

Juha Arvio

HIBI MEM DMA

DATASHEET

Table of Contents

[1. Identification 4](#_Toc305171811)

[2. Version history 5](#_Toc305171812)

[2.1 Documentation 5](#_Toc305171813)

[2.2 IP block 5](#_Toc305171814)

[3. Description 6](#_Toc305171815)

[4. Interface 7](#_Toc305171816)

[4.1 Interface signals 8](#_Toc305171817)

[4.2 VHDL generics 9](#_Toc305171818)

[4.3 Register map 10](#_Toc305171819)

[5. Functionality 12](#_Toc305171820)

[5.1 DMA access 12](#_Toc305171821)

[5.1.1 Read channel configuration 15](#_Toc305171822)

[5.1.2 Write channel configuration 15](#_Toc305171823)

[6. Use examples 17](#_Toc305171824)

[7. Performance and resource usage 18](#_Toc305171825)

[8. IP-block files 20](#_Toc305171826)

[8.1 Documentation 20](#_Toc305171827)

[8.2 Main component files 20](#_Toc305171828)

[8.3 Verification files 20](#_Toc305171829)

[9. Dependent components 22](#_Toc305171830)

[10. Verification 24](#_Toc305171831)

[11. Detailed specification for developers 25](#_Toc305171832)

[11.1 Address demux 26](#_Toc305171833)

[11.2 Config. module 26](#_Toc305171834)

[11.3 Memory access arbiter 26](#_Toc305171835)

[11.4 Internal structure and functionality 26](#_Toc305171836)

[11.5 Adding a custom Altera memory controller 28](#_Toc305171837)

List of Abbreviations

fifo first in, first out

FPGA Field Programmable Gate Array

Gbps Giga bits per second

H2P HIBI to PC

HDL Hardware Design Language

HIBI Heterogeneous IP Block Interconnect

HW Hardware

ID Identification

I/O Input / Output

IP Intellectual Property

IRQ Interrupt Request

P2H PC to HIBI

PC Personal Computer

SoC System-on-Chip

SW Software

tbd to be decided

TLP Transaction Layer Protocol

TUT Tampere University of Technology

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

# Identification

Vendor: TTY

Library: storage

Name: HIBI\_MEM\_DMA

Version: 0.2

Author(s): Juha Arvio

Author(s): Juha Arvio

Date created: 01.06.2010

Date last modified: 20.12.2010

# Version history

## Documentation

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Change description | Author |
| 01.06.2010 | 0.1 | Component created | Juha Arvio |
| 20.09.2010 | 0.2 | Latest implementation | Juha Arvio |
| 20.12.2010 | 0.3 | Fixing the structure and styles, added some explanation comments, and also some questions for Juha. | Erno Salminen |
| 21.09.2011 | 0.4 | Converted document into a new format | Juha Arvio |
| 30.09.2011 | 0.5 | Updated document | Juha Arvio |

## IP block

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Change description | Author |
| 01.06.2010 | 0.1 | Component created | Juha Arvio |
| 20.09.2010 | 0.2 | Latest implementation | Juha Arvio |

# 

# Description



*HIBI MEM DMA* is an adapter component to interface memory controllers to the HIBI bus.

Currently only Altera’s DDRx high performance II controllers are supported. The adapter also has DMA functionality and other features. Therefore, some of the first addresses are needed for configuring the DMA.

Features:

* DMA access
  + Dma transfers must be requested and configured first
  + Multiple dma transfers (channels) active simultaneously
  + Can access the whole memory
* Single word access
  + Faster than DMA accesses for small accesses on random memory addresses
  + Memory area access is limited by the amount of DMA channels and the available memory space allocated for the HIBI MEM DMA on the HIBI bus.

DMA access

Physical DDRx memory locations

Single word access

0x00000000

0x00000BFF

0x00000C00

0x1FFFFFFF

# 

# Interface

The interface for this component can be divided into two parts. The HIBI side is connected to the component through a HIBI wrapper r3 component which has in itself two types of signals, one for the normal data fifo and another for the message fifo. The memory controller side has an Avalon MM type interface for the Altera DDRx controllers.

Two interfaces to the HIBI are required to avoid deadlock.



Figure ‑ Interface signals

## Interface signals

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal group | Signal | Width | Dir. | Meaning |
| Reset | rst\_n | 1 | i | Active low reset |
| Clocks | clk | 1 | i | Clock, active on rising edge. |
| Data transfers | hibi\_addr\_in | 32 | i | HIBI incoming address |
| hibi\_data\_in | 32 | i | HIBI incoming data (that will be written to memory) |
| hibi\_comm\_in | 3 | i | HIBI incoming command type (write, read req…) |
| hibi\_empty\_in | 1 | i | HIBI empty, there’s no incoming data |
| hibi\_re\_out | 1 | o | HIBI read enable, pops one incoming data out of the wrapper’s FIFO |
| hibi\_addr\_out | 32 | o | HIBI outgoing address |
| hibi\_data\_out | 32 | o | HIBI outgoing data (that was read from memory) |
| hibi\_comm\_out | 3 | o | HIBI outgoing command type |
| hibi\_full\_in | 1 | i | HIBI full, wrapper cannot accept more outgoing data |
| Channel config. | hibi\_msg\_addr\_in | 32 | i | HIBI message incoming address , denotes the config register |
| hibi\_msg\_data\_in | 32 | i | HIBI message incoming data , the new config value |
| hibi\_msg\_comm\_in | 3 | i | HIBI message incoming comm type |
| hibi\_msg\_empty\_in | 1 | i | HIBI message empty |
| hibi\_msg\_re\_out | 1 | o | HIBI message read enabe |
| hibi\_msg\_addr\_out | 32 | o | HIBI message outgoing address |
| hibi\_msg\_data\_out | 32 | o | HIBI message outgoing data |
| hibi\_msg\_comm\_out | 3 | o | HIBI message write comm type |
| hibi\_msg\_full\_in | 1 | i | HIBI message full |
| Memory controller if. | mem\_init\_done\_in | 1 | i | DDRx memory initialization done |
| mem\_ready\_in | 1 | i | DDRx memory ready for write/read |
| mem\_wr\_req\_out | 1 | o | DDRx memory write request |
| mem\_rd\_req\_out | 1 | o | DDRx memory read request |
| mem\_addr\_out | 32 | o | DDRx memory address |
| mem\_wdata\_out | 32 | o | DDRx memory write data |
| mem\_be\_out | 4 | o | DDRx memory byte enable |
| mem\_rdata\_valid\_in | 1 | i | DDRx memory read data valid |
| mem\_rdata\_in | 32 | i | DDRx memory read data |
| mem\_burst\_begin\_out | 1 | o | DDRx memory write/read burst begin |
| mem\_burst\_size\_out | 1 | o | DDRx memory write/read burst size |

*Table 6-1 Interface signals*

## VHDL generics

|  |  |  |  |
| --- | --- | --- | --- |
| Generic and VHDL default | Type | Value range | Description |
| HIBI\_DATA\_WIDTH | integer | 32, 64 | HIBI data and address width |
| HIBI\_COMP\_ADDR\_WIDTH | integer | 8 |  |
| MEM\_DATA\_WIDTH | integer | 4…256 | physical DDRx data width \* 2 or 4 |
| MEM\_ADDR\_WIDTH | integer | 1…31 | DDRx byte address width |
| MEM\_BE\_WIDTH | integer | 1…32 | DDRx byte enable width |
| BURST\_SIZE\_WIDTH | integer | 2, 3 | DDRx burst size width |
| READ\_CHANNELS | integer | 1…1024 | DMA read channels |
| WRITE\_CHANNELS | integer | 1…1024 | DMA write channels |
| RW\_AMOUNT\_WIDTH | integer | 1…24 | DMA write/read length width |
| RW\_ADDR\_INC\_WIDTH | integer | 1…24 | DMA write/read address increment width |
| RW\_ADDR\_INTERVAL\_WIDTH | integer | 1…24 | DMA write/read interval width |
| RW\_ADDR\_INTERVAL\_INC\_WIDTH | integer | 1…24 | DMA write/read interval increment width |
| DEBUG | integer | 0, 1 | enable debug functionality in simulation |

Table 6- VHDL generics

## Register map

The registers are accessed using word addresses as can be seen on table . Byte addressing could be used in the future and the appropriate register map can be seen on table .

|  |  |  |
| --- | --- | --- |
| offset | Register | Description |
| 0x00000 | IP information regs | reserved |
| ... |  |  |
| 0x0001F | IP information regs | reserved |
| 0x00020 | Config | Configure HIBI wr access arbiter and mem. access arbiter |
| 0x00021 | Read request | Request a read channel |
| 0x00022 | Write request | Request a write channel |
| ... |  |  |
| 0x00100 | Read channel 0 configure |  |
| ... |  |  |
| 0x001FF | Read channel 255 configure |  |
| 0x00200 | Write channel 0 configure/data write register |  |
| ... |  |  |
| 0x002FF | Write channel 255 configure/ data write register |  |
| 0x00300 | Direct access |  |
| ... |  |  |
| 0x1FFFFFFF | Direct access |  |

*Table* *6-3 The register map using word-addresses*

|  |  |  |
| --- | --- | --- |
| offset | Register | Description |
| 0x00000 | IP information regs | reserved |
| ... |  |  |
| 0x0007F | IP information regs | reserved |
| 0x00080 | config | Configure HIBI wr access arbiter and mem. access arbiter |
| 0x00084 | Read request | Request a read channel |
| 0x00088 | Write request | Request a write channel |
| ... |  |  |
| 0x00400 | Read channel 0 configure |  |
| ... |  |  |
| 0x007FF | Read channel 255 configure |  |
| 0x00800 | Read channel 0 configure |  |
| ... |  |  |
| 0x00BFF | Write channel 255 configure |  |
| 0x00C00 | Direct access |  |
| ... |  |  |
| 0x1FFFFFFF | Direct access |  |

*Table* *6-4 The register map using byte addresses*

# Functionality

The interfaced memory can be read and written to by two main methods. These methods include access through the DMA functionality and direct access. In direct access, the incoming address simply denotes the accessed memory location and only single-word read or write accesses are supported. This is convenient for small, random accesses but larger transfers benefit from DMA functionality.

## DMA access

Controller supports multiple DMA *channels*. They must be requested and configured before usage. There are separate channels for read and write operations, but the procedure is pretty much the same.

Configuring the DMA operations:

1. Request a channel by sending **your** HIBI address to a request register with a HIBI **message write**. There is one request register for read operations and one for writes (see table for register map description)
2. HIBI\_MEM\_DMA replies with *offset value* that denotes the reserved channel (permission granted if offset differs from zero)
3. Calculate the actual configuration register address (Add offset to memory DMA’s base HIBI address)
4. Write all the configuration words to the same configuration register with HIBI message writes
5. Operation starts when the last configuration word is written to register (4-5 words for read and 3-4 for write, see below)

Figure shows an example of request procedure. First, there is request but it returns zero because all the channels are reserved. Then, IP retries the request and this time it succeeds.

**HIBI\_MEM\_DMA**

**requester**

**hibi**

0x00

3:

valid channel offset

7:

hibi return address

1:

hibi return address

5:

hibi return address

2:

0x00

4:

hibi return address

6:

valid channel offset

8:

Figure 7‑1 Channel request

Figure and Figure show the message sequence charts for read and write when the channel has already been reserved. On the left, N2H2 denotes the controlling IP (such as Nios, name refers to Nios-to-HIBI 2) and HIBI is the interconnection. In the middle, the HIBI\_MEM\_DMA, then Altera’s DDRx controller, and on the right the physical memory chip.



Figure 7‑2 Read channel configuration and operation once the channel has already been reserved.



Figure 7‑3 Write configuration and operation once the channel has already been reserved.

### Read channel configuration

A read channel is configured by sending 4 or 5 HIBI message write words which act as configuration words. After the last word is received by HIBI\_MEM\_DMA the configuration is done and the DMA adds the read operation to its task list.

Read operation configuration words:

1. Memory source address
2. Byte count
3. HIBI return address
4. Address interval, address increment
5. Address interval increment (optional)

Note that these are all written to the same configuration register (i.e. to the same address) and hence the order of the parameters is crucial. Also notice that the last configuration word is optional and should be configured only if address interval > 0. Table shows additional information on the read configuration parameters.

|  |  |  |
| --- | --- | --- |
| Configuration parameter | Word | Bits |
| Memory source address | 0 | MEM\_ADDR\_WIDTH … 0 |
| Byte count | 1 | RW\_AMOUNT\_WIDTH … 0 |
| HIBI return address | 2 | HIBI\_DATA\_WIDTH … 0 |
| Address increment | 3 | RW\_ADDR\_INC\_WIDTH … 0 |
| Address interval | 3 | RW\_ADDR\_INC\_WIDTH + RW\_ADDR\_INTERVAL\_WIDTH … RW\_ADDR\_INC\_WIDTH |
| Address interval increment | 4 | RW\_ADDR\_INTERVAL\_INC\_WIDTH … 0 |

*Table* *7-1 Read configuration parameters*

Parameter address increment describes the amount of word addresses the address is incremented after each word read. If address interval is > 0 then it describes the amount of words is read between each reading interval. Address interval increment sets the amount of word addresses the address is incremented after each interval.

### Write channel configuration

A write channel is configured by sending 3 or 4 HIBI message write words which act as configuration words. After the last word is received by HIBI\_MEM\_DMA the configuration is done and the DMA adds the write operation to its task list and waits for incoming data.

Write operation configuration words:

1. Memory target address
2. Byte count
3. Address interval, address increment
4. Address interval increment (optional)

Here, one must note that the DMA performs address translation. All the data is written to the same HIBI address and the DMA controller takes care of incrementing the memory address. Therefore the producer IP can use only single write address and this is beneficial when network uses multiplexed data and address lines (as in HIBI) or packet-switching.

The address for the write data is the same as the channel configuration register but the data is sent as normal HIBI write words as opposed to message write words. After the specified amount of words has arrived to the write channel the channel is freed and it can be requested again.

Table shows additional information on the write configuration parameters.

|  |  |  |
| --- | --- | --- |
| Configuration parameter | Word | Bits |
| Memory target address | 0 | MEM\_ADDR\_WIDTH … 0 |
| Byte count | 1 | RW\_AMOUNT\_WIDTH … 0 |
| Address increment | 2 | RW\_ADDR\_INC\_WIDTH … 0 |
| Address interval | 2 | RW\_ADDR\_INC\_WIDTH + RW\_ADDR\_INTERVAL\_WIDTH … RW\_ADDR\_INC\_WIDTH |
| Address interval increment | 3 | RW\_ADDR\_INTERVAL\_INC\_WIDTH … 0 |

*Table* *7-2 Write configuration parameters*

Additional parameter descriptions can be found below table .

# Use examples

Figure shows a conventional use example which has a Sopc sub-system with a NIOS II cpu connected to a Avalon bus. The NIOS II cpu has access to a N2H2 adapter component which connects the Avalon bus to a HIBI bus. The HIBI bus is connected to various components including a HIBI\_MEM\_DMA component connected to a DDR2 memory.



Figure 8‑ Use example

# Performance and resource usage

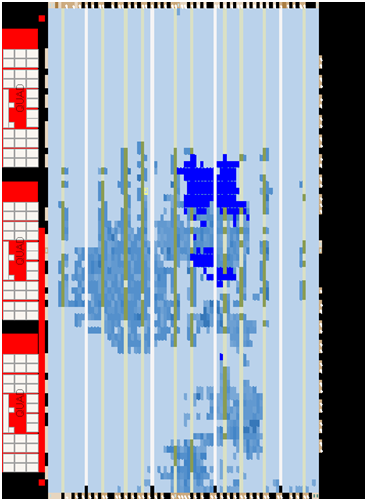
HIBI\_MEM\_DMA’s performance was measured using a Arria II GX FPGA board. The board uses a Arria II GX FPGA chip with about 125000 luts. The board also has several different types of memories but the 64-bit wide DDR2 mounted in the DIMM socket was used for the measures.

Figure shows the resource usage on the FPGA chip viewed in the Quartus Chip Planner. Below is some additional information on the HDL files used and the resource usage.

Figure 9‑ HIBI\_MEM\_DMA resource usage (screen capture from Quartus Chip Planner)

HIBI MEM DMA

other logic



Arria II GX fpga

Design files:

* hibi\_mem\_dma.vhd
  + main design file
  + 1634 rows of commented VHDL
* on\_chip\_ram\_u.vhd
  + generic 1-2 port memory component
  + memory is inferred into on-chip ram blocks during synthesis
  + tested on an Altera fpga but should also work on Xilinx devices

Resource usage:

* Arria II GX fpga, 128 write and 128 read channels
  + 1254 combinational ALUTs
  + 1268 dedicated logic registers
  + 12896 memory bits / 19 M9K ram blocks

HIBI\_MEM\_DMA’s write performance was measured to be maxed out with a transfer length of about 19 words. Read performance had a less steeper curve but both the write and read performance was maxed out at 400MB/s. Theoretical maximum troughput for a 32-bit bus running at 200Mhz is 800MB/s so the measured performance was roughly half of the theoretical maximum. This performance loss was mainly due to HIBI\_MEM\_DMA not using Altera DDRx controller’s burst functionality efficiently.

Performance:

* Theoretical maximum throughput for a 200MHz 32-bit bus is 800MB/s
* Arria II GX fpga, 200 MHz DDR2, 32-bit memory bus

Figures and show the write and read performance as a function of the transfer length.



Figure *9-2 Read/write performance for different transfer lengths*



Figure *9-3 Read/write performance for short transfer lengths*

# IP-block files

## Documentation

|  |  |  |
| --- | --- | --- |
| Directory | File | Description |
| doc\ | HIBI\_MEM\_DMA.docx | ip-block datasheet (this file) |
| doc\fig\ | HIBI\_MEM\_DMA.pptx | Powerpoint figures for the datasheet |
| HIBI\_MEM\_DMA.mdzip | Magicdraw flow charts for the datasheet |
| HIBI\_MEM\_DMA.mdr | Magicdraw flow charts for the datasheet |

*Table 8-1 Documentation files*

## Main component files

|  |  |  |
| --- | --- | --- |
| Directory | File | Description |
| hdl\ | hibi\_mem\_dma.vhd | main hdl file |
| onchip\_ram\_u.vhd | hdl file for internal memory |
| dual\_port\_ram\_u.vhd | hdl file for internal memory |
| dual\_ram\_async\_read.vhd | hdl file for internal memory |
| fifo\_ram.vhd | hdl file for a fifo |
| fifo\_u.vhd | hdl file for a fifo |

*Table 8-2 Files for HIBI\_MEM\_DMA*

## Verification files

|  |  |  |
| --- | --- | --- |
| Directory | File | Description |
| tb\ | makefile | debug printing |
|  | hibi\_mem\_dma\_perf\_test.do |  |
|  | hibi\_mem\_dma\_wave.do |  |
|  | hibi\_mem\_dma\_perf\_test\_tb.vhd | top level testbench file for both synthesis and simulation |
|  | hibi\_mem\_dma\_tester.vhd | tester component for both synthesis and simulation |
|  | hibi\_seg.vhd |  |
|  | alt\_mem.vhd |  |

*Table 8-3 Files for HIBI\_MEM\_DMA verification*

# Dependent components

HIBI\_MEM\_DMA is dependent on Altera High Performance II DDRx controller (generated with Megaqizard) and HIBI bus. table lists the files required for a Altera DDRx controller. Required files for the HIBI segment and it’s wrappers can be found on the HIBI component documentation.

|  |  |  |
| --- | --- | --- |
| Directory | File | Description |
| “alt\_ddrx”\hdl\ | “alt\_ddrx”.v | top level file for Altera DDRx |
| alt\_mem\_phy\_defines.v | verilog include file |
| “alt\_ddrx”\_controller\_phy.v |  |
| “alt\_ddrx”\_phy.v |  |
| “alt\_ddrx”\_phy\_alt\_mem\_phy.v |  |
| “alt\_ddrx”\_phy\_alt\_mem\_phy\_pll.v |  |
| “alt\_ddrx”\_phy\_alt\_mem\_phy\_dq\_dqs.v |  |
| “alt\_ddrx”\_phy\_alt\_mem\_phy\_seq.vhd |  |
| “alt\_ddrx”\_phy\_alt\_mem\_phy\_seq\_wrapper.v |  |
| alt\_ddrx\_addr\_cmd.v |  |
| alt\_ddrx\_afi\_block.v |  |
| alt\_ddrx\_avalon\_if.v |  |
| alt\_ddrx\_bank\_timer.v |  |
| alt\_ddrx\_bank\_timer\_info.v |  |
| alt\_ddrx\_bank\_timer\_wrapper.v |  |
| alt\_ddrx\_bank\_tracking.v |  |
| alt\_ddrx\_bypass.v |  |
| alt\_ddrx\_cache.v |  |
| alt\_ddrx\_clock\_and\_reset.v |  |
| alt\_ddrx\_cmd\_gen.v |  |
| alt\_ddrx\_cmd\_queue.v |  |
| alt\_ddrx\_controller.v |  |
| alt\_ddrx\_csr.v |  |
| alt\_ddrx\_ddr2\_odt\_gen.v |  |
| alt\_ddrx\_ddr3\_odt\_gen.v |  |
| alt\_ddrx\_decoder.v |  |
| alt\_ddrx\_decoder\_40.v |  |
| alt\_ddrx\_decoder\_72.v |  |
| alt\_ddrx\_ecc.v |  |
| alt\_ddrx\_encoder.v |  |
| alt\_ddrx\_encoder\_40.v |  |
| alt\_ddrx\_encoder\_72.v |  |
| alt\_ddrx\_input\_if.v |  |
| alt\_ddrx\_odt\_gen.v |  |
| alt\_ddrx\_rank\_monitor.v |  |
| alt\_ddrx\_state\_machine.v |  |
| alt\_ddrx\_timers.v |  |
| alt\_ddrx\_timers\_fsm.v |  |
| alt\_ddrx\_timing\_param.v |  |
| alt\_ddrx\_wdata\_fifo.v |  |
| altera\_avalon\_half\_rate\_bridge.v |  |

*Table* *9-1 Files for Altera DDRx controller*

|  |  |  |
| --- | --- | --- |
| Directory | File | Description |
| “alt\_ddrx”\tb\ | “alt\_ddrx”\_full\_mem\_model.v | DDRx simulation memory model |

*Table 9-2 Files for DDRx simulation memory model*

# Verification

Verification of the controller has been done by simulation on ModelSim and physical testing on a Arria II GX development board.

# Detailed specification for developers

An adapter component with DMA functionality to interface memory controllers to Hibi network. Only Altera’s DDR memory controllers created with the Megawizard are supported at the moment.



## Address demux

Demuxes the incoming data to either the config module, one of the channel modules or to the direct access modules. Config and channel modules address range is one while the direct access modules can have an address range of 1 MB for example.

## Config. module

Configuration module is used to configure the other blocks.

## Memory access arbiter

Memory access arbiter decides which module it gives access to the Altera memory controller. Possible arbitration modes are round-robin, fixed priority and variable priority.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| b96...b93 | b92 | b91...b78 | b77...b64 | b63...b50 | b49...b30 | b29...b00 |
| byte enable | configuration done | address interval increment | address interval | address increment amount | current write amount | current mem. address |

Table 11- Write configure memory

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| b124...b93 | b92 | b91...b78 | b77...b64 | b63...b50 | b49...b30 | b29...b00 |
| HIBI return address | configuration done | address interval increment | address interval | address increment amount | current read amount | current mem. address |

*Table 11-2 Read configure memory*

## Internal structure and functionality



The functionality of the memory to HIBI r2 adapter is divided mainly to five components. These include the HIBI message fifo reader, request processer, memory access processer, HIBI data writer and the internal memory used by these components.

The HIBI message fifo reader has exclusive access to the HIBI read message fifo and it contains in itself two different components. The address demux simply sends the incoming messages to the correct component based on the address of the message. Messages containing write or read requests are sent to the request processer and the configuration for these writes or reads are sent to the configuration prosesser. The configuration processer updates read/write configuration state memory and the write and read configuration memories as it receives the configuration words.

Write and read requests are handled by the request processer which has access to the HIBI message write fifo.

## Adding a custom Altera memory controller

Memory controllers for the Stratix III development board will be supplied but it’s possible to add any Altera memory controller to use with this adapter by creating it in the Altera’s Megawizard.