MICRON TECHNOLOGY, INC.

Hybrid Memory Cube Generation 2

SystemC Model Documentation

Elliott Cooper-Balis

5/23/2017

Micron Confidential and Proprietary

AUTHOR	REVISION	DESCRIPTION	DATE
Elliott Cooper-Balis	2.1	- Increased accuracy - Added trace reader - Added cube-only model	6/28/2016
Elliott Cooper-Balis	2.2	- Increased accuracy	11/14/2016
Elliott Cooper-Balis	2.3	- Increased accuracy -Bug Fixes	3/17/2017
Elliott Cooper-Balis	2.4	-Bug fix in trace reader	5/23/2017

Revision History

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

Contents

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

Introduction

This SystemC model of the Hybrid Memory Cube (HMC) was created to enable accurate, system-level modeling that would not be possible with RTL or SystemVerilog models due to their lengthy execution times and/or inability to interface with existing system simulators. Due to the radically different nature of the HMC, system level analysis is paramount in understanding how this new architecture will impact system performance as a whole. The top-level architecture of the SystemC Model and the library in which it is included is depicted in **Figure 1**. There are three main SystemC modules included in the delivered library: a synthetic traffic generator, an example host controller, and a wrapper around the HMC and its links. A module used to monitor generated traffic is provided as well, and is used to determine when a simulation should terminate.

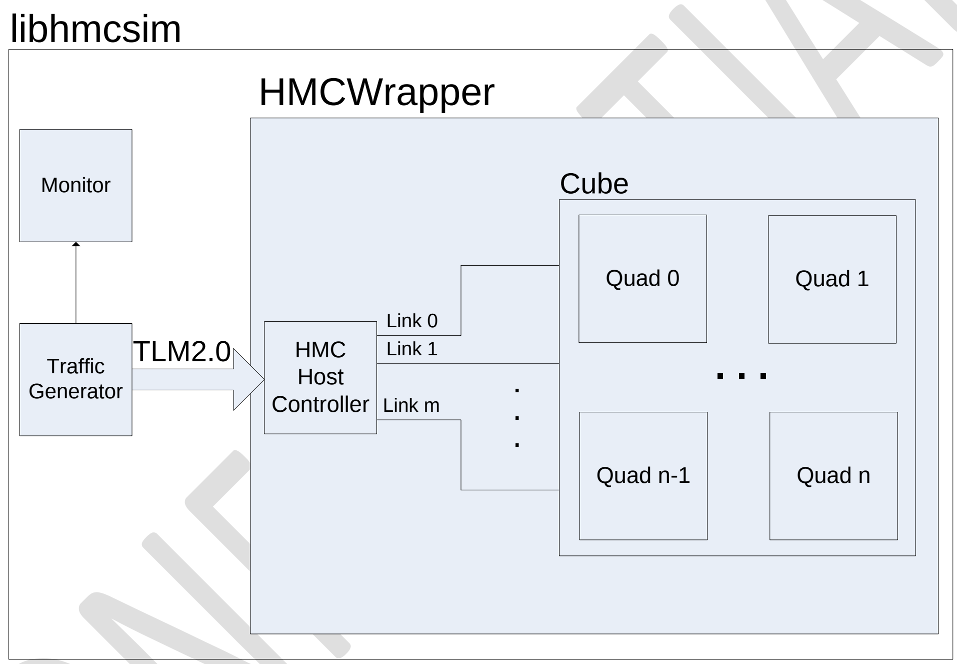


Figure 1. Contents of the SystemC Library

The architecture and abstraction of the SystemC Model mimics that of the HMC Gen2's logical architecture, making it easy to correlate with existing performance metrics. Each logical block in the hardware implementation has a corresponding SystemC module, with similar interfaces, parameters, and timing. This ensures accurate behavior throughout all portions of the model. The model is packaged into a library called `libhmcsim` which can be easily linked to an existing SystemC application. Several header files are included to detail what is accessible within the library.

The `libhmcsim` library also includes a SystemC module of an example host controller. It is designed to receive simple read or write requests over a TLM2.0 socket and decompose them into the packet-based protocol that the HMC expects on each of its links. This allows the end user to focus more on how their request stream or application might perform with an HMC and less on properly formatting packets, link

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

timing, and token handling. A standard TLM2.0 socket is used as an interface with the host controller module. It allows trivial interfacing with other external SystemC modules, such as the included synthetic traffic generator.

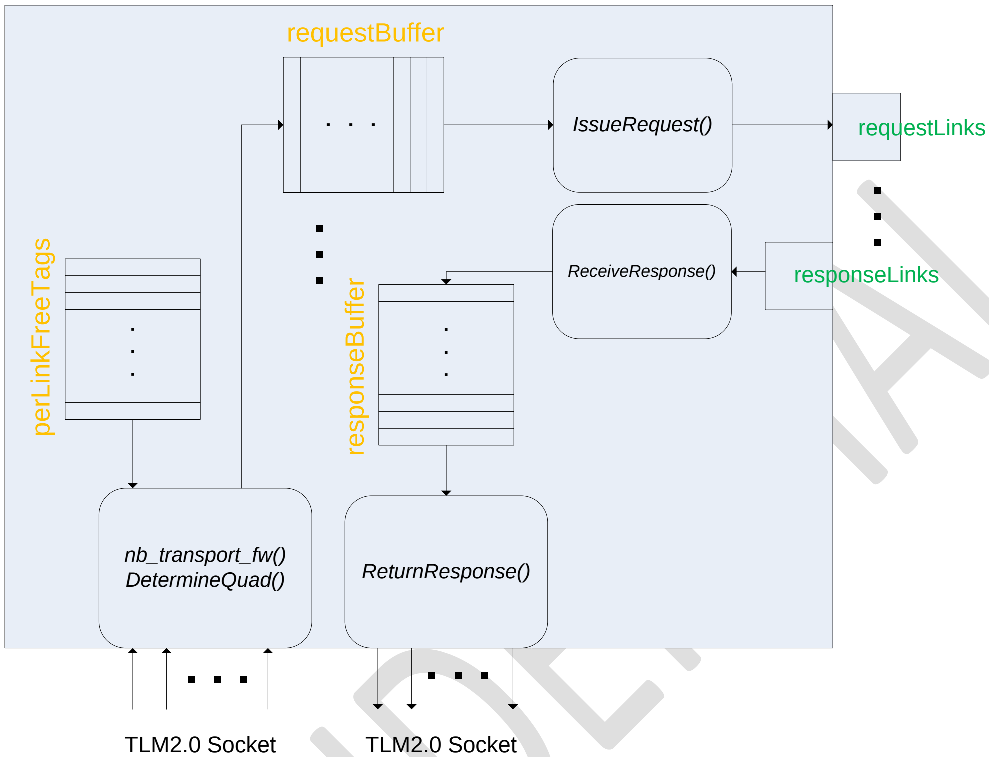


Figure 2. Example Host Controller Included With the Library

Upon receiving a request via the TLM2.0 socket, the host controller module performs two functions: 1) map the request to a particular link, and 2) creates the properly formatted packets (placing them in the corresponding request buffer).

The library includes two ways to issue a request stream over TLM2.0 to the HMCWrapper module: a synthetic traffic generator and a trace file reader. The synthetic traffic generator is used to create a request stream with parameters described in the config.def file at a rate dictated by the clock attached to its clock input port. The parameters used to tailor the request stream (described in detail below) include the read-to-write ratio, request size mix, etc. The trace file reader will parse an input file (whose format is described below), and issue requests based on said file to the model.

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

Simulation Setup

This section provides details on how to compile the included `Tester.cpp` file to run simulations using the `hmc2_hmc_wrapper` SystemC module. `hmc2_hmc_wrapper` includes a host controller module, the Gen2 HMC model, and all of the necessary connections. This is then attached to one or more traffic generators (synthetic of trace-based) which generate requests and receive responses.

Getting Started – Visual Studio 2010/2013

1.Unzip the included file

This should unpack the following files :

- `HMCGen2Library.lib` – Static library of SystemC modules (separated by platform and release)
- `config.def` – Configuration file necessary for execution
- `HMCSim.h` – Header file describing contents of library
- `HMC2Cube.h` – Header file describing cube-only version of Gen2 HMC
- `hmc2_flit_p.h` – Header file of flit software object used with cube-only model
- `Tester.cpp` – Example application using SystemC library
- `mml_generator.cpp/h` – Example source code for the traffic generator
    - **NOTE :** This file contains different logic from that of the `hmc2_generator` class included in the library, and is simply meant to show proper interfacing
- `mml_memory_manager.cpp/h` – Memory management for example traffic generator
- `HMCSim.exe` – Precompiled Tester.cpp binary for Windows machines

2.Start Visual Studio 2010/2013

Add the `Tester.cpp`, `HMCSim.h`, and `config.def` files to an empty project. Add the SystemC `/include` directory to the Include Directories field (under project properties). Add the directory that contains `HMCGen2Library.lib` and `SystemC.lib` to the Library Directories. Add `SystemC.lib` and `HMCGen2Library.lib` to the Additional Dependencies field under the Linker section of the project properties. Add `/vmg` to the command line options.

3. Build solution

Select build solution. If there are no issues, an executable is create that uses the HMC Gen3 library and runs the program within `Tester.cpp`.

4. Run the test application

To run this executable, the `config.def` file that was included must be in the same directory as the executable, and added as a command line argument. This can be done in one of two ways. Either use the windows command prompt to traverse to the directory where the executable and `config.def` reside and running the executable with the `config.def`, or in the project Properties->Debugging change the working directory to `$(SolutionDir)$(Configuration)\` and add `config.def` to the Command Arguments field.

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

Getting Started – Linux

1. Unzip the included file to a folder

This should unpack the following files:

- `libhmcsim.so` – Shared library of SystemC modules (in LinuxLibrary directory)
- `config.def` – Configuration file necessary for execution
- `HMCSim.h` – Header file describing contents of library
- `HMC2Cube.h` – Header file describing cube-only version of Gen2 HMC
- `hmc2_flit_p.h` – Header file of flit software object used with cube-only model
- `Tester.cpp` – Example application using SystemC library
- `mml_generator.cpp/h` – Example source code for the traffic generator
    - **NOTE :** This file contains different logic from that of the `hmc2_generator` class included in the library, and is simply meant to show proper interfacing
- `mml_memory_manager.cpp/h` – TLM memory manager used in generator

2. Compile the example program

To compile the included test program which uses the `libhmcsim` library, you must first ensure that you have the standard SystemC libraries and header files available and note their location (which will be referred to as `$SYSTEMC_HOME`). In the directory where the file was unpacked, run the following command:

```
$> g++ -I$SYSTEMC_HOME/include -L$SYSTEMC_HOME/lib-linux64 -lsystemc -o Tester.exe
Tester.cpp libhmcsim.so
```

The command uses the `-I`, `-L`, and `-l` flags to include the standard SystemC libraries and headers, as well as the `libhmcsim.so` shared library. This will create an executable called `Tester.exe`, which is used in the next step.

3. Run the program

After the program is successfully compiled, it can be run by executing the following command in the same directory where it was compiled:

```
$> ./Tester.exe config.def
```

This will execute a simulation with the parameters dictated in the `config.def` file, which is described below.

4. Simulation output

Upon completion of the simulation, statistics about the execution will be printed to the console. An example of this output can be seen below.

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.
CONFIDENTIAL

Example Simulation Output

```
Reading ini file 'config.def'
=== Creating generator 0 ===
=== Random addressing to local and adjacent vaults
=== Creating generator 1 ===
=== Random addressing to local and adjacent vaults
Starting simulation!!!!
=[19013333 ps]= Generator 0 finished issuing [50.06% R/W
=[19022400 ps]= Generator 1 finished issuing [50.02% R/W
=[19568034 ps]= All responses receives in Generator 0
=[19568034 ps]= GENERATOR 0 DONE!!!
=[19585090 ps]= All responses receives in Generator 1
=[19585090 ps]= GENERATOR 1 DONE!!!

Info: /OSCI/SystemC: Simulation stopped by user.
======= Dumping stats at 19585.1ns / 12240 cube cycles =
==== Generator Requests ====
         (READS)     (WRITES)
 16B ]          0                0
 32B ]          0                0
 48B ]          0                0
 64B ]      10008             9992
 80B ]          0                0
 96B ]          0                0
112B ]          0                0
128B ]          0                0
-----
Total : 20000

== HMC Controller
   ==================================
     DATA BANDWIDTH   : 65.3558 GB/s
     Bytes Moved      : 1280000
   ==================================
   Reads : 10008 (10008 Responses)
   Writes: 9992 (9992 Responses)
   Total : 20000 (20000 Responses)
   R/W%  : 50.04 %

==== Per-Packet Bandwidth ====
Size:16B,32B,48B,64B,80B,96B,112B,128B,
Reads:0,0,0,32.7,0,0,0,0,
Writes:0,0,0,32.65,0,0,0,0,
== Link 0 [16 lanes @ 15Gbps = 30 GB/s]
   Reads : 5006 (Rsp : 5006)
   Writes: 4994 (Rsp : 4994)
   Request Utilization  : 81.58% (24.47 GB/s)
   Response Utilization : 81.71% (24.51 GB/s)
   Effective Bandwidth  : 32.68 GB/s [640000 bytes]
        Read Bandwidth  : 16.36 GB/s [320384 bytes]
       Write Bandwidth  : 16.32 GB/s [319616 bytes]
   Link Avg. Latency : 268.3 ns
   Max Latency       : 461.5 ns
   Min Latency       : 81.54 ns
== Link 1 [16 lanes @ 15Gbps = 30 GB/s]
   Reads : 5002 (Rsp : 5002)
   Writes: 4998 (Rsp : 4998)
   Request Utilization  : 81.62% (24.49 GB/s)
   Response Utilization : 81.67% (24.5 GB/s)
   Effective Bandwidth  : 32.68 GB/s [640000 bytes]
        Read Bandwidth  : 16.35 GB/s [320128 bytes]
```

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

```
        Write Bandwidth  : 16.33 GB/s [319872 bytes]
    Link Avg. Latency : 268.8 ns
    Max Latency       : 464.4 ns
    Min Latency       : 82.6 ns
 == Link 2 [16 lanes @ 15Gbps = 30 GB/s]
    Reads : 0 (Rsp : 0)
    Writes: 0 (Rsp : 0)
    Request Utilization  : 0% (0 GB/s)
    Response Utilization : 0% (0 GB/s)
    Effective Bandwidth  : 0 GB/s [0 bytes]
         Read Bandwidth  : 0 GB/s [0 bytes]
        Write Bandwidth  : 0 GB/s [0 bytes]
    Link Avg. Latency : -
    Max Latency       : -
    Min Latency       : -
 == Link 3 [16 lanes @ 15Gbps = 30 GB/s]
    Reads : 0 (Rsp : 0)
    Writes: 0 (Rsp : 0)
    Request Utilization  : 0% (0 GB/s)
    Response Utilization : 0% (0 GB/s)
    Effective Bandwidth  : 0 GB/s [0 bytes]
         Read Bandwidth  : 0 GB/s [0 bytes]
        Write Bandwidth  : 0 GB/s [0 bytes]
    Link Avg. Latency : -
    Max Latency       : -
    Min Latency       : -
 == Vault 0 at 1.959e+004ns
         Bank Counts : [153 159 154 151 162 153 162 159 ]
         Reads       : 625
         Writes      : 628
         Total       : 1253
         Vault BW    : 4.16 GB/s
 == Vault 1 at 1.959e+004ns
         Bank Counts : [153 153 159 163 149 162 159 158 ]
         Reads       : 613
         Writes      : 643
         Total       : 1256
         Vault BW    : 4.17 GB/s
 == Vault 2 at 1.959e+004ns
         Bank Counts : [154 160 155 155 162 155 152 158 ]
         Reads       : 626
         Writes      : 625
         Total       : 1251
         Vault BW    : 4.153 GB/s
 == Vault 3 at 1.959e+004ns
         Bank Counts : [165 151 159 155 151 159 156 152 ]
         Reads       : 645
         Writes      : 603
         Total       : 1248
         Vault BW    : 4.144 GB/s
 == Vault 4 at 1.959e+004ns
         Bank Counts : [170 157 155 161 147 169 156 141 ]
         Reads       : 622
         Writes      : 634
         Total       : 1256
         Vault BW    : 4.17 GB/s
 == Vault 5 at 1.959e+004ns
         Bank Counts : [146 156 161 147 164 150 163 166 ]
         Reads       : 624
         Writes      : 629
         Total       : 1253
         Vault BW    : 4.16 GB/s
```

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

```
== Vault 6 at 1.959e+004ns
        Bank Counts : [160 149 151 165 151 160 161 152 ]
        Reads       : 630
        Writes      : 619
        Total       : 1249
        Vault BW    : 4.147 GB/s
== Vault 7 at 1.959e+004ns
        Bank Counts : [148 157 155 153 166 151 153 167 ]
        Reads       : 633
        Writes      : 617
        Total       : 1250
        Vault BW    : 4.15 GB/s
== Vault 8 at 1.959e+004ns
        Bank Counts : [157 151 161 159 151 156 154 156 ]
        Reads       : 623
        Writes      : 622
        Total       : 1245
        Vault BW    : 4.134 GB/s
== Vault 9 at 1.959e+004ns
        Bank Counts : [160 161 149 152 162 151 153 157 ]
        Reads       : 618
        Writes      : 627
        Total       : 1245
        Vault BW    : 4.134 GB/s
== Vault 10 at 1.959e+004ns
        Bank Counts : [160 152 152 158 153 161 159 154 ]
        Reads       : 626
        Writes      : 623
        Total       : 1249
        Vault BW    : 4.147 GB/s
== Vault 11 at 1.959e+004ns
        Bank Counts : [147 161 159 155 159 154 159 159 ]
        Reads       : 630
        Writes      : 623
        Total       : 1253
        Vault BW    : 4.16 GB/s
== Vault 12 at 1.959e+004ns
        Bank Counts : [147 157 157 151 164 146 155 167 ]
        Reads       : 611
        Writes      : 633
        Total       : 1244
        Vault BW    : 4.13 GB/s
== Vault 13 at 1.959e+004ns
        Bank Counts : [162 157 152 162 147 166 152 148 ]
        Reads       : 628
        Writes      : 618
        Total       : 1246
        Vault BW    : 4.137 GB/s
== Vault 14 at 1.959e+004ns
        Bank Counts : [149 162 161 150 162 151 155 163 ]
        Reads       : 643
        Writes      : 610
        Total       : 1253
        Vault BW    : 4.16 GB/s
== Vault 15 at 1.959e+004ns
        Bank Counts : [167 153 156 160 148 160 159 146 ]
        Reads       : 611
        Writes      : 638
        Total       : 1249
        Vault BW    : 4.147 GB/s
```


Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.
CONFIDENTIAL

Bandwidth Calculations

All bandwidth metrics printed at the end of a simulation follow the formula in **Figure 3** below. The calculation uses the time of first activity on any link (`t`$_0$) and the end time of the simulation (`t`$_{end}$) as the period over witch bandwidth is calculated, and the total data transferred for particular block/bus/data path.

If a measurement of peak achieved bandwidth is desired, do not begin issuing requests until all links are ready and available to receive, and end the simulation after a reasonable period of steady state (as opposed to waiting for all responses to be returned).

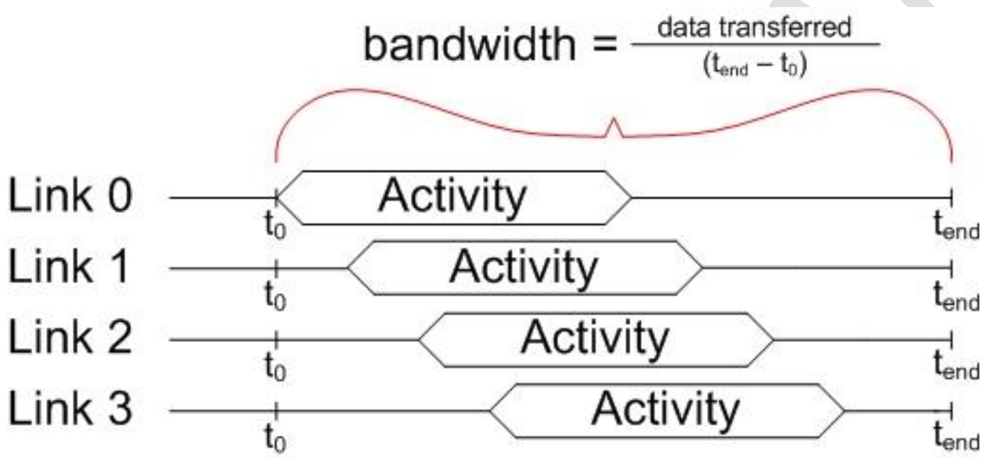


Figure 3. Visual description of how bandwidths are calculated

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

Library Header File

HMCSim.h

The library comes with a header file (`HMCSim.h`) that outlines the accessible classes and functions within the library.

- `class Config;` - Forward declaration of the struct which contains all the parameters that define the system
- `class hmc2_call_wrapper;` - Forward declaration of the HMC's host controller module callbacks
- `class hmc2_generator/hmc2_trace_reader;` - SystemC module that generates synthetic traffic.
    - `SetClock(sc_clock *clock);` – Attaches a clock that dictates how frequently requests are generated
    - `SetDone(sc_signal<bool> *done);` – Attaches the signal that indicates the generator has issued all requests and received all responses
    - `GetRequestSocket();` – Returns a reference to the generator's TLM2.0 `simple_initiator_socket` so it can be bound to the HMC host controller module
- `class hmc2_hmc_wrapper;` - SystemC module that contains both the HMC host controller and the actual HMC with all of the necessary connections
    - `GetControllerSockets()` – Returns a vector of references to the HMC host controller's (`hmc2_host_controller`) TLM2.0 `simple_target_sockets`. Used to bind external sockets to the host controller module
    - `DumpStats()` – Prints collected stats to the console
- `GetConfig(std::string configFilename)` – Uses the filename which was passed in as a command line argument to populate the `Config` object
- `GetHMCWrapper(const Config& config)` – Returns a reference to the `hmc2_hmc_wrapper` object
- `GetGenerator/TraceReader(sc_core::sc_module_name module_name,`
  `unsigned index,`
  `const Config& config)` – Returns a reference to a `hmc2_generator` module with a name, index, and parameters passed in as arguments
- `GetNumGenerators()` – Returns the number of traffic generators to be attached to the HMC host controller. Necessary to ensure full TLM socket binding
- `TraceEnabled()` – Returns flag that dictates whether or not trace file input is enabled
- `class Monitor;` - Simple SystemC module that keeps track of what generators have finished, and ends the simulation accordingly

HMC2Cube.h

This is the header file that allows instantiation of a cube-only model of the Gen2 HMC. **NOTE:** This header file is for users who wish to write their own SystemC host controller module

- `class hmc2_cube` – Forward declaration of Gen2 HMC SystemC module
- `class Config` – Forward declaration of the struct which contains all system parameters
- `GetCube()` – Creates Gen2 HMC SystemC module and returns reference to object
- `sc_vector<sc_in<Flit_P> >* GetRequestLinks()` – Returns a handle to a vector of the SystemC ports used to send flits to the HMC
- `sc_vector<sc_out<Flit_P> >* GetResponseLinks()` – Returns a handle to a vector of the SystemC ports used to receive flits from the HMC

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

- `GetLinkRate()` – Returns sc_time object corresponding to flit rate on a particular link
- `DumpStats()` – Prints statistics to the console

hmc2_flit_p.h

This is the header file that allows the user to create a flit software object that is used to communicate with the cube-only model of the Gen2 HMC. **NOTE:** Some of the member fields and functions in this file can be ignored, as they will not be used directly by the user

- `enum HMCCommand` – The different command types of a `Flit_P` class
- `class Flit_P` – Software object representation of a flit used to model the HMC protocol
    - `Flit_P()` – Constructor that creates a `Flit_P` object using function arguments
    - `AttachTokens(unsigned tokenCount)` – Method used to attach tokens to a tail flit
    - `GetTag()` – Returns the tag associated with this flit
    - `IsHead()` – Returns boolean indicating if this flit is a header flit
    - `IsTail()` – Returns boolean indicating if this flit is a tail flit
    - `GetAddress()` – Returns physical address of the request that this flit is a part of
    - `GetRequestSize()` – Returns the size of the request that this flit is a part of
    - `GetCommand()` – Returns the command type for packet that this flit is a part of
    - `GetNumReturnTokens()` – Returns the number of tokens attached to this flit
        - **NOTE :** Tokens are always attached to tail flits
    - `GetFlitNumber()` – Returns the number indicating this flits place within a whole packet
    - `operator<<()` – Allows `Flit_P` object to be printed to a stream

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

Configuration File (`config.def`)

All aspects of the simulation and the SystemC Model are dictated by the config.def file. The format of the file is:

```
PARAMETER_NAME=value
```

Each parameter is described below:

config.def Parameters	Description	Supported Values
DUMP_STATS_FILE	Enables/Disables the writing of output statistics to a file	true/false Default : false
STATS_FILENAME	Specifies the filename for the output statistics file. **NOTE:** No quotes on filename	Non-empty string Default : stats.txt
NUM_GENERATORS	Dictates the number of traffic generators that are attached to the HMC host controller module	Min : 1 Max :4 Default : 2
NUM_REQUESTS	Dictates the number of requests each traffic generator will issue	Min : 1 Max : - Default : 10000
REQUEST_STREAM	Dictates what type of request stream will be issued from each generator	0 : Random addressing to any vault 1 : Random addressing to local vaults 2 : Round robin addressing to local vaults 3 : Random addressing to local and adjacent vaults 4 : Round-robin addressing to local then adjacent vaults
ALIGNED_32B	Flag that determines whether generated addresses will be 32-byte aligned	true/false Default : true
PERCENTAGE_READ	Dictates what percentage of the request stream will be read requests	Min : 0 Max: 100 Default : 50
PERCENTAGE_(RD/WR)_xB	Dictates what percentage of the request stream will be made up of that particular size request. **NOTE:** All of these fields must sum to 100	Min: 0 Max: Sum of 100
USE_TRACE	Flag which enables the use of input trace files	True/False Default : False
TRACE_FILENAME_X	Path to input trace file. **NOTE :** Up to 4 different trace files can be used at once. When using multiple trace files, ensure that `NUM_GENERATORS` is the same as the number of	Valid path to trace file

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

	traces you'd like to use and that HOST_MAPPING is set to the value 2	
TRACE_DEBUG	Enables or disables debug output for trace file parsing	True/False Default : False
REQUEST_BUFFER_SIZE	Dictates the size of the request buffer (in flits) in the HMC host controller module	Min:9 Max : - Default:384
TAGS_PER_LINK	Dictates the number of unique tags that each link has to attach to a particular request	Min:1 Max:512 Default:512
TOKENS_PER_LINK	Dictates the number of tokens each link has when sending to the cube	Min : 9 Default : 219 Max : 219
POSTED_WRITES	Flag to determine whether a write request requires a response and whether it should consume a tag	true/false Default : false
RESPONSE_OPEN_LOOP	Flag that determines whether the cube must account for tokens when returning responses to the host controller. Assumes no back-pressure seen by host controller when returning responses back to module which sent the initial request	true/false Default : false
REQUEST_SPACING	Dictates number of link cycles between the tail of the previous request and the header of the next request	Integer value Default : 0
HOST_MAPPING	Determines how the host assigns an incoming request to a link	0 : Random link assignment 1 : Map request to link for local vault 2 : Use TLM Socket index as link index
LINK_SPEED	Determines link clock rate	10, 12.5, 15 Default : 15.0
LINK_WIDTH	Number of data lanes used on the request and response path of each link	8 or 16 Default : 16
MAX_BLOCK_SIZE	Used in address mapping	32, 64, 128 Default : 64
Tj	Operating temperature range	Tj<=105C

Table 1. config.def Parameters

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

Interfacing With Host Controller Model

The interface between the example traffic generator module and the HMC library consists of a variable number of TLM2.0 `simple_initiator_socket`. The number of these sockets is dictated by the `NUM_GENERATORS` field in the `config.def` file. These sockets send and receive `tlm_generic_payload` objects using non-blocking transport calls (`nb_transport_fw()` and `nb_transport_bw()`). This is shown in detail in **Figure 3 & 4** below, which are taken from the TLM2.0 specification. [**NOTE : The SystemC model seamlessly alternates between 2-phase and 4-phase non-blocking transfer modes. See description of Request & Response Path below.]**

The `tlm_generic_payload` object has several fields which are used to send a request to the HMC library (other fields within this object are unused). These fields are set with the following functions within the `tlm_generic_payload`:

- `set_address()` – Sets the address of the transaction. Must be within the bounds dictated by the size of the cube in use (2GB or 4GB).
- `set_data_length()` – Sets the size of the transaction. Sizes are dictated by HMC specification (16B, 32B, 48B, 64B, 80B, 96B, 112B, or 128B).
- `set_command()` – Dictates whether the transaction is a read or a write.
- `set_streaming_width()` – [*Hack*] Dictates which link the request should be issued to when using `HOST_MAPPING` 2.

Request Path

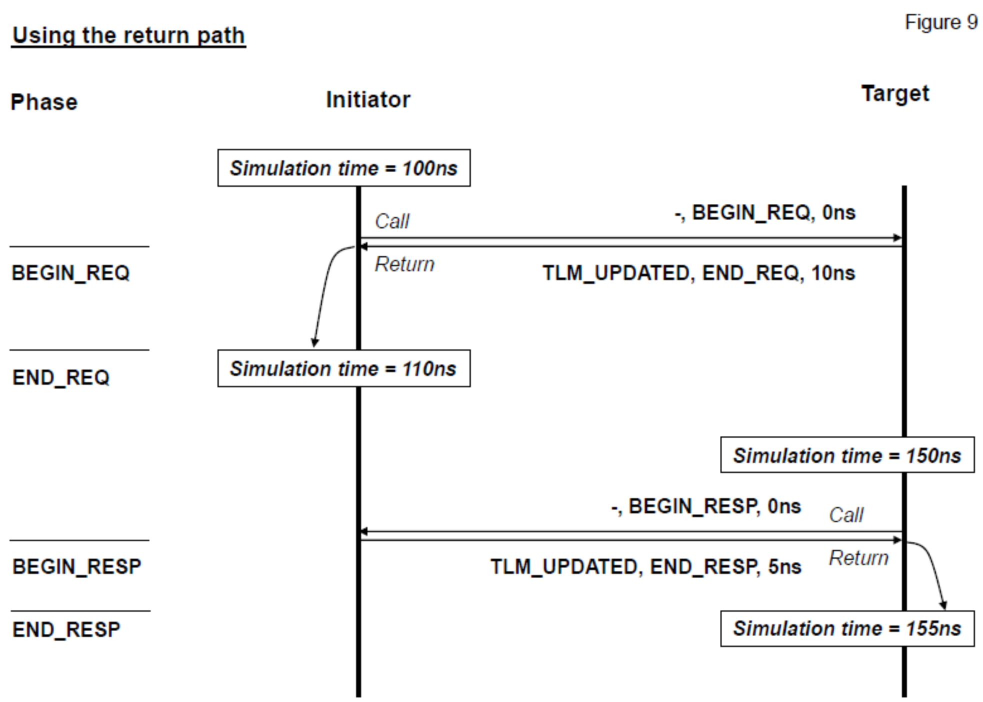
To issue requests, the `nb_transport_fw()` call is used. The return status of this call tells the generator module (or the module making the request) how to proceed. If the return status is `TLM_UPDATED`, then the request was properly received, and the generator can make another request after the amount of time returned in the `delay` reference (in this case, 1.6ns). If the return status was `TLM_ACCEPTED`, then there was some issue with accepting the request (either a full request buffer or lack of tags), and the generator module must wait for an explicit call of `nb_transport_bw` with an `END_REQ` phase as indication that the request was finished.

Response Path

The generator's `nb_transport_bw()` call handles both the end of issuing requests (above), and responses after they have been serviced within the HMC. The action to be taken on this function call is dependent on the `tlm_phase` argument that is passed in. If the phase is `BEGIN_RESP`, then response is being returned to the generator. When the generator is capable of receiving the response, the return value from this call can be either `TLM_COMPLETED` or `TLM_UPDATED`, and the `delay` reference indicates the time necessary to transfer the current response. After this amount of time, another response will be returned. If the return value of the `nb_transport_bw()` call is `TLM_ACCEPTED` the host controller module will wait until an explicit call of `nb_transport_fw()` with an `END_RESP` phase before it continues to return responses. [**Note:** Timing and phases only used when `RESPONSE_OPEN_LOOP` is `false`]

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.
CONFIDENTIAL

The above functionality can be seen in the provided source code for the traffic generator module (`mml_generator.cpp/h`) and the diagrams below show the protocol outlined by the TLM2.0 specification.



Figured 3. Example 2-Phase non-blocking transfer using TLM2.0

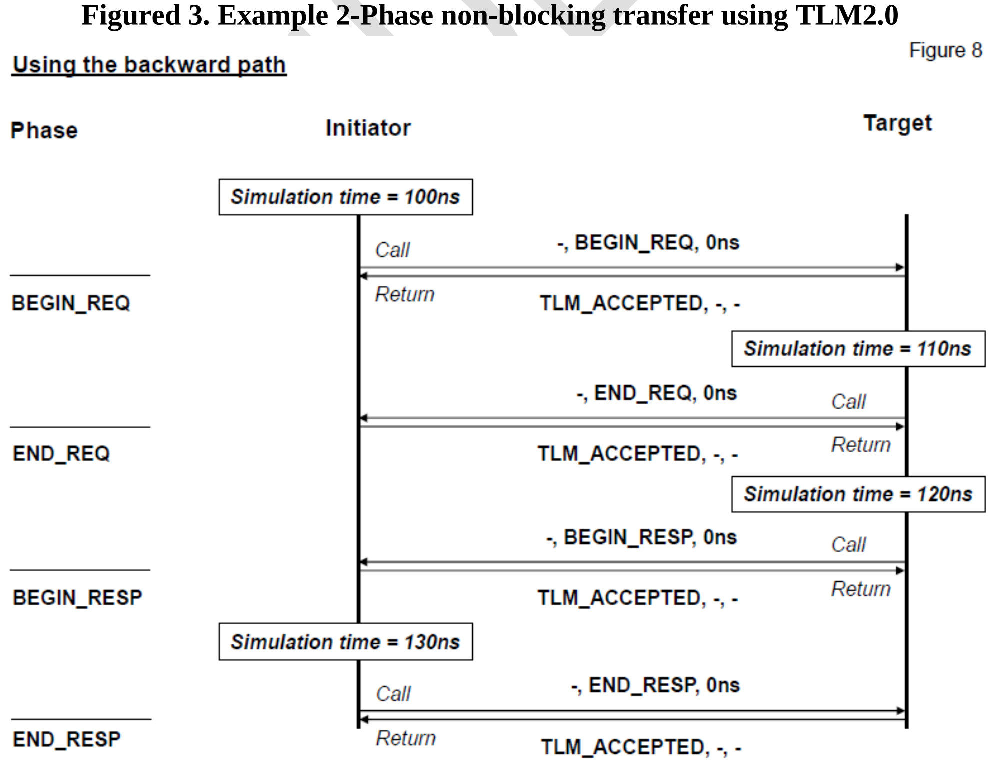


Figure 4. Example 4-Phase non-blocking transfer using TLM2.0

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

Interfacing With Cube Model

The interface to the Gen2 HMC SystemC module uses standard SystemC ports to send and receive flit software objects. This `Flit_P` class contains several fields used to model the HMC protocol, and must be set correctly for the model to work correctly. These fields are set when calling the constructor for each `Flit_P` object and are described (in order) below [**NOTE:** For more detail see the above section which describes the `hmc2_flit_p.h` file]:

- `unsigned tag` – The tag is used to uniquely identify the current request. **NOTE: All requests in the model at a given time must have a globally unique tag. A tag may be re-used once it has evacuated the model**
- `uint64_t address` – The address of this request. Must be properly bound by the size of the model's current configuration
- `HMCCommand command` – The command type for this packet. The full list of command types can be found in the `Flit_P.h` file
- `unsigned length` – The request length (in bytes). Must be one of the values dictated by the specification – 16B, 32B, 48B, 64B, 80B, 96B, 112B, or 128B
- `bool header` – Flag that states this flit is a header flit
- `bool tail` – Flag that states this is a tail flit
- `unsigned number` – The flit number in the current packet. Multi-flit packets (write requests) should be numbered 1 through `length/16+1`

These `Flit_P` objects must be written to the HMC model's input ports at a rate (**`flitRate`**) dictated by both the link speed and link width. This is determined by the following example calculation which assumes full lane width (16) and max lane speed (15 Gbps):

```
#define FLIT_SIZE_IN_BITS 16*8
#define LINK_WIDTH 16
#define LINK_SPEED 15
unsigned uiPerFlit = FLIT_SIZE_IN_BITS / LINK_WIDTH;
float uiPeriod = 1 / LINK_SPEED;
sc_time flitRate = sc_time((float)uiPerFlit * uiPeriod, SC_NS);
```

This value can also be retrieved with the `Cube` class's function `GetLinkRate()` which returns an `sc_time` corresponding to the correct value.

IMPORTANT NOTE FOR 15G LINK RATE : When determining the flit rate for 15G link rates, special care must be taken to account for floating-point round-off. The real-world flit rate for this case is 533 1/3 ps, but SystemC does not have a notion of 1/3. Therefore, the time will be rounded down to 533ps. If this value is used, it will eventually cause incorrect behavior as it is technically faster than the expected rate of issued flits. To account for this, an extra picosecond must be added every third cycle to ensure continued clock alignment. For example :

```
if(clockCounter==2)
   wait(sc_time(534, SC_PS));
else
   wait(sc_time(533, SC_PS));
clockCounter++;
if(clockCounter==3)clockCounter=0;
```


Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

Some other important things to note :

- Unused links on the model must be bound to dummy objects per SystemC requirements that existing ports must be attached to a module.
- A request's tag must be globally unique while within the model. This tag may be reused once that request's response has been evacuated from the model. For `P_WRITE_REQUEST`, the tags should be a value outside of the regular pool (i.e., a value greater than `TAGS_PER_LINK` * 4).
- When not sending a useful flit, a NULL_COMMAND flit must be sent instead

Example Flit Generation Code

The following example shows how a TLM generic payload object can be used to create properly formatted software flits that are used to interface with the Gen2 HMC SystemC module. This code is taken from the example Host Controller module in the included library and references members that are not accessible to users; it is simply meant to be an example of the process of creating `Flit_P` objects. Please reference **Figure 2**.

```
bool HMC::HMCController::ReceiveRequest(tlm::tlm_generic_payload& trans)
{
    unsigned tag;

    //get the address from the TLM generic payload
    unsigned incomingAddress = trans.get_address();
    //get the request size from the TLM generic payload
    unsigned incomingRequestLength = trans.get_data_length();
    //decide which link this request should be issued on
    unsigned incomingDestinationLink = DetermineQuad(incomingAddress);

    //if this request is a read
    if(trans.is_read())
    {
        //make sure there is room in the request buffer
        if(requestBuffers[incomingDestinationLink].size()!=REQUEST_BUFFER_SIZE)
        {
            //check for available tags
            if(perLinkFreeTags[incomingDestinationLink].size())
            {
                tag = perLinkFreeTags[incomingDestinationLink].front();
                perLinkFreeTags[incomingDestinationLink].pop_front();
            }
            //break out if there are no available tags
            else return false;

            //create a read request flit
            Flit_P newFlit = Flit_P(tag, incomingAddress, READ_REQUEST, incomingRequestLength, true, true, 1);

            //add it to buffer used to write to respective link
            requestBuffers[incomingDestinationLink].push_back(newFlit);

            //set event used to issue flits
            issueEvent.notify(flitRate);
            return true;
        }
    }
    else
    {
        //figure out the number of flits this packet will be passed on the request size
        unsigned numFlits = incomingRequestLength/FLIT_SIZE+1;
        //make sure there is room in the request buffer
        if(requestBuffers[incomingDestinationLink].size()+numFlits<=REQUEST_BUFFER_SIZE)
        {
            if(POSTED_WRITES)
            {
                //use tags not part of the normal tag pool for posted write requests
                tag = postedWriteTags;
                postedWriteTags++;
                //release the TLM generic payload
                trans.release();

                //create the correct number of flits
                for(unsigned i=1;i<=numFlits;i++)
                {
                    //header
                    if(i==1)
                    {
                        //create header flit
                        Flit_P newFlit = Flit_P(tag, incomingAddress, P_WRITE_REQUEST, incomingRequestLength, true, false, i);
                        //add it to buffer used to write to respective link
                        requestBuffers[incomingDestinationLink].push_back(newFlit);
                    }
```


Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.
CONFIDENTIAL

```
                //tail
                else if(i==numFlits)
                {
                    //create tail flit
                    Flit_P newFlit = Flit_P(tag, incomingAddress, P_WRITE_REQUEST, incomingRequestLength, false, true, i);
                    //add it to buffer used to write to respective link
                    requestBuffers[incomingDestinationLink].push_back(newFlit);
                }
                //data
                else
                {
                    //create data flits
                    Flit_P newFlit = Flit_P(tag, incomingAddress, P_WRITE_REQUEST, incomingRequestLength, false, false, i);
                    //add it to buffer used to write to respective link
                    requestBuffers[incomingDestinationLink].push_back(newFlit);
                }
            }
        }
        else //non-posted write request
        {
            //make sure we have a tag to assign to this request
            if(perLinkFreeTags[incomingDestinationLink].size())
            {
                tag = perLinkFreeTags[incomingDestinationLink].front();
                perLinkFreeTags[incomingDestinationLink].pop_front();
            }
            //break out if there are no available tags
            else return false;

            //create the correct number of flits
            for(unsigned i=1;i<=numFlits;i++)
            {
                //header
                if(i==1)
                {
                    //create header flit
                    Flit_P newFlit = Flit_P(tag, incomingAddress, WRITE_REQUEST, incomingRequestLength, true, false, i);
                    //add it to buffer used to write to respective link
                    requestBuffers[incomingDestinationLink].push_back(newFlit);
                }
                //tail
                else if(i==numFlits)
                {
                    //create tail flit
                    Flit_P newFlit = Flit_P(tag, incomingAddress, WRITE_REQUEST, incomingRequestLength, false, true, i);
                    //add it to buffer used to write to respective link
                    requestBuffers[incomingDestinationLink].push_back(newFlit);
                }
                //data
                else
                {
                    //create data flit
                    Flit_P newFlit = Flit_P(tag, incomingAddress, WRITE_REQUEST, incomingRequestLength, false, false, i+1);
                    //add it to buffer used to write to respective link
                    requestBuffers[incomingDestinationLink].push_back(newFlit);
                }
            }
        }

        //set event used to issue flits
        issueEvent.notify(flitRate);
        return true;
    }
  }
  return false;
}
```


Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

Input Trace File Format

In lieu of using the synthetic traffic generator, a trace file reader can be used to issue requests to the SystemC Model of the Gen2 HMC. This allows the user to issue a predetermined request stream to the model so as to determine the performance during a very specific use case. The format of the trace file can be seen below:

```
//comment
id:0000,lnk:0,cmd:enu_read,vlt:0,bnk:0,dram:0x04A12,dbytes:32,nop:0
id:0001,lnk:1,cmd:enu_write,vlt:1,bnk:3,dram:0x1AB02,dbytes:64,nop:4
id:0002,lnk:2,cmd:enu_read,vlt:0,bnk:7,dram:0x034F2,dbytes:32,nop:0
id:0003,lnk:3,cmd:enu_read,vlt:11,bnk:2,dram:0x17B30,dbytes:32,nop:3
id:0004,lnk:0,cmd:enu_read,vlt:2,bnk:1,dram:0x220AB,dbytes:64,nop:1
id:0005,lnk:1,cmd:enu_write,vlt:8,bnk:4,dram:0x91E60,dbytes:128,nop:0
//comment
//comment
id:0006,lnk:2,cmd:enu_write,vlt:0,bnk:5,dram:0x147AC,dbytes:32,nop:8
id:0007,lnk:3,cmd:enu_read,vlt:28,bnk:10,dram:0xA0341,dbytes:64,nop:2
id:0008,lnk:0,cmd:enu_read,vlt:16,bnk:6,dram:0x783AD,dbytes:32,nop:0
id:0009,lnk:1,cmd:enu_write,vlt:2,bnk:0,dram:0xB0034,dbytes:128,nop:0
id:000A,lnk:2,cmd:enu_read,vlt:30,bnk:3,dram:0x18ADE,dbytes:256,nop:0
id:000B,lnk:3,cmd:enu_write,vlt:31,bnk:13,dram:0xDEAD0,dbytes:64,nop:5
id:000C,lnk:0,cmd:enu_read,vlt:19,bnk:15,dram:0xBEEF1,dbytes:64,nop:1
id:000D,lnk:0,cmd:enu_read,vlt:27,bnk:2,dram:0x11111,dbytes:32,nop:1
```

The trace file format must be written exactly as what is shown above in order to work properly. The trace file contains many fields that perform functions described below. To enable the trace reader functionality, the `USE_TRACE` parameter in the `config.def` file must be set to `true`, and the `TRACE_FILENAME` field must point to the trace file to be used.

NOTE: The trace reader is considered a "generator" from the model's perspective. There will be a number of trace readers equivalent to the `config.def` parameter `NUM_GENERATORS`. Each trace reader will read, and issue from the same trace. If only one trace is desired, set `NUM_GENERATORS` to 1.

Field	Description	Supported Values
`//`	Indicates the line is a comment and will be ignored	Anything following will be ignored
`id`	Not used by the model, but useful for debugging trace file and keeping track of request count	0 and greater
`lnk`	Determines which link this particular request will be issued on	0, 1, 2, or 3
`cmd`	Determines whether the request is a read or a write	`enu_read` for read `enu_write` for write
`vlt`	Determines which vault the request will be sent to	0 through 15

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

`bnk`	Determines which bank within a vault the request will be sent to	0 through 7
`dram`	Determines the addressing within the DRAM array. A concatenation of the row and column address	0 through 2^20 in hexadecimal
`dbytes`	Determines the request size in number of bytes	16, 32, 48, 64, 80, 96, 112, 128
`nop`	Determines the number of idle link cycles before this request is issued. Link cycles that have been idled for other reasons will count towards this total	0 and greater

Table 2. Description of trace file format

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

HMC Addressing

Request packet headers include an address field of 34 bits for accessing memory, of which only 32 bits are used for HMC Gen2 devices. This address field includes byte, vault, bank, and DRAM address. Memory accesses are in 16-byte granularity. Maximum block size transferred per request is 32B, 64B, or 128B. The maximum block size is set by the host through the address map mode register and determines which additional bits of the address are used for byte selection. The vault address selects 1 of 16 vaults. The bank address selects 1 of 8 banks. The DRAM field selects the row and column address of the DRAM, as shown in the table below.

Address	Description	4-Link Configuration
Byte address	Bytes within the maximum supported block size	The 4 LSBs of the byte address are ignored for READ and WRITE requests
Vault Address	Addresses vaults within the HMC	Lower 2 bits of the vault address specify 1 of 4 vaults within the logic chip quadrant
		Upper 2 bits of the vault address specify 1 of 4 quadrants
Bank Address	Addresses banks within a vault	Addresses 1 of 8 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

Table 2: Addressing Definitions

In **Table 3** we see how the address is mapped to byte, vault, bank, and DRAM fields based on the HMC maximum block size. The byte address field selects a byte within the block as the starting address of a transfer. For the minimum block size of 32B, 5-byte address bits are required and for the maximum block size of 128B, 7-byte address bits are required. The lower 4 bits of the byte address field are always ignored since transfers are on blocks of 16B. The remaining byte address bits are passed to the DRAM as part of the column address. The next 4 address bits after the byte address field select 1 or 16 vaults. The next 3 address bits select 1 of 8 banks. The remaining address bits are then used as the DRAM address.

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.

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

Table 3: Default Address Map Mode Table

CONFIDENTIAL

Micron Technology, Inc., reserves the right to change products or specifications without notice.
©2013 Micron Technology, Inc. All rights reserved.