in the link output FIFO.
- Exit: When the in-flight packet transmission is complete.

StartRetry_Init State

In this state, the link master transmits a programmable number of single-FLIT IRTRY packets across the link. These packets are not saved for retransmission, thus their forward retry pointer and sequence number aren't used. Within each IRTRY packet, Bit 0 of the FRP field is redefined to hold the StartRetry flag, which signals the other end of the link to initiate its LinkRetry sequence. The link master also embeds the last FRP value it received in the RRP field of every IRTRY packet. This stream of IRTRY packets accomplishes the following:

- It signals to the link slave on the far end of the link that this end detected a link error.
- It provides the latest value of the retry pointer, embedded in the RRP field, to the other end of the link.

StartRetry_Init State Control

- Set: When StartRetry_Begin state clears
- Action: Start transmission of contiguous stream of IRTRY packets that include the StartRetry flag (FRP[0]=1). The number of packets is determined by the IRTRY packet transmit number field in the Link Retry Control Register. NULL FLITs must not be inserted between IRTRY packets.
- Clear: After the programmable number of IRTRY packets has been transmitted.

When the link master exits the StartRetry_Init state, it reverts back to its normal packet stream that has been accumulating in its link output buffer.

Retry Timer

The Retry Timer monitors the success of the current retry attempt by timing the assertion of the Error Abort mode signal from the local link slave.

The vast majority of link retries will be successful on the first retry attempt, in which case the Error Abort mode will clear out within the normal retry loop time, and the Retry Timer will not affect the delay of the overall retry operation. Given that the retry loop

time is less than or equal to the retry buffer full period, the retry timer should be set to at least 3 times the retry buffer full period, as shown in the right most column of the Retry Pointer Loop Time table. This is to eliminate the possibility of issuing a second StartRetry before the first retry has a chance of completing during a multiple error case where both sides of the link may be executing a link retry sequence, before the first Init_Retry/StartRetry stream is allowed to be transmitted.

Retry Timer Algorithm

If Error_Abort_Mode = 0

- Retry_Timer = 0
- Retry_attempts = 0
- Retry_Timer_expired = 0

Elseif Retry_Timer = Retry Timeout Period AND retry_attempts < retry_attempt_limit

- Retry_Timer_expired = 1
- Retry_attempts = Retry_attempts + 1
- Retry_Timer = 0

Elseif retry_attempts = retry_attempt_limit

- Set link_retry_failed

Else

- Retry_Timer = Retry_Timer + 1
- Retry_Timer_expired = 0

EndIF

LinkRetry Sequence

If the link master receives a StartRetry pulse from the local link slave, it is an indication that the link slave on the other end of the link detected an error, and the link master should initiate its LinkRetry sequence. This sequence will suspend the flow of packets from the link output buffer and send the packets currently being saved for retransmission.

LinkRetry Sequence States

The LinkRetry sequence includes three states: LinkRetry_Begin, LinkRetry_Init, and LinkRetry_FLIT. The LinkRetry_FLIT state may or may not be entered depending on if there are any packets to be retransmitted.

LinkRetry_Begin State

This state enables the completion of an in-flight packet transmission, after which the normal packet flow stops. The in-flight packet must also be saved for retransmission.

LinkRetry_Begin State Control

- Enter: When the link master receives a StartRetry pulse from the local link slave.
- Actions:
  1. Enable the packet currently in transmission to complete.
  2. Stop normal packet flow and enter next retry state.
- Exit: When the in-flight packet transmission is complete.

LinkRetry_Init State

In this state, the link master transmits a programmable number of single-FLIT IRTRY packets across the link. Within each of these IRTRY packets, the ClearErrorAbort flag will be set (FRP[1]=1). These packets are not saved for retransmission, thus their Forward Retry Pointer and SEQ number are not used. The link master also embeds the last valid FRP it received from its local link slave in the RRP field of every IRTRY packet. The stream of IRTRY packets enables the following:

- The ClearErrorAbort flag enables the link slave section on the far end of the link to detect the link master is in the retry sequence.
- It provides a period of time for the link slave to detect a number of good IRTRY packets that establish confidence in the link.

LinkRetry_FLIT State

In this state, the link master transmits the packets currently being saved for retransmission. These packets have already been transmitted but have been aborted at the receive end of the link. If an error occurs in a packet that this link master had previously transmitted, this state will be entered. If an error occurs only during the transmission of a stream of continuous NULL FLITs and there is not any FLITs currently saved for retransmission, this state will not be entered.

If the link master is retransmitting packets in the LinkRetry_FLIT state and it detects the assertion of the Error Abort mode signal, it can choose to either:

1. Finish the LinkRetry_FLIT state and allow all packets to be retransmitted before sending a stream of IRTRY packets with the StartRetry flag set.
2. Pause the LinkRetry_FLIT state and send a stream of IRTRY packets with the StartRetry flag set, and then revert back to the completion of the LinkRetry_FLIT state.

The HMC implementation completes the LinkRetry_FLIT state to empty the retry buffer, and then sends the stream of IRTRY packets.

LinkRetry_FLIT State Control

- Enter: When LinkRetry_Init state clears AND there are packets currently saved for retransmission.
- Actions: Retransmit all saved packets across the link. All packets retransmitted must have the latest valid FRP value received from the local link slave embedded into their RRP field.
- Exit: When all packets saved for retransmission have been sent to the link.

LinkRetry Sequence Exit

The link master exits the retry sequence from either the LinkRetry_Init state or the LinkRetry_FLIT state depending on whether there were packets saved for retransmission when the LinkRetry_Init state was exited. The normal packet stream may now resume.

Link Slave Retry Functions

The link slave section parses incoming packets, performs the CRC and sequence check for each packet, and if no error forwards the packets to the internal destination. The link slave also extracts the FRP and RRP fields from each good packet and forwards those to the local (adjacent) link master section. When a link error occurs, the link slave goes into error abort mode, which stops the forwarding of the incoming packets. It also

stops the forwarding of FRP and RRP pointers to the local link master. The assertion of error abort mode causes the local link master to transmit a stream of IRTRY packets to signal to the other end to activate its link retry sequence. When a link retry is in progress, the link slave detects and counts two different types of IRTRY packets, including those with the StartRetry flag set (FRP[0]=1) and those with the ClearErrorAbort flag set (FRP[1]=1).

Packet CRC/Sequence Check

As incoming packets flow through the link slave CRC is calculated from the header to the tail (inserting 0s into the CRC field) of every packet. As the tail passes through, the calculated CRC is compared to the incoming CRC included in the tail of the packet. A mismatch indicates that bits in the packet have changed since the CRC was originally generated at the link master. If the calculated CRC compares correctly, an additional packet check is provided by extracting the packet sequence number out of the tail and comparing it with the saved value of the sequence number from the last successful packet. The sequence number should always be incremented by + 1 for a correct transmission of packets across the link.

FRP and RRP Extraction

If the incoming packet passes the CRC and sequence checks, and the link slave is not in error abort mode, the link slave extracts the values of the FRP and RRP fields and forwards them to the local link master section.

Error Abort Mode

If an LNG, CRC, or sequence error occurs on an incoming packet, the link slave executes the following actions:

- The link slave sets the Error Abort mode.
- The link slave drops or poisons the packet that had the CRC or sequence error. Poisoning involves inverting the calculated CRC for the packet, so it will be dropped by a downstream receiver of the packet. This action is required because in the case of packets that may span multiple core cycles, the beginning of the packet may have been forwarded before the CRC check was performed at the tail, to minimize latency.
- During error abort mode, the link slave will drop (not forward) all subsequent packets following the packet that caused the error.
- During error abort mode the link slave normally does not extract values from the RRP, FRP, or RTC fields. There is one exception to this, which is described in the IRTRY Packet Operation section.
- Error Abort mode is cleared by monitoring the IRTRY packets as described in the IRTRY Packet Operation section.

IRTRY Packet Operation

When the link slave receives an IRTRY packet, it performs a LNG check and CRC check similar to checking any other incoming packet. The SEQ check is inhibited for IRTRY packets because they are not retried if they have a failure. A stream of multiple IRTRY packets up to a programmable threshold must be received before an action is performed so that a data payload that looks like an IRTRY packet arriving after a link error occurred doesn't falsely trigger a retry action.

Each successfully received IRTRY packet will cause the link slave to increment one of two counters as described below. This occurs even if Error Abort mode is set. No IRTRY

packets are forwarded beyond the link slave. If an IRTRY packet has a LNG error or CRC error, the link slave will set error abort mode if it is not already set.

The link slave decodes two types of IRTRY packets:

1. StartRetry: IRTRY with FRP[0] = 1, indicating the StartRetry flag. In this case the link slave will increment the StartRetry counter for each StartRetry flag that it receives in a continuous stream of IRTRY packets. Once the counter reaches the threshold specified in the Init_Retry Packet Receiver Number field of the link retry control register, the link slave will generate a StartRetry pulse that is sent to the local link master. Additional StartRetry flags may arrive after the threshold is met if the transmitting link master has its Init_Retry Packet Transmit Number programmed to be greater than the threshold. These excess StartRetry flags should not affect the operation and should not cause a second StartRetry pulse. If a FLIT or packet other than an IRTRY is received, the link slave will clear the StartRetry counter, which should arm the counter to begin counting again if additional IRTRY packets with the StartRetry flag set arrive later. If the link slave detects a link error and sets Error Abort mode, it will also clear the StartRetry counter. When receiving IRTRY packets with the StartRetry flag, there are two cases of RRP extraction:
    a. If error abort mode is not set, the link slave extracts an RRP out of every good packet that it receives, including all IRTRY packets with the StartRetry flag set. These RRPs are sent to the local link master.
    b. When error abort mode is set, normally the link slave does not extract any retry pointers or tokens from incoming packets. There is one exception: If Error Abort mode is set and the link slave is receiving IRTRY packets with the StartRetry flag, this is a special case where a link error occurred on both sides of the link. The link slave will count the StartRetry flags, and when reaching the threshold it will extract the RRP from the IRTRY packet that caused the threshold to be reached. The link slave will send this RRP along with the StartRetry pulse to the local link master.
2. ClearErrorAbort : IRTRY with FRP[1] = 1, indicating the ClearErrorAbort flag. In this case the link slave will increment the ClearErrorAbort counter for each ClearErrorAbort flag that it receives in a continuous stream of IRTRY packets. Once the counter reaches the threshold specified in the Init_Retry Packet Receiver Number field of the link retry control register, the link slave will clear error abort mode if it is currently set. It will also extract the RRP out of the IRTRY packet that reached the receiver number threshold and send the RRP to the local link master. Additional IRTRY packets with the ClearErrorAbort flag set may arrive after the threshold is met if the transmitting link master has its Init_Retry Packet Transmit Number programmed to be greater than the threshold. Specific logical implementation may require more IRTRY packets with the ClearErrorAbort flag set to be transmitted than the link slave is expecting. These excess ClearErrorAbort flags will not affect the operation as long as Error Abort mode clears before a retried packet is received. The link slave should follow normal procedure and extract all RRPs after error abort mode clears and send them to the local link master. If a FLIT or packet other than an IRTRY is received, the link slave will clear the ClearErrorAbort counter, which should arm the counter to begin counting again if additional IRTRY packets with the ClearErrorAbort flag set arrive later. If the link slave detects a link error on any IRTRY packet with ClearErrorAbort flag set when error abort mode is clear, it should set error abort mode and also clear the ClearErrorAbort counter.

Resumption of Normal Packet Stream after the Retry Sequence

At some point after the retry sequence, a normal packet (either a retried one or a new one) will be received by the link slave. The link slave operates normally on that packet, verifying the LNG, CRC, and sequence number fields and forwarding it on to its internal destination. Because the link slave captured a valid sequence number from the last good packet received before the retry sequence (no sequence numbers were extracted from the IRTRY packets), it will compare that sequence number with the first one received after the retry sequence. The sequence number should be incremented by 1 if the retry sequence was successful and the normal packet stream resumed correctly.

Retry Pointer Loop Time

The HMC protocol includes retry pointer space to support a retry buffer that can hold up to 576 FLITs. In order to maintain link performance, the maximum allowable retry pointer loop time must be less than the retry buffer full period shown in the table. The maximum retry pointer loop time could be allowed to be greater than the time it takes to fill the retry buffer, but the retry buffer full condition will then throttle the packet stream.

The result is that NULL FLITs will be sent between transaction packets, degrading the performance of the link. To avoid link throttling, the host's portion of the retry pointer loop delay must be less than the "Host Allowable Delay" shown in the table.

The following is an implementation example that highlights the steps involved to identify the total retry pointer loop time:

- The time it takes for a packet (and its FRP) to travel from a link master to a link slave, including:
  - Serialization time at the transmitting end of the link
  - Deserialization time at the receiving end of the link
- The time for the longest packet's tail to be received in the link slave, to the point of CRC check and FRP extraction
- The transfer of the extracted FRP from the link slave to the local link master
- Time to wait in the link master for just missing embedding the retry pointer into the RRP field of the previous packet and waiting for the next tail of the longest packet
- The time it takes for a packet with the embedded retry pointer to travel back to the source end of the link, including:
  - Serialization time at the transmitting end of the link
  - Deserialization time at the receiving end of the link
- Time for the longest packet's tail to be received in the link slave, to the point of CRC check and RRP extraction
- The transfer of the RRP from the link slave to the local link master (the original transmitter in the first step)

Link Flow Control During Retry

When the normal packet stream is transmitted by the link master, it has already determined that the link slave has enough buffer space in its buffers to accept all FLITs of the transmitted packets, as controlled by the return token count protocol. If a link error occurs and the packet stream is aborted during error abort mode in the link slave, the reserved space in the receive buffers still exists, and therefore, when those packets are re-

transmitted across the link, there is no need for any additional flow control, except for the following clarification. At the receiver, the link slave may poison the packet that has errored, but it has already propagated it forward into the receive buffer. Because this takes up some FLIT locations, the return token count control logic must not return the tokens for a poisoned packet. That packet will be retransmitted, and when it is finally loaded into the receive buffer with good CRC and propagated further, the tokens will be returned. All subsequent packets (after the errored one) dropped in the link slave will not be forwarded into the receive buffer. As a result, the buffer space will be available for those packets as they are retransmitted.

A consequence of forwarding the poisoned packet and not returning tokens until the packet is retransmitted successfully is as follows: The poisoned packet could have been the last packet that caused the full condition at the receive buffer, and then the receive buffer could be temporarily stalled due to downstream flow control. If this were the case, the failed packet would be the only packet to be transmitted. When the packet was retransmitted, there would be no room for that packet in the receive buffer. An easy solution to this is to adjust the maximum tokens available for that receive buffer to 17 FLITs less than the actual buffer size. This accommodates an extra copy (the poisoned one) of the largest packet that could be retransmitted during a link retry.

System Error Handling – Host Error Management

HMC-30G-VSR has several automatic methods of finding and correcting errors. This section defines the host's actions based on the polling status of the HMC device, either by measuring bandwidth to determine operation degradation or by receiving errors from the HMC device. The table below outlines a few examples of errors and actions the host may take.

Table 105: Examples of Host Error Management

	Symptom (What the host may experience)	Action for the Host to Take	Operational Impact
Faulty TX/RX lane	Higher than normal BER; High number of retries necessary	Retrain the link	Quiesce traffic on link, wait for the buffers to clear. No impact on other link traffic (even to the affected local quadrant).
	Link retry fails; Dropped bits in scrambler; Fatal error	Warm reset per quadrant	Can only be implemented if all link traffic only accesses local quadrants to each link.
	Link retry fails; Dropped bits in scrambler; Fatal error	Warm reset for entire HMC device (more common)	Must be implemented when links access affected local quadrant of HMC.
	Consistent link retry fails (identified to specific lane in link); Dropped bits in scrambler	Re-initialize link for entire HMC device to reduce link width configuration	Host must throttle bandwidth to accommodate the new link-width configuration.

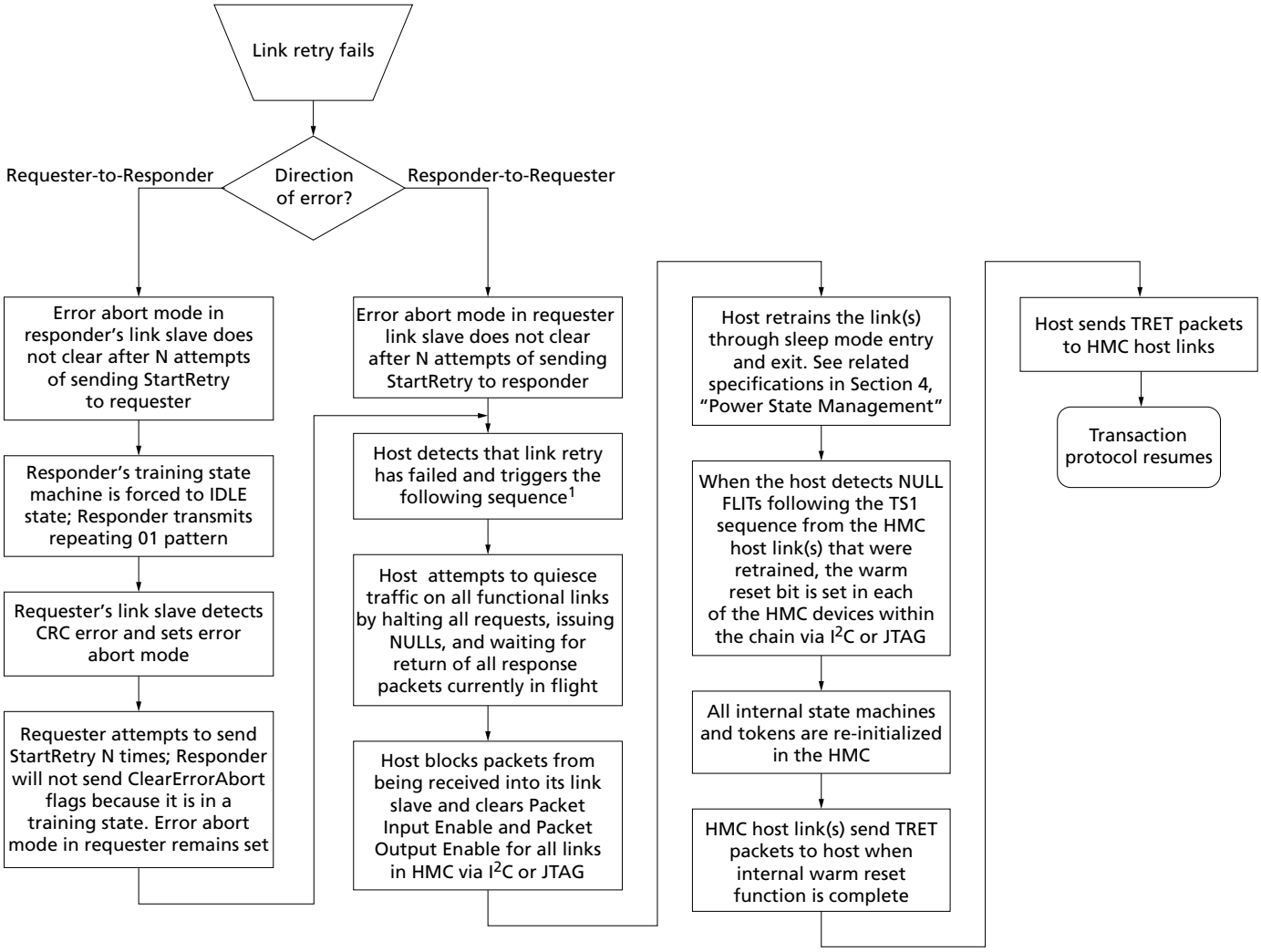
Retraining

Retraining a Link with High Error Rate

The host can monitor the frequency of link retry occurrences in the ERRSTAT field of response packets and initiate a retraining procedure when it determines the error rate on any downstream link is above an acceptable threshold. The steps for this procedure are as follows:

1. Host detects an unacceptable rate of link retries on any downstream link.
2. Host quiesces traffic (ensures closure on all transactions in flight).
3. Host blocks packets from being received into its link slave.
4. Host retrains links through sleep mode entry and exit. All specifications described in “Power State Management” are applicable.

Figure 21: Host Recovery after Link Retry Fails



Note: 1. Host detection of link retry failure on link not directly connected to host will occur via an Error Response packet from a downstream HMC.

Warm Reset

Warm reset provides a mechanism to re-initialize the HMC state machines and tokens to allow on-line activity again after a system or HMC fatal error has occurred.

The occurrence of an HMC fatal error is communicated to the host through an Error Response packet (if possible) and by the assertion of the FERR_N signal.

Warm reset is a run-time operation and should not be performed prior to Init Continue completion. Before a warm reset is issued, all configured links should be in the active state.

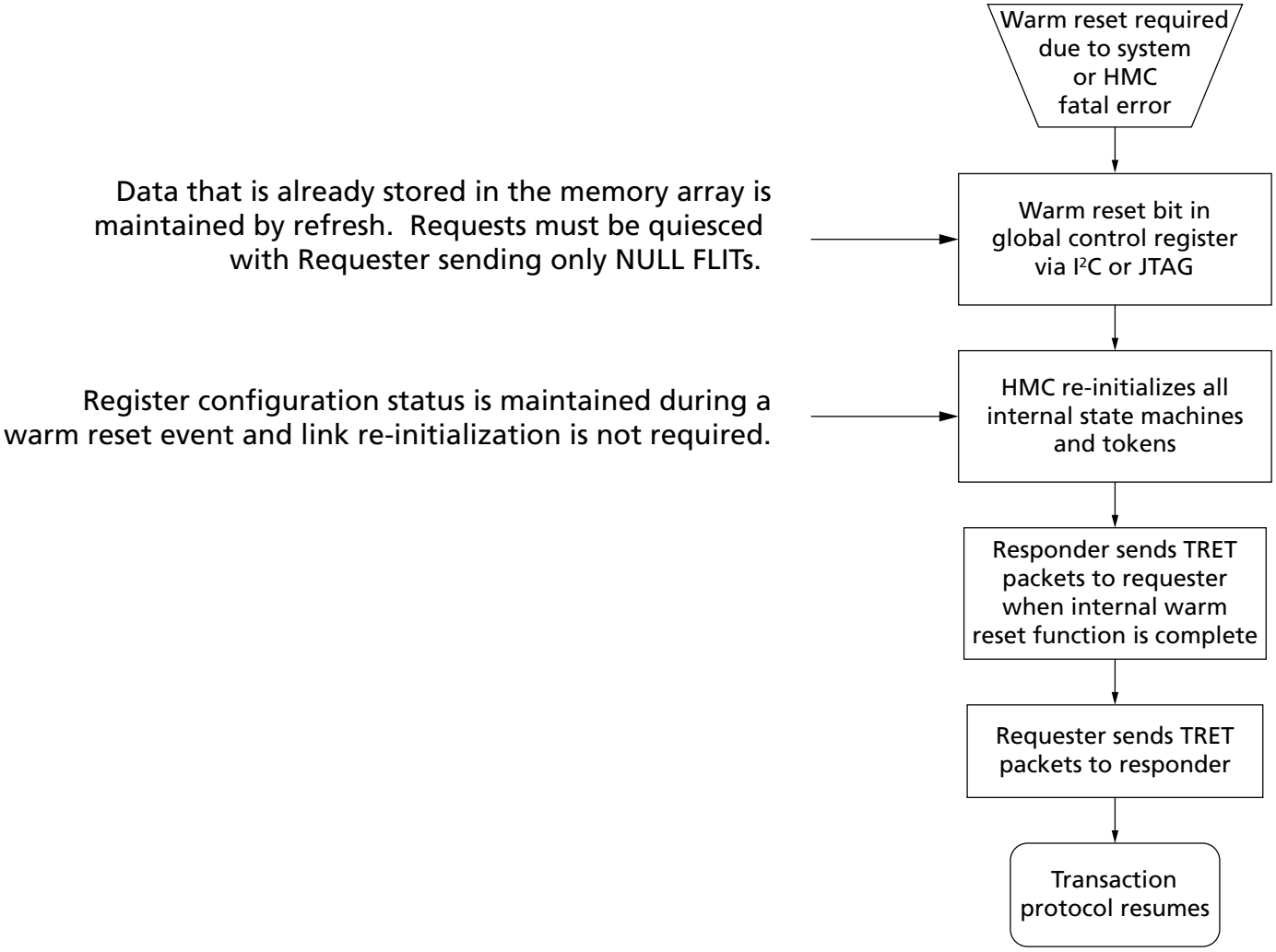
HMC supports a warm reset per quadrant/link. With specific host topologies that guarantee no inter-quadrant traffic, it is possible to perform a warm reset to one or more quadrants without affecting other quadrants. This warm reset method provides recovery for a limited number of errors local to the quadrant, such as a Link Retry X Failed error, while allowing other quadrants to continue running online.

In multilink configurations that require communication across quadrants, a warm reset must be across all links. This prevents packet flow errors being managed in the affected quadrant during cross-quadrant traffic. This is the more common implementation.

Upon completion of the warm reset, transaction layer initialization will occur as described in step 5 g of the initialization steps in the Power-On and Initialization section, which includes the transmission of initial tokens across all configured links. Note that while an in-band MODE WRITE command can be issued to initiate a warm reset, it is not recommended as the HMC may not respond appropriately when in such a state that a warm reset is required.

Upon issuing the warm reset, the host must wait a minimum of 200ms before issuing another configuration command via the I^2C or JTAG sideband interface.

Figure 22: Warm Reset Flow Chart



Functional Characteristics

Packet Ordering and Data Consistency

There can be multiple packet-reordering points within the HMC, including the following implementation examples:

1. The output of each link slave input buffer could reorder packets arriving from one link that are destined for different vaults. This would prevent a busy vault from blocking the packet flow out of the receive buffer and enable the flow of packets across the link to continue.
2. Each vault controller command queue could reorder packet execution to banks that are not busy, potentially optimizing the memory bus usage.

All reordering points within an HMC must maintain the order of a stream of request packets from a specific link to a specific bank within a vault. This ensures that if a host

sends a write request packet to address x followed by a read request packet to the same address x, the READ will always return the newly written data.

Packet ordering performed by the HMC supports messaging or data-passing programming between two or more hosts with access to the same HMC. This common method of host-to-host messaging is known as producer-consumer programming, and proceeds as follows:

1. Host 0 wants to pass a message/data to host 1. Host 0 writes memory location A (this could be multiple locations) with the message/data.
2. Upon receiving all expected successful write responses for the write(s) to location A, host 0 sets a flag in memory location B indicating that the message/data is available in location A.
3. Host 1 executes a loop, polling (that is, issuing read requests to) location B to observe the set flag.
4. Upon detecting the set flag in the read response for location B, host 1 issues a read request to location A.
5. Host 1 receives the read response for location A containing the newly-written data from host 0.

This scenario proceeds as defined if host 0 waits for the successful write response for the write to location A before issuing the write request packet to location B. The write response is generated at the vault controller after the write request is fully executed and is identified with the same tag as the write request. Locations A and B can be any memory locations. This scenario is not supported with the use of posted write requests.

Data Access Performance Considerations

Due to the internal 32-byte granularity of the DRAM data bus within each vault in the HMC, inefficient utilization of this bus occurs when starting or ending on a 16-byte boundary rather than a 32-byte boundary. An extreme example of this would be the host issuing a series of 16-byte read requests. Each read request would fetch 32 bytes from the DRAM and return half of the data in the response packet, throwing away the other 16 bytes of data. For bandwidth optimization it is advantageous to issue requests with 32-byte granularity.

If the host issues a series of commands that access portions of the same data block (as defined by the maximum block size) rather than issuing one request accessing the entire data block, two performance disadvantages exist:

1. The probability of bank conflicts will increase in the accessed vault.
2. The multiple request and response packets have additional header/tail overhead on the link.

Because of this, it is desirable to perform commands with larger data sizes. In an example implementation, if a host has defined the maximum block size as 256-byte and instead of issuing a 128-byte request to a specific location, it issues four consecutive 32-byte requests to the same block location. The cube will send these four independent requests to the specific bank whereby it must open and close the row in the same bank four times, reducing bandwidth utilization and increasing latency as compared to the case in which a single 128-byte request is issued.

To maximize bandwidth utilization and reduce system latency the following should be taken into consideration:

1. Request data block sizes that are as large as possible, up to the configured maximum block size, that meet the requirements of the system.
2. Configure the maximum block size based upon the most frequently requested data size. If the most frequently requested data size does not equal any of the possible maximum block sizes (32B, 64B, 128B, or 256B), select the smallest block size that encompasses the most frequently requested data size. For example, if a system most often requires 48B data transfers, 64B should be chosen as the maximum block size.

Vault ECC and Reference Error Detection

Memory READ and WRITE operations have a parity bit added to the command and address signals that are sent from each vault controller to the memory partitions and banks associated with each controller. If the bank receiving a request determines that there is a parity error, the reference is retried.

Data is protected in memory using ECC. The ECC should provide correction of a single-bit error as well as all data bits that are transmitted over a single, failed TSV between the memory and the logic base.

Memory locations are initialized with correct ECC data patterns before the host begins using the locations. This eliminates a write-before-read requirement to avoid SBEs and MUEs.

Refresh

DRAM refresh is handled internally within the HMC by the vault controllers. As an example implementation, every vault controller could have a unique refresh rate for each of its bank groups (where a bank group is a set of banks within one memory die). An implementation may also choose to vary refresh rates dynamically, driven by conditions such as temperature and/or detected memory error rates.

Scrubbing

Scrubbing is used inside the cube to mitigate soft failures in memory. Two categories of scrubbing are provided in each vault controller:

- **Demand scrubbing:** If a correctable error is detected in the read data of a read request, the data is corrected and returned to the host in the response packet. A scrubbing cycle is invoked that writes the corrected data back into the memory array.
- **Patrol scrubbing:** This form of scrubbing is performed by the cube refresh logic. If a correctable error is detected, a scrubbing cycle is invoked that writes the correct data back into the memory array.

The scrubbing cycle writes the corrected data immediately after the correctable error is discovered on the read. This READ-UPDATE-WRITE operation is atomic, meaning that no other reference to the location will be possible between the read and the associated write. This is handled as a bank conflict within the vault controller command queue.

All scrubbing cycles can be programmed to execute a reread of the scrubbed memory location to determine whether the scrubbing operation was successful. If it is successful, a soft error is logged. If it is unsuccessful and another correctable error occurs during the reread, a hard error is assumed and it is logged. Correctable errors that are designated as hard errors enable the system to continue to run, but they also increase the probability that the occurrence of another soft or hard error within the 16-byte ECC

word will turn the correctable situation into an uncorrectable one. To cover this case, the HMC provides in-field repair of the hard failures.

Response Open Loop Mode

During normal transaction layer packet flow, return token counts (RTCs) are returned in both directions on each link. This controls the flow of request packets in one direction and response packets in the other direction. Response open loop mode can be configured on the upstream links that are connected to the host. This mode eliminates the requirement for the host to send tokens back to the cube, and results in two simplifications:

- The host link slave does not have to generate tokens when response packets are consumed.
- The host link master does not have to embed any RTCs into request packets being transmitted to the HMC device.

The response open loop mode will force the link master to send response packets to the host without the token requirements. (Other packet flow controls such as retry buffer full conditions and retry in progress will still suspend packet flow if necessary.)

JTAG Interface

HMC incorporates a standard test access port (TAP) controller that operates in accordance with IEEE Standard 1149.1-2001. The input and output signals of the test access port use V_{DD} as a supply.

The JTAG test access port uses the TAP controller on the HMC, from which the instruction, bypass, ID, boundary scan, and CADATA registers can be selected. Each of these functions of the TAP controller is described in detail below.

The HMC includes boundary scan support that complies with the IEEE 1149.1 and 1149.6 standards. Boundary scan chain order is provided within BSDL files for each HMC configuration.

Disabling the JTAG Interface

It is possible to operate HMC without using the JTAG interface. To disable the TAP controller, tie the test reset signal, TRST_N, LOW. TCK, TDI, and TMS can all be considered "Don't Care" when TRST_N is LOW. TDO should be left unconnected. Upon power-up, the TAP controller will come up in a reset state; this will not interfere with the operation of the HMC device.

For more information on JTAG operation, see vendor-specific documentation.

Table 106: Instruction Codes

Instruction	Code	Active Register	Description
SAMPLE/PRELOAD	0x01	Boundary Scan	1149.1 SAMPLE/PRELOAD instruction
IDCODE	0x02	Device ID	1149.1 IDCODE instruction
CLAMP	0x04	Boundary Scan	1149.1 CLAMP instruction
HIGHZ	0x08	Boundary Scan	1149.1 HIGHZ instruction
EXTEST	0x09	Boundary Scan	1149.1 EXTEST instruction

Table 106: Instruction Codes (Continued)

Instruction	Code	Active Register	Description
EXTEST_PULSE	0x0A	Boundary Scan	1149.6 EXTEST_PULSE instruction
EXTEST_TRAIN	0x0B	Boundary Scan	1149.6 EXTEST_TRAIN instruction
CFG_RDA	0x2C	CADATA	Allows Configuration & Status Register to be read
CFG_WRA	0x2D	CADATA	Allows Configuration & Status Register to be written
BYPASS	0xFF	BYPASS	1149.1 BYPASS instruction

I²C Interface

The I²C bus is provided as a sideband interface to program, monitor and access various modes within the HMC. The I²C bus complies with the UM-10204 I²C bus specification and includes the following attributes:

1. Operates with $_{DDK}$ supply (1.8V nominal) with open-drain output stage
2. 7-bit slave addressing
3. Operates at 100 Kb/s (standard-mode), 400 Kb/s (fast-mode), and 1 Mb/s (fast-mode plus)
4. FTDI I²C compatibility
5. Complies with the standard-mode, fast-mode, and fast-mode plus electrical and timing parameters found in the UM-10204 I²C bus specification. The V_{DD} referred to in these tables is equivalent to the HMC 1.8V V_{DDK} supply.
6. Supports the Device ID READ command.
7. Supports byte-level clock stretching on configuration READ operations.

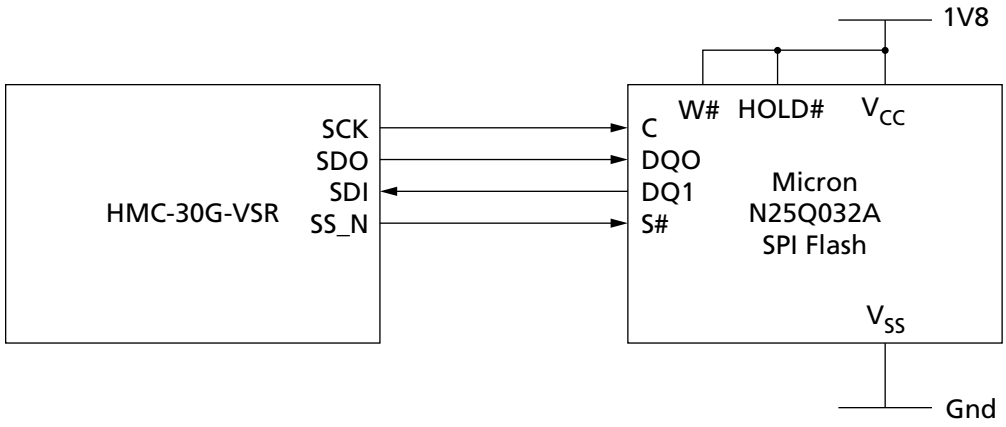
For more detailed information on I²C use, see vendor-specific documentation.

Serial Peripheral Interface

The serial peripheral interface (SPI) allows an external SPI Flash device to be fitted in accordance with the application circuit shown in the following figure.

The SPI Flash device is managed wholly by the HMC. When fitted, the SPI Flash device extends the nonvolatile storage capability of the HMC. It provides additional capability and flexibility for system debug. It is strongly recommended that customers make provision in their designs to fit this device. It can be depopulated in production systems.

Figure 23: SPI Application Circuit



HMC-30G-VSR Electrical Specifications

The HMC-30G-VSR specification is designed for a chip-to-chip link, NOT a chip-to-module link. HMC-30G-VSR will support a channel with insertion loss of ~10dB at Nyquist and will enable low-power SerDes. As such, it is different from HMC-15G-SR in terms of reach (which will support a channel with insertion loss of ~15dB at Nyquist).

Absolute Maximum Ratings

Stresses greater than those listed may cause permanent damage to the device. This is a stress rating only, and functional operation of the device at these or any other conditions outside those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may adversely affect reliability.

Supply Current

Supply current is the current drawn from the respective power rails used for power budgeting. Because these currents vary greatly with the resources used, refer to vendor-specific documentation for supply current estimates of your design.

DC Operating Conditions

Table 107: DC Electrical Characteristics

Specifications within this table represent limits that must be met at the HMC package balls

Description	Parameter	Minimum (V)	Nominal (V)	Maximum (V)	Notes
Logic core supply	V_{DD}	0.873	0.9	0.927	1
Phy core supply	AV_{DD}	0.873	0.9	0.927	1, 2
DRAM supply	V_{DDM}	1.067	1.1	1.133	
DRAM wordline boost supply	V_{PP}	1.71	1.8	1.89	1, 2
GPIO, side-band control supply	V_{DDK}	1.71	1.8	1.89	
Analog Phy ground	AV_{SS}	–	0	–	
Ground	V_{SS}	–	0	–	

Notes:
1. See the vendor-specific HMC Design Users Guide for voltage supply regulation sharing restrictions.
2. Supplies may require filtering, refer to the vendor-specific HMC Design Users Guide for filtering information.

HMC-30G-VSR Side-Band Electrical Specifications

JTAG Electrical Parameters

Table 108: JTAG Voltage and Timing Parameters

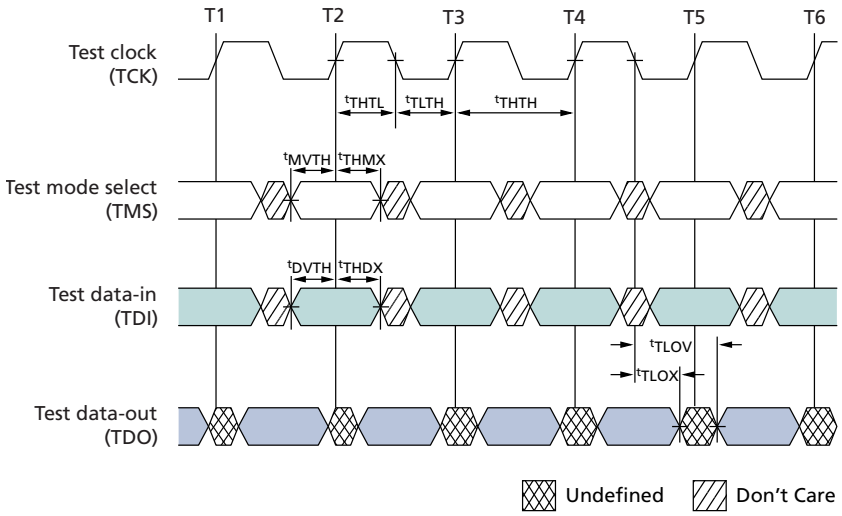
Parameter Description	Symbol	Min	Max	Units	Notes
JTAG Voltage Parameters					
Input LOW voltage - DC	V_{IL_DC}	V_{SS} - 0.4V	V_{SS} + 0.5V	V	1
Input HIGH voltage - DC	V_{IH_DC}	V_{DDK} - 0.5V	V_{DDK} + 0.2V	V	2
Output LOW voltage	V_{OL}	–	V_{SS} + 0.1V	V	3
Output HIGH voltage	V_{OH}	V_{DDK} - 0.1V	–	V	4
JTAG Timing Parameters					
Clock cycle time	^{t}THTH	10	–	ns	5

Table 108: JTAG Voltage and Timing Parameters (Continued)

Parameter Description	Symbol	Min	Max	Units	Notes
Clock frequency	^{t}TF	–	100	MHz	5
Clock LOW time	^{t}TLTH	5	–	ns	
Clock HIGH time	^{t}THTL	5	–	ns	
TCK LOW to TDO unknown	^{t}TLOX	0	–	ns	
TCK LOW to TDO valid	^{t}TLOV	5	–	ns	
TDI valid to TCK HIGH	^{t}DVTH	2.5	–	ns	
TCK HIGH to TDI invalid	^{t}THDX	2.5	–	ns	
TMS capture setup	^{t}MVTH	2.5	–	ns	
TMS capture hold	^{t}THMX	2.5	–	ns	

Notes:
1. V_{SS} measured at HMC package ball.
2. V_{DDK} measured at HMC package ball.
3. I_{OL} = 1mA.
4. I_{OH} = 1mA.
5. Minimum clock cycle time (maximum clock frequency) for boundary scan register is 50ns (20 MHz) when boundary scan register is active.

Figure 24: TAP Timing



I²C Electrical Parameters

Table 109: I²C Voltage Parameters

Parameter	Symbol	Conditions	Min	Max	Unit
Input low voltage	V_{IL}		–	0.3 V_{DDK}	V
Input high voltage	V_{IH}		0.7 V_{DDK}	–	V
Output low voltage	V_{OL}	Open-drain at 2mA sink current	–	0.2 V_{DDK}	V

Table 109: I²C Voltage Parameters (Continued)

Parameter	Symbol	Conditions	Min	Max	Unit
Output high voltage	V_{OH}	V_{EXT} (external bus supply)	–	V_{EXT}	V

SPI Electrical Parameters

Table 110: SPI Voltage Parameters

Parameter	Symbol	Conditions	Min	Max	Unit
Input low voltage	V_{IL}		–	0.3 V_{DDK}	V
Input high voltage	V_{IH}		0.7 V_{DDK}	–	V
Output low voltage	V_{OL}		V_{SS} - 0.2	V_{SS} + 0.2	V
Output high voltage	V_{OH}		V_{DDK} - 0.2	V_{DDK} + 0.2	V

HMC-30G-VSR Physical Link Specifications

Physical Link Electrical Interface

The HMC transmitter and receiver circuits are designed to allow for flexible integration with the host. Various supported configurations are shown in Termination Configurations below. Both the HMC RX and TX circuits are meant to be connected to their respective host lanes through the use of 100Ω differential line impedance. The VSR PHY supports channels consistent with existing standards (OIF CEI-28-VSR).

Termination Configurations

The impedance values shown are nominal. Host transmitter and receiver termination and signaling parameters may differ from those of the HMC as long as the HMC receiver signaling parameters are met and the HMC transmitter through the channel meets the host receiver signaling parameters.

The HMC receivers are internally AC-coupled, thereby removing the need for external AC coupling capacitors in the channel. However, the HMC receiver can support channels that have external AC coupling present. The HMC has been designed to minimize the probability of long DC run lengths with its scrambling circuitry. The host transmitter must meet the requirements in the Lane Run Length Limitation section. Additional run length limit circuitry to avoid run lengths of longer than 100UI at the HMC transmitter output is also available (See Lane Run Length Limitation). Specific host AC coupled receiver encoding requirements beyond this, such as 64b/66b encoding or 8b/10b encoding, are not supported.

HMC-30G-VSR supports both AC and DC coupling from HMC TX to host RX connections. The specific coupling option will depend on the host RX design. HMC-30G-VSR supports both AC and DC coupling for host TX to HMC RX connections. DC coupling is recommended as the on-die AC coupling on the HMC-30G-VSR receiver is always enabled. If AC coupling is preferred, a 0.1µF AC coupling cap is recommended.

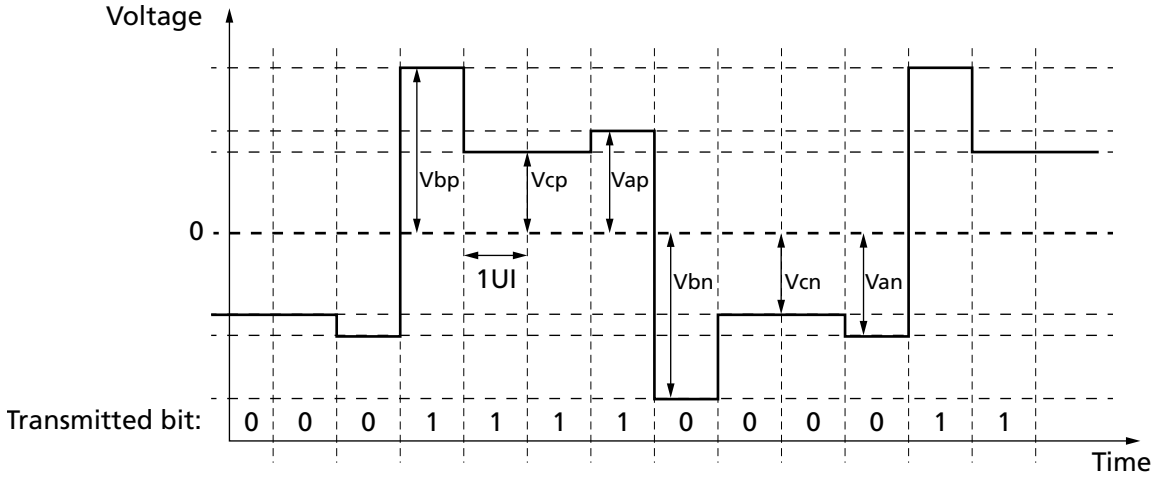
Equalization Schemes

HMC uses equalization schemes on both its transmitter and receiver to mitigate signal integrity issues that can arise from media losses, crosstalk, reflection, and intersymbol interference (ISI).

The HMC transmitter implements feed forward equalization (FFE) using a programmable three-tap, baud-spaced finite impulse response (FIR) driver with the following equation: $H(Z) = (C(-1)z^{+1} + C(0)z^{0} + C(+1)z^{-1})$.

The relative weights of C(-1), C(0), and C(+1) are user-configurable to create a wide variety of transmitter finite impulse response pulse shaping filters.

Figure 25: Digital Waveform with Pre-Tap (C(-1) and Post-Tap (C(+1)) On



Note: 1. Pre-tap and post gains are defined as:

$$TX_{pre\_gain} = 20\log10\left(\frac{V_{ap}-V_{an}}{V_{cp}-V_{cn}}\right) \quad (2)$$

$$TX_{post\_gain} = 20\log10\left(\frac{V_{bp}-V_{bn}}{V_{cp}-V_{cn}}\right) \quad (3)$$

Under nominal TX and RX termination impedance conditions, the HMC transmitter supports a minimum 2.5 ±0.5dB pre-tap gain, with a step size no greater than 1.3dB, and a minimum 4.0 ±0.5dB post-tap gain, with a step size no greater than 1.6dB.

The receiver has differential input voltage range sufficient to accept a signal produced at its package balls by the combined transmitter and channel, up to their respective worst conditions. The minimum differential input voltage at the RX input will be when FFE is at its maximum attenuation, which includes tolerance. Pre-tap gain, which will be 3.0dB, and post-tap gain, which will be 4.5dB, are defined by Equations (2) and (3).

The receiver provides a lower-power linear equalizer [that is, continuous time linear equalizer (CTLE)] that complements the transmitter equalization capability to equalize the channel when its insertion loss, return loss, and crosstalk are all at the worst case. The receiver should assume the jitter, differential voltage swing, termination/return loss of the TX are all at worst case conditions and that the FFE taps are at the minimum gain levels, which includes tolerance (pre-tap gain at 2.0dB and post-tap gain at 3.5dB).

Link Bit Rate and Link Bit Error Rate

The HMC link interface is synchronous on both the upstream and downstream lanes. It is not plesiochronous. The choices (optional) for link bit rate are shown in the table below. Any fixed skew is allowable between the reference clock at the host and HMC.

The HMC 30G-VSR link is expected to operate at a BER of 1e-15 or lower.

Table 111: Synchronous Link Bit Rate Specifications

Notes 1 and 2 apply to the entire table

Parameter	Symbol	Value	Unit	Supported REFCLK (f_{REFCLK} = MHz)				Notes
				125	156.25	166.67	312.5	
Bit rate	BR7.5	7.5	Gb/s	Yes	Yes	Yes	Yes	
Bit rate	BR12.5	12.5	Gb/s	Yes	Yes	Yes	Yes	
Bit rate	BR14	14.0	Gb/s	Yes	No	Yes	No	
Bit rate	BR15	15.0	Gb/s	Yes	Yes	Yes	Yes	
Bit rate	BR25	25.0	Gb/s	Yes	Yes	Yes	Yes	
Bit rate	BR28	28.0	Gb/s	Yes	No	Yes	No	
Bit rate	BR30	30.0	Gb/s	Yes	Yes	Yes	Yes	

Notes:
1. Bit-rate tolerance is within ±100 ppm.
2. The link interface is synchronous, that is, the TX and RX run from the same reference clock.

High Speed Signaling Parameters

All parameters within Table 112 (page 105) and Table 113 (page 108) represent those of the HMC. Host transmitter and receiver termination and signaling parameters may differ from those of the HMC as long as the parameters within the tables are still met for the HMC.

Note:

1. Both the host and HMC must use the same reference clock source.
2. The ppm tolerance for the reference clock source must be less than 100 ppm.

Table 112: TX Signaling Parameters

Measurements are assumed to be at the package BGA ball; Parameters listed apply to HMC device only

Parameter	Symbol	Test Conditions/Comments	Min	Typical	Max	Unit	Notes
Link supply TX voltage	AV_{DD}	0.9V ±0.027V	0.873	0.9	0.927	V	1
Differential peak-to-peak output voltage	$V_{DIFF\_TX\ SWING}$	Measured with slow square wave (64 zeros followed by 64 ones) with ideal 100Ω differential termination	750 (with FFE off) 250 (with FFE on)	–	1025	mV	2
Single-ended voltage (with respect to V_{SS}) on V_{TX_P}, V_{TX_N}	V_{TX_SE}	Active mode	0.2 × AV_{DD}	–	0.8 × AV_{DD}	mV	
Down or sleep mode output voltage	V_{TX_PD}		–	See Note 3	–	mV	3
DC common mode output voltage	V_{TX_CM}	V_{TX_CM} = DC of $\lvert V_{TX\_P} + V_{TX\_N} \rvert/2$	–	See Note 4	–	mV	4

Table 112: TX Signaling Parameters (Continued)

Measurements are assumed to be at the package BGA ball; Parameters listed apply to HMC device only

Parameter	Symbol	Test Conditions/Comments	Min	Typical	Max	Unit	Notes
AC common mode noise	$V_{TX\_CM\_NZ\ SQUARE\ WAVE}$	Measured with slow square wave output amplitude	–	–	50	mV_{PP}	
	$V_{TX\_CM\_NZ\ PRBS}$	Measured with PRBS data pattern	–	–	2	mV_{TMS}	
Short circuit current	I_{SHORT_TX}	Output pins shorted to GND or each other	–50	–	50	mA	
Differential TX output rise/fall time	$^{t}TX\_RISE$, $^{t}TX\_FALL$	Measured from 20% to 80% into ideal 100Ω load	8	–	–	ps	
Differential transmitter resistance	R_{OD}		80	100	120	Ω	5
Single-ended transmitter resistance	R_{OSE}		40	50	60	Ω	5
Differential transmitter termination mismatch	$R_{TX\_DELTA\_AC}$	AC-coupled; Measured within a differential pair	–	–	10	%	
	$R_{TX\_DELTA\_DC}$	DC-coupled (V_{HRX} = AV_{DD}/2 ±50mV); Measured within a differential pair	–	–	10	%	6
Output return loss - Relative to 100Ω differential system	$RL_{TX\text{-}DIFF}$	f0 - f1	–	–	–12	dB	
		f1 - fb	–	–	–12 + 12 × log (f/f1)	dB	
Common mode return loss - relative to ideal 25Ω impedance	$RL_{TX\text{-}CM}$	f0 - f2	–	–	–6	dB	
		f2 - fb	–	–	–4		
Jitter generation up to 30 Gb/s	DJ_{TX}	Measured at output ball	–	–	0.15	UI	7, 8
	DCD_{TX}	Measured at output ball; Considered part of and included within DJTX	–	–	0.035	UI	9
	RJ_{TX}	BER = 1e-15 (peak-to-peak jitter)	–	–	0.15	UI	10, 11
	TJ_{TX}	BER = 1e-15		–	0.28	UI	11, 12
Output differential skew	$^{t}TXSKEW\_diff$		–	–	±0.12	UI	
Lane-to-lane output skew at TX	$L_{TX\text{-}SKEW}$	TX output skew is the difference between propogation delays of any two outputs of the same HMC LInk at identical transitions	–	–	200	ps	12

Notes:
1. Reference points are defined at the device package balls.
2. Slow square-wave is defined as repeating 64 zeros followed by 64 ones. Differential voltage measurement is defined by the following figure.
3. TX differential output may be pulled HIGH or LOW, or High-Z.

4. TX output common-mode voltage in a DC-coupled link is dependent on the type of termination used at the receiver. See “Termination Configurations” for supported termination topologies.
5. Refers to the resistive portion of the termination.
6. This behavior is not compensated for with impedance calibration.
7. f0 = 50 MHz, f1 = 0.1714 × fb, f2 = 10 GHz, and fb is the bit rate frequency.
8. DJ_{TX}, RJ_{TX}, and TJ_{TX} are measured with a reference clock recovery unit (CRU) whose jitter transfer function is defined by a “golden” PLL, namely 0dB when f ≥ BR/2578, 20 dB/dec when f< BR/2578.
9. DJ_{TX} is measured with PRBs2^{15} - 1 pattern.
10. DCD_{TX} is measured with a clock pattern.
11. TJ_{TX} is measured with PRBs2^{31} - 1 pattern.
12. RJ_{TX} is measured with PRBs2^{15} - 1 pattern.
13. This parameter applies to HMC device only. It is measured at the crossing point of each differential pair.

Table 113: RX Signaling Parameters

Parameters listed apply to HMC device only

Parameter	Symbol	Test Conditions/Comments	Min	Typical	Max	Unit	Notes
Link supply RX voltage	AV_{DD}	0.9V ±0.027V (±3%)	0.873	0.9	0.927	V	1
Differential peak-to-peak input voltage	V_{DIFF_RX}	$V_{DIFF\_RX} = 2 \times \lvert V_{RX\_P} - V_{RX\_N} \rvert$ Measured with reference load	200	–	1200	mV	2
Single-ended voltage (with respect to V_{SS}) on D+, D-	V_{RX_SE}		0	–	AV_{DD} + 300	mV	
Common mode of the input voltage	V_{RX_CM}	V_{RX_CM} = DC of $\lvert V_{RX\_P} + V_{RX\_N} \rvert/2$	300	–	AV_{DD}	mV	
Short circuit current, RX off	$I_{SHORT\ RX\_OFF}$		–100	–	100	mA	3
Short circuit current, RX on	$I_{SHORT\ RX\_ON}$		–100	–	100	mA	3
Differential input resistance	R_{ID}		80	100	120	Ω	
	R_{ISE}		40	50	60	Ω	
Differential receiver termination mismatch	R_{RX_DELTA}		–	–	10	%	
Differential input loss - Relative to 100Ω differential system	$RL_{RX\text{-}DIFF}$	f0 - f1	12	–	–	dB	4
		f1 - fb	–12 + 12 × log (f/f1)	–	–	dB	4
Common mode return loss - relative to ideal 25Ω	$RL_{RX\text{-}CM}$	f0 - f2	–6	–	–	dB	4
		f2 - fb	–4	–	–	dB	
Jitter tolerance	DJ_{RX}	Comprised of ISI, DCD, reflections, crosstalk, and SJ	–	–	1	UI	
	DDJ_{RX}	Composed of DCD and ISI	–	–	0.60	UI	5
	SJ_{RX}	Sinusoidal, low (MAX) and high (MIN) frequency	5	–	0.05	UI	5, 6
	RJ_{RX}	Random jitter (rms)	–	–	7.5	mUI	5
Input differential skew	R_{XSKEW_diff}		–	–	±0.25	UI	
Lane-to-lane skew at RX	$L_{RX\text{-}SKEW}$	Lane-to-lane skew at a receiver that must be tolerated. RX INPUT skew is the difference between propogation delays of any two inputs of the same HMC LInk at identical transitions	–	–	1066	ps	7

Notes:
1. Reference points are defined at the device package balls.
2. $V_{DIFF\_RX\_min}$ is an estimate that will be revised after the benefit of the RX equalization has been fully assessed. $V_{DIFF\_RX\_ax}$ value is based on a low frequency data pattern (64 ones followed by 64 zeros).
3. The receiver input absolute maximum voltage ratings in the Absolute Maximum Ratings table cannot be exceeded for extended periods of time without affecting device reliability.

4. f0 = 50 MHz, f1 = 0.1714 × fb, f2 = 10 GHz, and fb is the bit rate frequency.
5. RX equalization can effectively open a “closed” eye signal at its input, to achieve a BER of 1E-15 or better. These jitter tolerance specifications correspond to the minimum equalization capability where RX DFE is not required. Assessment of RX equalization capabilities and BER for a given host TX and channel model can be performed using proven device model and simulation methods such as an IBIS-AMI receiver model, or other equivalent circuit or behavioral models.
6. See Figure 27 (page 110).
7. This parameter applies to HMC device only. Lane-to-lane skew contributes directly to memory latency, so minimizing skew between RX lanes is recommended. RX input skew is the difference between the propagation delays of any two inputs of the same link at identical transitions.

Figure 26: TX and RX Return Loss

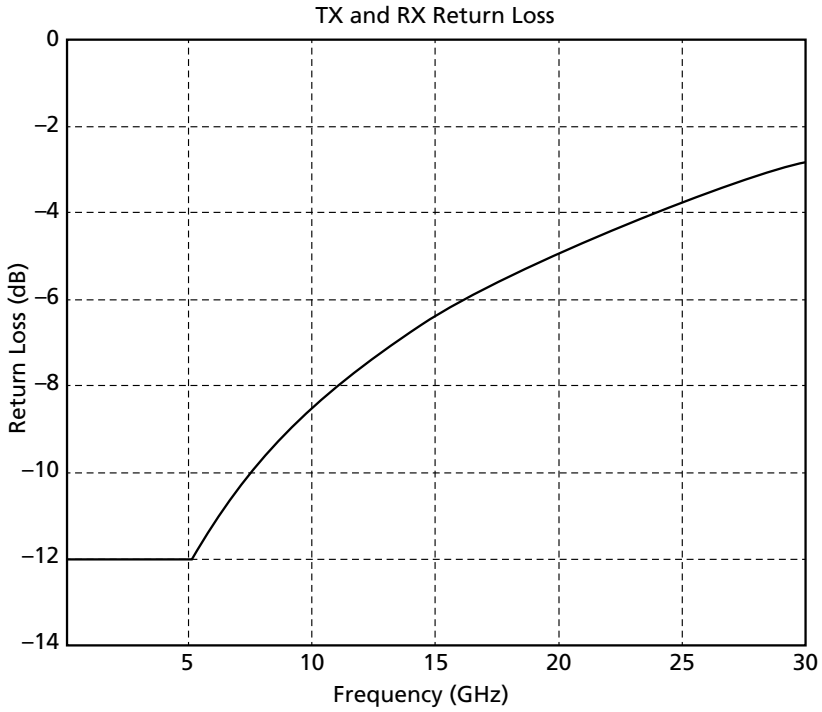
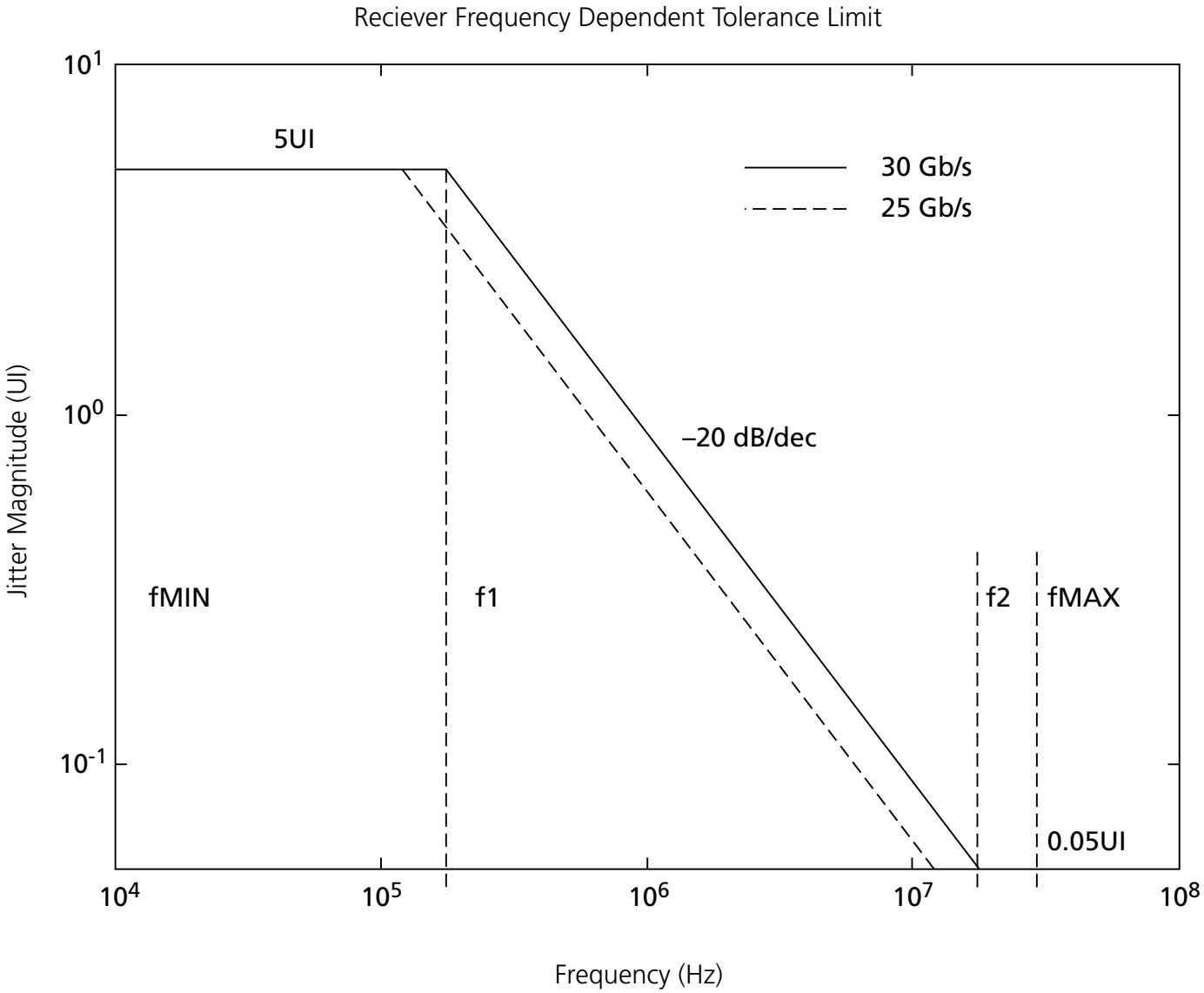


Figure 27: Receiver Sinusoidal Jitter Tolerance



Where: f1 = fRX/257800, f2 = fRX/2578, fMIN = 10 KHz, and fMAX = 30 MHz.

Supported Channels

The HMC-30G-VSR short-reach PHY targets communication at up to 30 Gb/s per lane over channels with ~10dB insertion loss at the Nyquist of the link data rate. The length of the channel supported depends on the PCB trace material used to build the channel. For example, if Megtron-6 material is used, the channel length would be ~12 inches with a link speed of 30 Gb/s. If higher loss materials (such as FR4, Nelco 4000-13) are used, the channel length would be shorter.

Figure 28: Example of HMC-30G-VSR Implementation

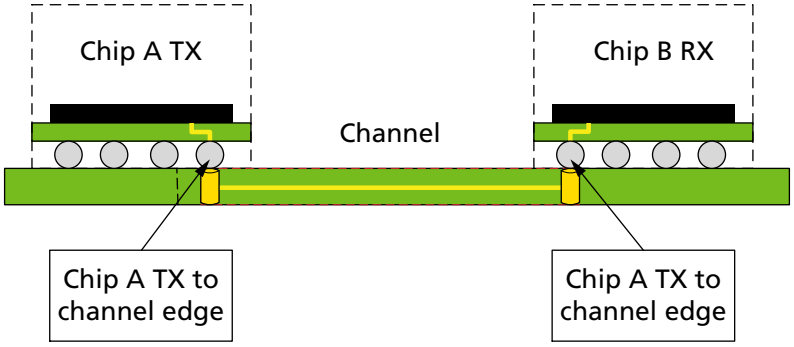
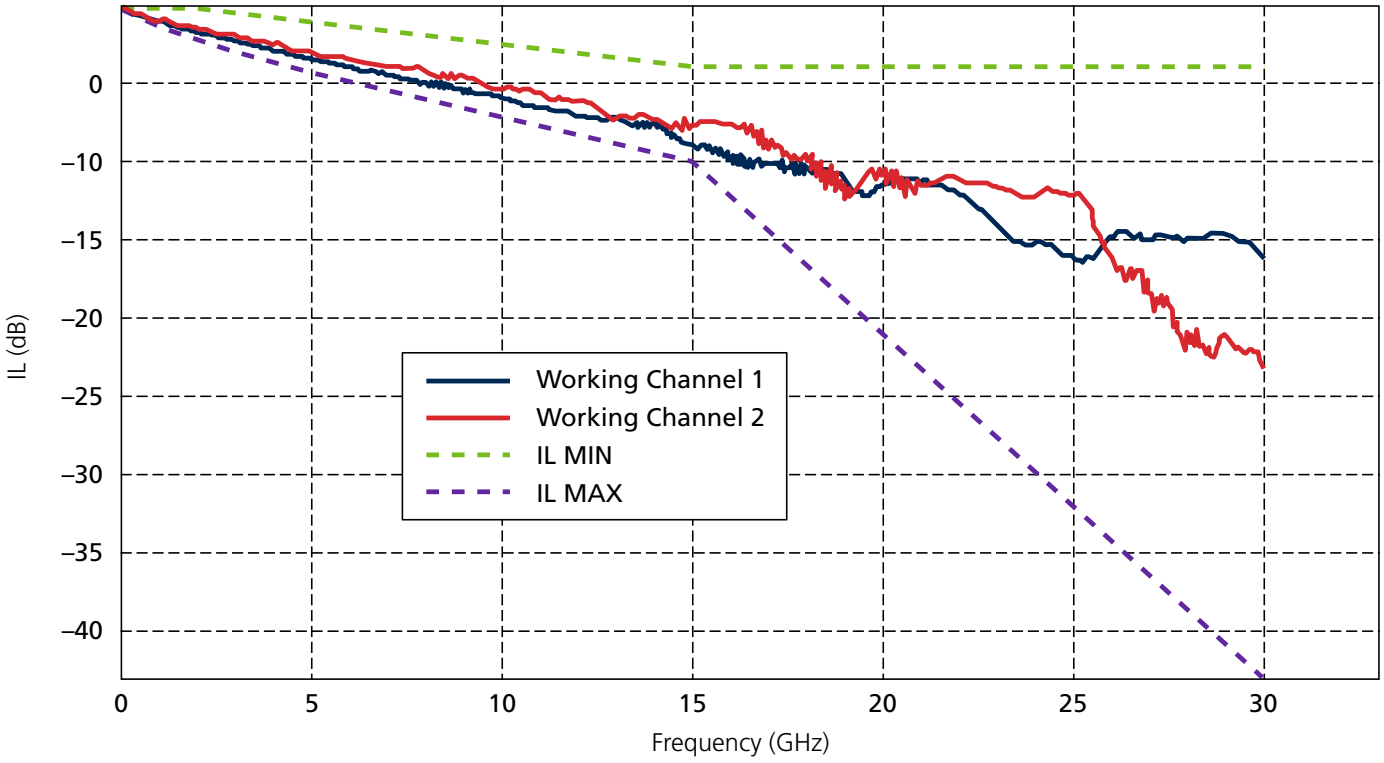


Figure 29: Examples of Channel Insertion Loss and MIN/MAX Limits



The figure above shows the plots for HMC-30G-VSR channels with insertion losses of ~-7dB and ~-9dB at 15 GHz. Both IL minimum and maximum limits are also shown.

In addition to IL, important channel impairment is the insertion loss deviation (ILD), which manifests the channel impedance dis-continuity. Mathematically, ILD is defined as the difference between the IL and the best model fit for IL, namely ILD = IL – IL_fit.

The figure below shows an example of model-fit to the IL, a), and ILD, b). All the informative channel parameters essential for insuring interoperability, are provided in the following table, including IL, ILD, RL, crosstalk, and differential impedance.

Figure 30: Determining Insertion Loss Deviation

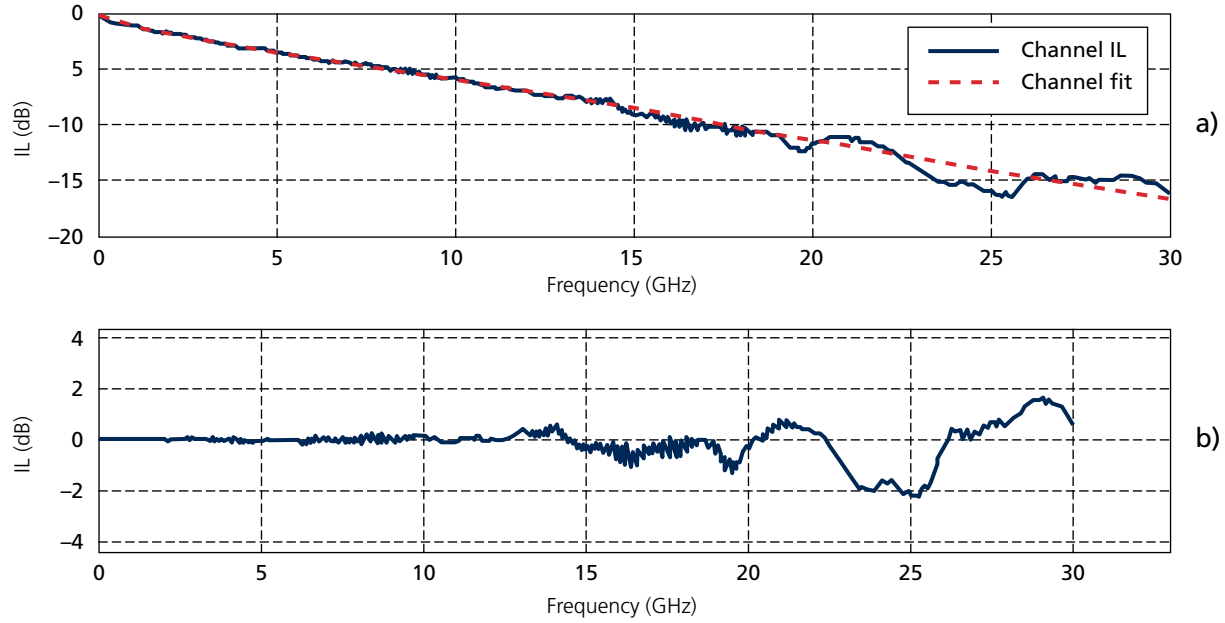
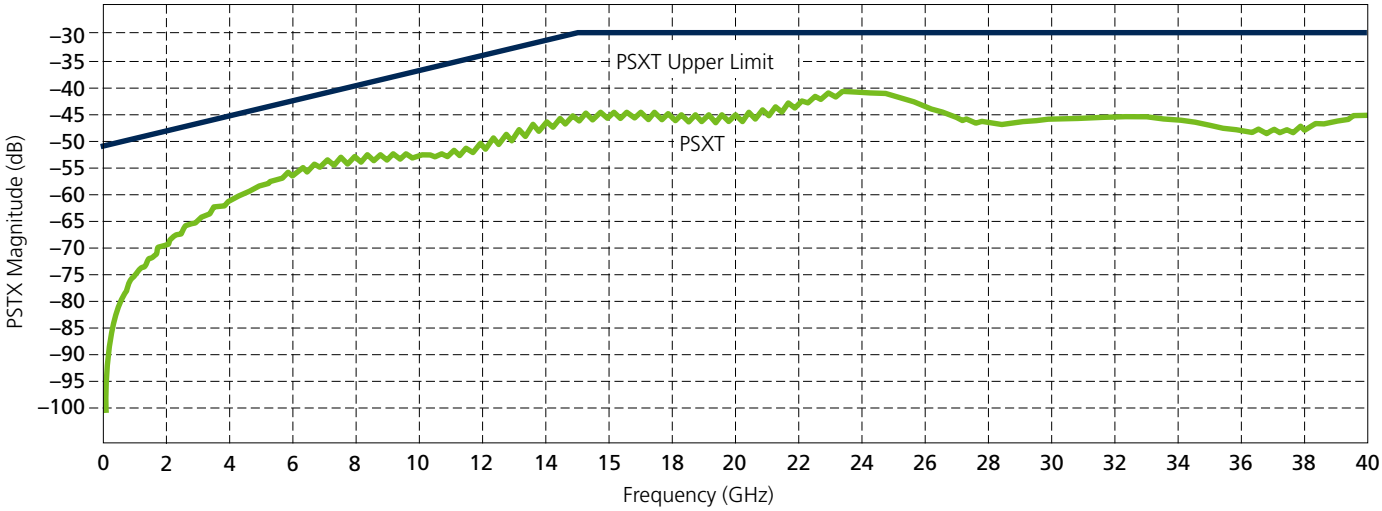


Figure 31: Channel Crosstalk Specification



The fourth channel impairment is return loss (RL). Channel RL is confined by its upper limit mask (see the following the Channel Return Loss figure and Informative Channel Parameters table).

Figure 32: Channel Return Loss

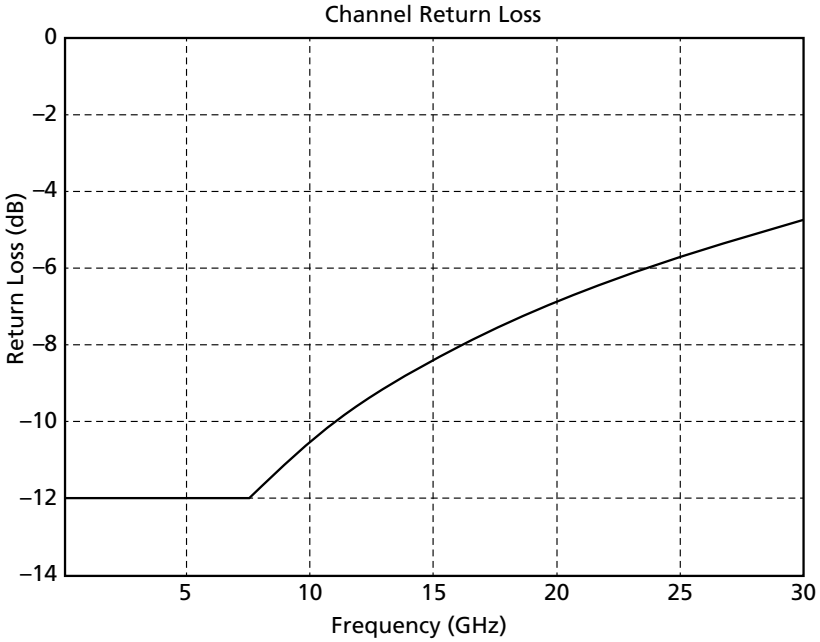


Table 114: Informative Channel Parameters

Parameters	Min	Typ	Max	Notes
Differential insertion loss MIN: IL (dB)	–	–	≤0, fMIN < f < 1 GHz; <–(0.286(f - 1), 1 GHz ≤ f < fb/2; <-4, fb/2 ≤ f ≤ fb	fb: Baud rate, fMIN = 0.05 GHz
Differential insertion loss MAX: IL (dB)	–(0.073 + 0.9458 × sqrt (f × 30/fb) + 0.4176 × f × 30/fb), fMIN ≤ f < fb/2;(23 - 2.2 × f × 30/ fb), fb/2 ≤ f ≤ fb	–	–	
Differential input return loss: RLi (dB)	–	–	≥ –12, fMIN < f < 0.25fb; ≥–12 + 15log(4f/fb), 0.25fb < f < fb	
Differential output return loss: RLo (dB)	–	–	Same as RLi	
Power sum Xtalk: PSXT (dB)	–	–	<–30, 0.5fb ≤ f<fb ≤ 40 × (f/fb) - 50, fMIN ≤ f <0.5fb	
Insertion loss deviation peak (absolute): |ILD_pk| (dB)	–	–	1 + 4(f/fb)	fMIN < f < 0.25fb
	–	–	2	0.25fb < f < 0.75fb
Diff Impedance: Z_diff (Ω)	90	100	110	

Non-High Speed Link Parameters

Table 115: Initialization Timing Parameters

Description	Symbol	Min	Max	Unit
Power supply ramp time	^{t}DD	–	200	ms
Power supply slew rate	V_{DVDT}	–	10	V/ms
Assertion time for P_RST_N	^{t}RST	20	–	ns
P_RST_N slew rate	$V_{RSTDVDT}$	0.1	–	V/ns
PLL and register configuration time	$^{t}INIT$	–	20	ms
Link receiver-phase acquisition (no DFE)	$^{t}RESP1$	–	1	µs
FLIT synchronization time	$^{t}RESP2$	–	1	µs

Table 116: Link Power Management Parameters

Parameter Description	Symbol	Min	Max	Unit	Comments
Voltage Parameters					
LxRXPS input LOW voltage	V_{IL}	–	V_{SS} + 0.5V	V	V_{SS} measured at HMC package ball
LxRXPS input HIGH voltage	V_{IH}	V_{DDK} - 0.5V	–	V	V_{DDK} measured at HMC package ball
LxTXPS output LOW voltage	V_{OL}	–	V_{SS} + 0.1V	V	I_{OL} = 1mA
LxTXPS output HIGH voltage	V_{OH}	V_{DDK} - 0.1V	–	V	I_{OH} = 1mA
Timing Parameters					
Power state transition timing	^{t}PST	–	80	ns	Delay from the transition of a link's LxRXPS signal to the transition of its LxTXPS signal
LxRXPS setup timing to enter down mode	^{t}IS	10	–	ns	Setup time of LxRXPS to P_RST_N transition from LOW to HIGH
Simultaneous power transition stagger	^{t}SS	–	500	ns	Entry/exit time from a state during simultaneous link power state transitions
Sleep mode entry	^{t}SME	–	600	ns	Time from transition LxTXPS LOW to sleep mode entry
Time to transition links from sleep mode to down mode	^{t}SD	–	150	µs	Measured from last link to take LxTXPS LOW to down mode Entry of all links
Time required to stay in self refresh	$^{t}SREF$	1.0	–	ms	Measured from last LxRXPS LOW to first LxRXPS HIGH
Time required to stay within active mode	^{t}OP	1.0	–	ms	Measured by LxTXPS HIGH time
SerDes PLL self calibration timing	^{t}PSC	–	TBD		Required after returning to active mode from down mode
Low power exit timing: transition of LxRXPS to RX PRBS	^{t}RXD	–	50	µs	Delay from transition of responder's LxRXPS to receiving PRBS at its RX (only applicable for cube-to-cube link)
Low power exit timing: transition of LxTXPS to TX PRBS	^{t}TXD	–	50	µs	Delay from transition of responder's LxTXPS to its TX sending PRBS

Table 116: Link Power Management Parameters (Continued)

Parameter Description	Symbol	Min	Max	Unit	Comments
Low power exit timing: TX PRBS to RX NULL	^{t}SigDet	–	50	µs	Delay from responder's TX sending PRBS to receiving NULL FLITs at its RX (only applicable for cube-to-cube link)
Link Retraining Parameters					
TBD					

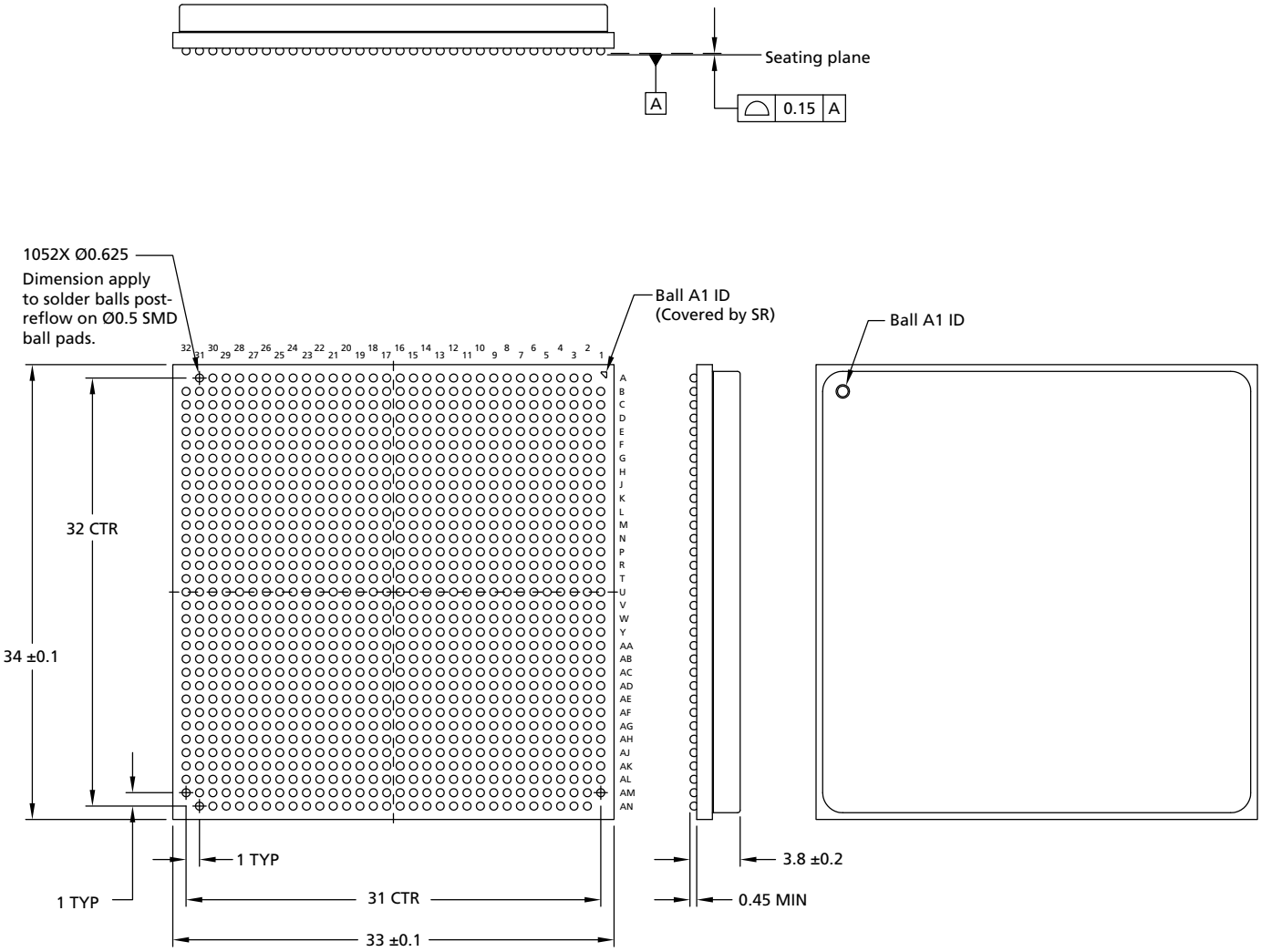
Table 117: Reference Clock Parameters

Parameter Description	Symbol	Min	Nom	Max	Unit	Notes
Reference clock input frequency	^{f}REFCLK	–	125, 156.25, 166.67, 312.50	–	MHz	
Reference clock frequency tolerance	^{f}TOL	–100	–	100	ppm	
AC Coupled Inputs (LVPECL, LVDS Compatible Signaling)						
Input swing voltage: LVPECL	V_{ID_LVPECL}	310	–	–	mV_{PP}	1
Input swing voltage: LVDS	V_{ID_LVDS}	200	–	–	mV_{PP}	1
Reference Clock Phase Noise Requirements						
Offset from nominal input frequency	100Hz	–	–	–85	dBc/Hz	
	1 kHz	–	–	–97	dBc/Hz	
	10 kHz	–	–	–97	dBc/Hz	
	100 kHz	–	–	–114	dBc/Hz	
	1 MHz–1 GHz	–	–	–126	dBc/Hz	

Note: 1. Using a 0.1µF AC coupling capacitor is recommended.

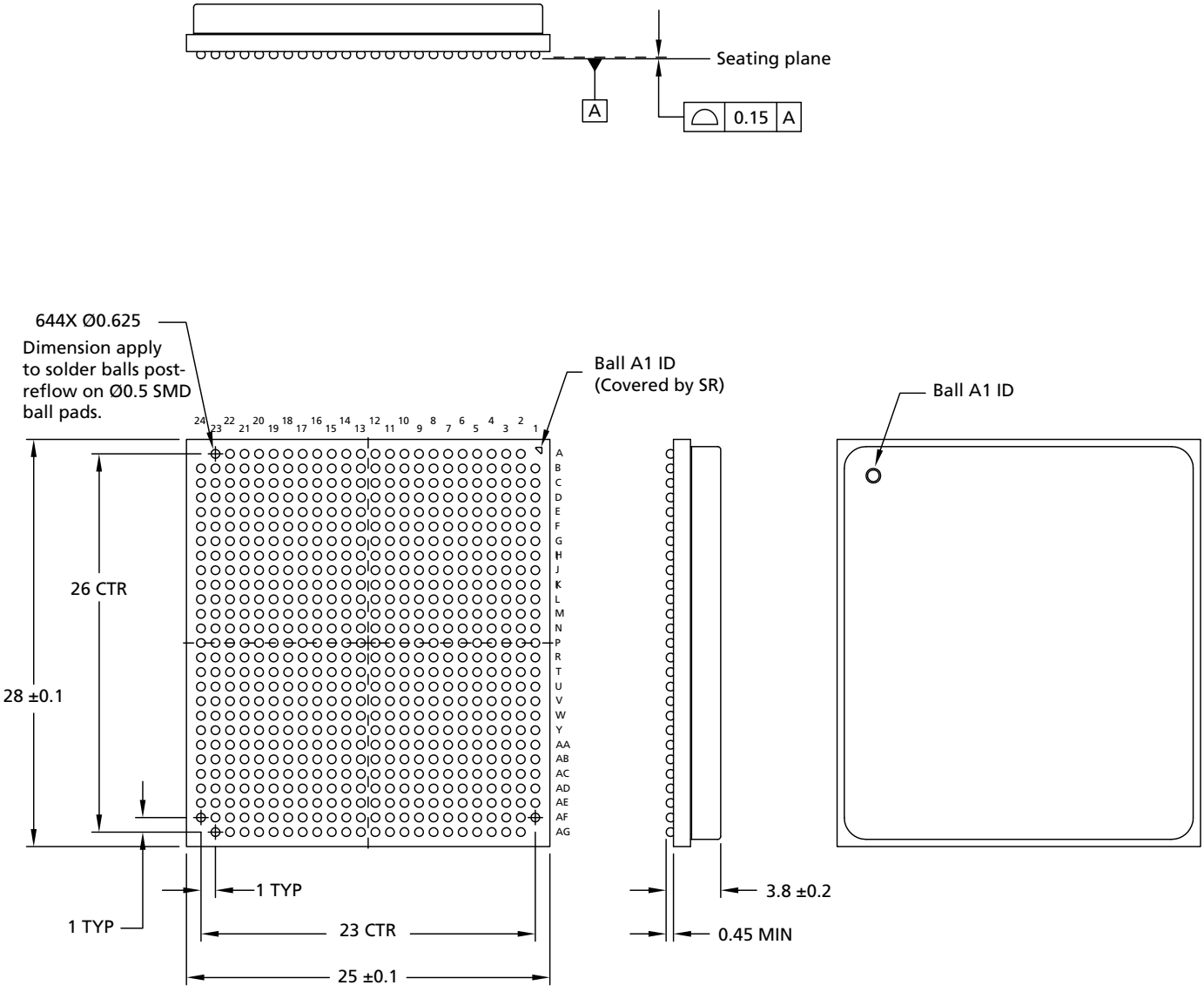
Package Dimensions

Figure 33: HMC Package Drawing (4-Link HMC-30G-VSR Device) — 1.0mm Ball-Pitch



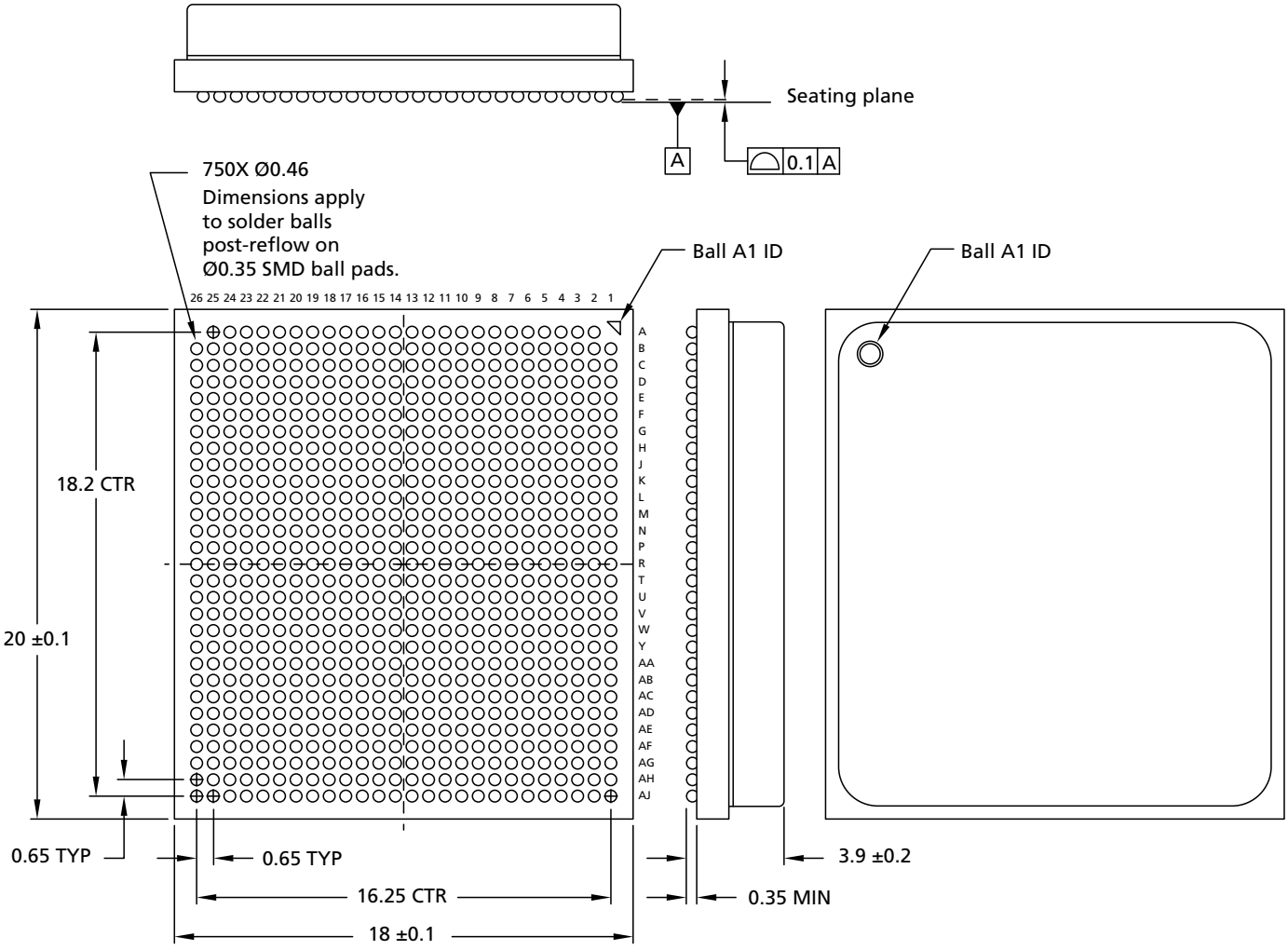
Note: 1. All dimensions are in millimeters.

Figure 34: HMC Package Drawing (2-Link HMC-30G-VSR Device) — 1.0mm Ball-Pitch



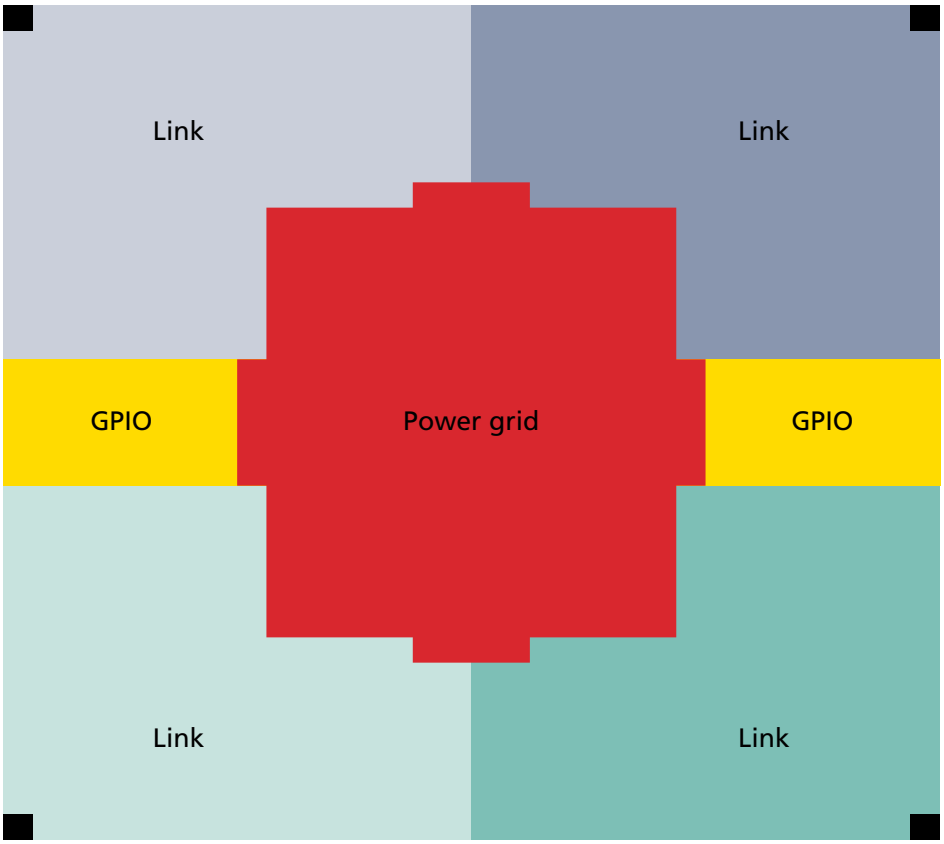
Note: 1. All dimensions are in millimeters.

Figure 35: HMC Package Drawing (2-Link HMC-30G-VSR Device) — 0.65mm Ball-Pitch



Note: 1. All dimensions are in millimeters.

Figure 36: Connector Map — 1.0mm Ball-Pitch 4-Link Package

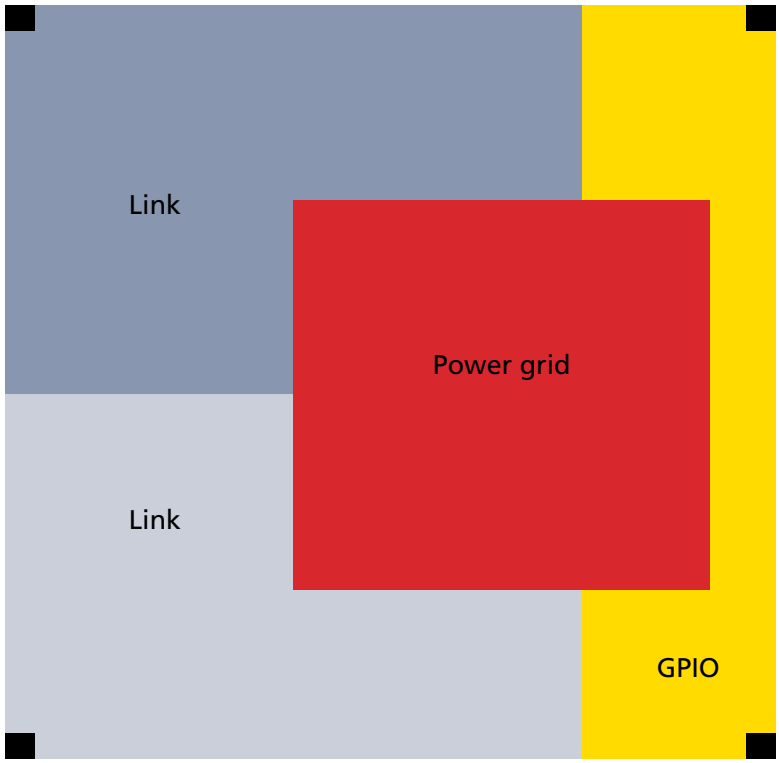


Notes:
1. Pin is used for internal test purposes and is not part of power delivery network. TST_V_{SS} must be tied to V_{SS}; V_{DDK} must be tied to V_{DDK}.
2. Ballout represents an X-ray view, looking through package down to the balls.

Table 118: Functional Pin List — 1.0mm Ball-Pitch 4-Link Package

Link 1 (RX, TX, Power-Down)	66
Link 2 (RX, TX, Power-Down)	66
Link 3 (RX, TX, Power-Down)	66
Link 4 (RX, TX, Power-Down)	66
GPIO (JTAG, I^2C, Clocks, Boot strap, Test)	69
Digital/Analog power	153
V_{SS}/Analog ground	566
Total	1052

Figure 37: Connector Map — 1.0mm Ball-Pitch 2-Link Package

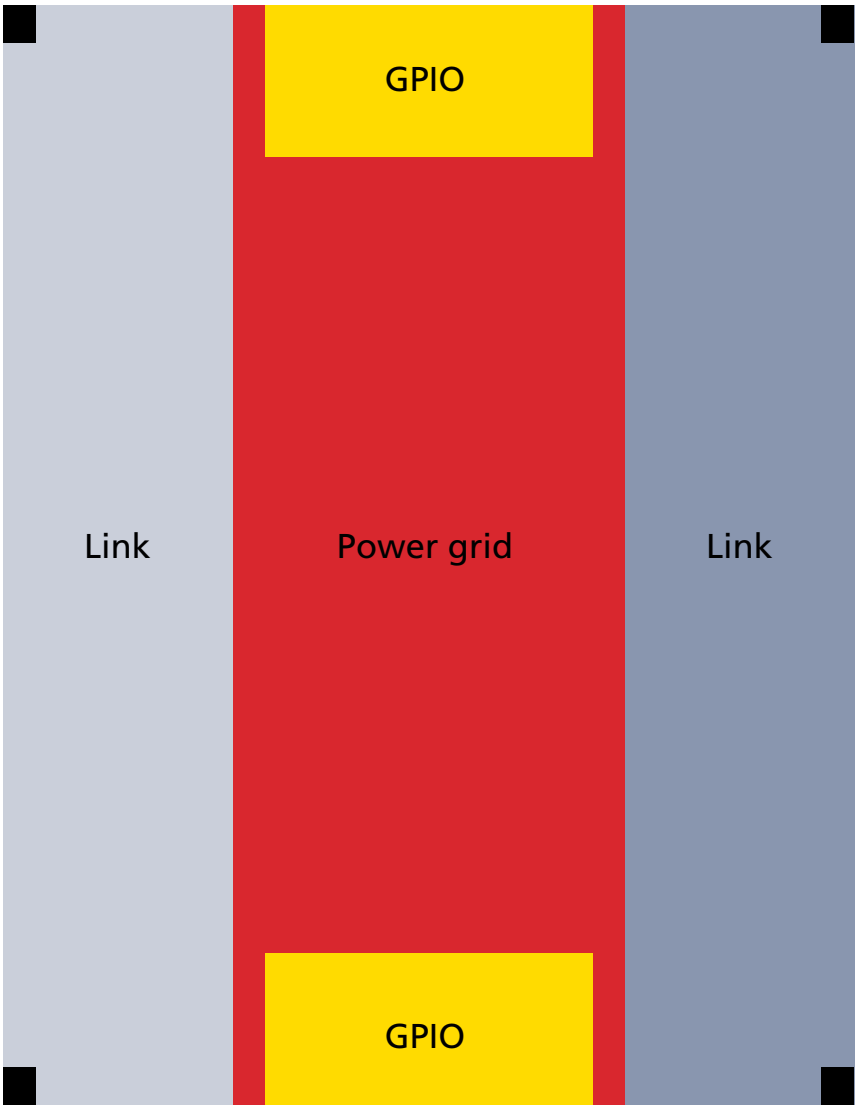


Notes:
1. Pin is used for internal test purposes and is not part of power delivery network. TST_V_{SS} must be tied to V_{SS}; V_{DDK} must be tied to V_{DDK}.
2. Ballout represents an X-ray view, looking through package down to the balls.

Table 119: Functional Pin List — 1.0mm Ball-Pitch 2-Link Package

Link 1 (RX, TX, Power-Down)	66
Link 2 (RX, TX, Power-Down)	66
GPIO (JTAG, I^2C, Clocks, Boot strap, Test)	69
Digital/Analog power	101
V_{SS}/Analog ground	342
Total	644

Figure 38: Connector Map — 0.65mm Ball-Pitch 2-Link Package



Notes:
1. Pin is used for internal test purposes and is not part of power delivery network. TST_V_{SS} must be tied to V_{SS}; V_{DDK} must be tied to V_{DDK}.
2. Ballout represents an X-ray view, looking through package down to the balls.

Table 120: Functional Pin List — 0.65mm Ball-Pitch 2-Link Package

Link 1 (RX, TX, Power-Down)	66
Link 2 (RX, TX, Power-Down)	66
GPIO (JTAG, I^2C, Clocks, Boot strap, Test)	69
Digital/Analog power	157
V_{SS}/Analog ground	392
Total	750

Appendix A: Glossary of Terms

Table 121: Glossary of Terms

Term	Definition
ADR or ADRS	Address
Downstream	Away from the host
FIFO	First in, first out
FLIT	Contraction of FLow unit; the 128-bit unit that forms the smallest quanta of data transported across an HMC link.
Forward	From the point of view of the link master, "forward" meaning in the direction that the link master is driving, on this side of the link.
FRP	Forward retry pointer
HMC	Hybrid memory cube
Host link	HMC link configuration that uses its link slave to receive request packets and its link master to transmit response packets.
SerDes	Serialiser/Deserialiser
Lane	A pair of differential signal lines, one in each direction (transmitters and receivers); multiple lanes are combined to form links.
LFSR	Linear feedback shift register
Link	A fully-duplexed interface connecting two components, implemented with SerDes lanes that carry system commands and data.
LSB	Least significant bit
MSB	Most significant bit
MUE	Multiple uncorrectable errors
Partition	Portion of a memory die that is connected to and controlled by a single vault controller
Pass-thru Link	HMC link configuration that uses its link master to transmit the request packet toward its destination cube and its link slave to receive response packets destined for the host processor.
Quadrant	The eight local vaults associated with a given link that do not require routing across the crossbar switch.
Requester	Represents either a host processor or an HMC link configured as a pass-thru link. A requester transmits packets downstream to the responder.
Responder	Represents an HMC link configured as a host link. A responder transmits packets upstream to the requester.
Return	From the point-of-view of the link master, "return" meaning in the direction on the other side of the link that the local link slave is receiving.
RMW	Read-modify-write sequence
RRP	Return retry pointer.
SBE	Single bit error
TSV	Through-silicon via: Electrical connection through a die to enable die stacking without wires or spacers, providing excellent electrical properties.
Upstream	Toward the host
Vault	Independent memory controller and local memory stacked directly above, in a cube

Revision History

Rev. 2.0 – 11/2014

- HMCC Specification

Hybrid Memory Cube

CONSORTIUM