Dini Buses User FPGA Design User Manual

5/12/2011

Rev 1.3

Table of Contents

Table of Tables

Table of Figures

Table 1 – Change Log

Revision	Name	Details
1.1	Brian Poladian	Initial
1.2	Brian Poladian	Added detail for target/dma accesses
1.3	Brian Poladian	Added TSK_DMA_READ/TSK_DMA_WAIT_FINISH usage.

1 Overview

This document describes the interfaces, signals, and procedures necessary to understand and simulate the Dinigroup FPGA Reference Design that interfaces to the backend of the Dini Group PCIe-DMA ConfigFPGA Design. There are several different physical bus connections that share this interface module including NMB, SFB, and DDR-PCIe. All of these are discussed here.

This document is not pertinent to the PCIe8T family of boards; please see http://www.dinigroup.com/product/common/pcie8t_user_interface_manual.pdf if working with a DN9000K10PCIe8T, DN9002K10PCIe8T, DN9200K10PCIe8T, DN7006K10PCIe8T, DN7406K10PCIE8T, or DNMEG_V5T_PCIE.

The Dinigroup PCIe-DMA design includes BAR memory access and DMA engines. Using various physical bus interfaces, the reference design connects the user's design in the field FPGA to the ConfigFPGA. If you have questions/comments/concerns, please email support@dinigroup.com .

2 Verilog User Interface

Here we discuss the signals in and out of the pcie_interface module. It is recommended that the user instantiate this module to interface with the configFPGA. This module will connect to either a PCIe DDR, NMB, or SFB IO module, which will convert the physical interface to the internal interface described here. Direction of ports described is in relation to the pcie_interface module.

Table 2 – Backend Clocks, Resets

Signal Name	Direction	Description
reset	Input	Main module reset. Input from user. Allows resetting the module from an external source.
reset_out	Output	Reset to user module. When de-asserted this indicates that the interface is ready to use.
user_clk	Input	User clock. Clocks logic on the user side of the interface. Can be sourced by clk_out.
clk_out	Output	Clock sent to user. Clocks logic on the physical side of the interface.

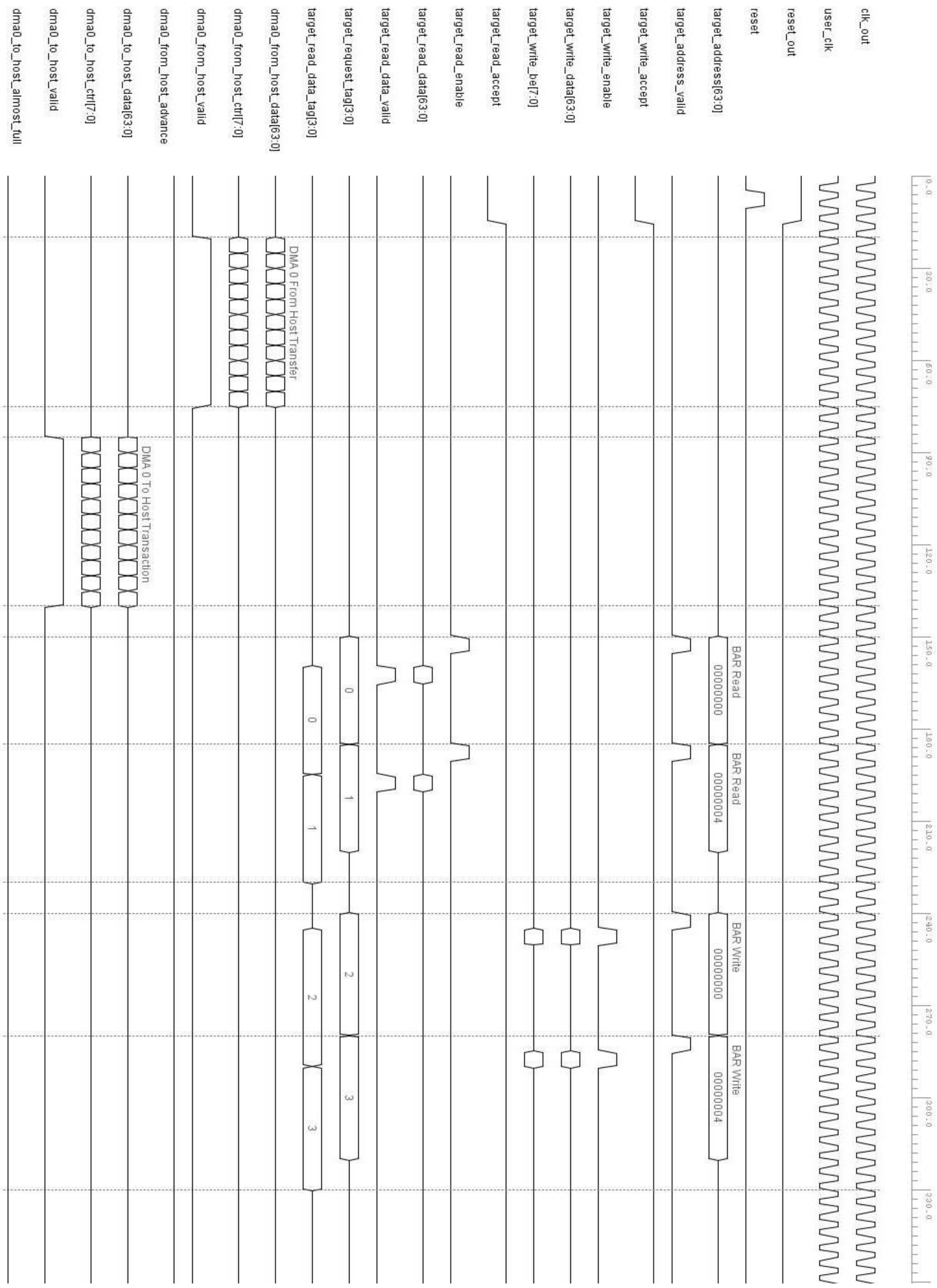
Table 3 – User Signals

Signal Name	Direction	Description
Target Interface		
target_address[63:0]	Output	Byte address (bits 1:0 always 0)
target_address_valid	Output	One clock cycle strobe that indicates the target_address is valid
target_write_data[63:0]	Output	Data.
target_write_be[7:0]	Output	Byte enables, bit 0 means data[7:0] is valid, etc.

target_write_enable	Output	Indicates valid write data on this interface.
target_write_accept	Input	Accepts the valid write data. Data is allowed to transfer when target_write_accept and target_write_enable are active. If connecting to blockRAM, this signal can be tied high.
debug_target_bar[2:0]	Output	Valid for reads and writes. Indicates which bar is being accessed (bar number 1, 2, or 4). Onehot, so 3’b001 means BAR1, 3’b010 means BAR2, and 3’b100 means BAR4. To treat target_address as a unified 64-bit address space, this signals should be used for debug/informational purposes only.
target_read_enable	Output	High on a target read request.
target_request_tag[3:0]	Output	Tag associated with this read request.
target_read_accept	Input	Acceptance of read request. Allows interface to “move on” before the read data is returned by the user. Target_read_enable and target_read_accept being high signals “transfer” of request. If connecting to blockram, this signal can be tied high.
target_read_data[63:0]	Input	Read Data to return to configFPGA.
target_read_be[7:0]	Output	Byte enables, bit 0 means data[7:0] is requested, etc.
target_read_data_tag[3:0]	Input	Tag that accompanies this data. This value must match the value provided on target_request_tag when target_read_enable was high.
target_read_data_valid	Input	Clock cycle pulse indicating read data is valid.
target_read_ctrl[7:0]	Output	Tag associated with a target read. Bit [0] indicates both dwords are requested (non-dword aligned or quadword transfer)
target_read_data_ctrl[7:0]	Input	Tag associated with a target read. User must save target_read_ctrl[7:0] when target_read_enable is asserted and return it on target_read_data_ctrl[7:0]
DMA0 Interface		
dma0_from_host_data[63:0]	Output	Address, length, or data, depending on the state of dma0_from_host_ctrl[5,0].
dma0_from_host_ctrl[7:0]	Output	Bit meanings: 5, 0 indicates type of data. 3:2 are dword enables (when bit 3 is 1, 63:32 is valid, when bit 2 is 1, 31:0 is valid) 4 indicates read request when 1, write “packet” when 0.

		Bits 1, 6, 7 reserved. Decode of 5,0: 01: 64 bit board dword address. 10: 24 bits of dword length [23:0]. Upper 40 bits are reserved. For reads, user must return this many dwords of data, properly aligned based on bit 0 of the dword board address. For writes this information is provided but informational only. 00: Data 11: Reserved/Undefined/Never Occurs.
dma0_from_host_valid	Output	Indicates valid data on data and ctrl signals.
dma0_from_host_advance	Input	This signal high and dma0_from_host_valid high indicates data "transfer".
dma0_to_host_data[63:0]	Input	Data to return to PCI-E FPGA.
dma0_to_host_ctrl[7:0]	Input	Bit meanings: 1:0 are dword enables (when bit 1 is 1, 63:32 is valid data, when bit 0 is 1, 31:0 is valid data). 3: Indicates last data for this read request. Bits 2, 4-7 are reserved.
dma0_to_host_valid	Input	When 1, data/ctrl is transferred into the module.
dma0_to_host_almost_full	Output	When high, user should stop writing data to the module soon.
DMA1 Interface		
Same as DMA0	Same	Same as DMA0, but '1' in the name instead of '0'. Separate interfaces for each DMA engine.
DMA2 Interface		
Same as DMA0	Same	Same as DMA0, but '2' in the name instead of '0'. Separate interfaces for each DMA engine.
User Interrupts		
user_interrupts [?:0]	Input	A parameterizable number of user-level interrupts, which will be combined into a single interrupt and sent to the PCIe core. Interrupts must be asserted until cleared by configFPGA.

Figure 1 – Example User Transactions



2.1 Target Interface Detail

2.1.1 Target Writes

- target_write_accept must be asserted for the transaction to begin.
- target_address_valid will assert when target_address is valid.
- target_write_enable will assert when target_write_data and target_write_be are valid.

2.1.2 Target Reads

- target_read_accept must be asserted for the transaction to begin.
- target_address_valid and target_read_enable will pulse when target_address, target_read_be, target_request_tag, and target_read_ctrl are valid.
- target_read_data_valid should be assered when target_read_data, target_read_data_tag, and target_read_data_ctrl are valid.

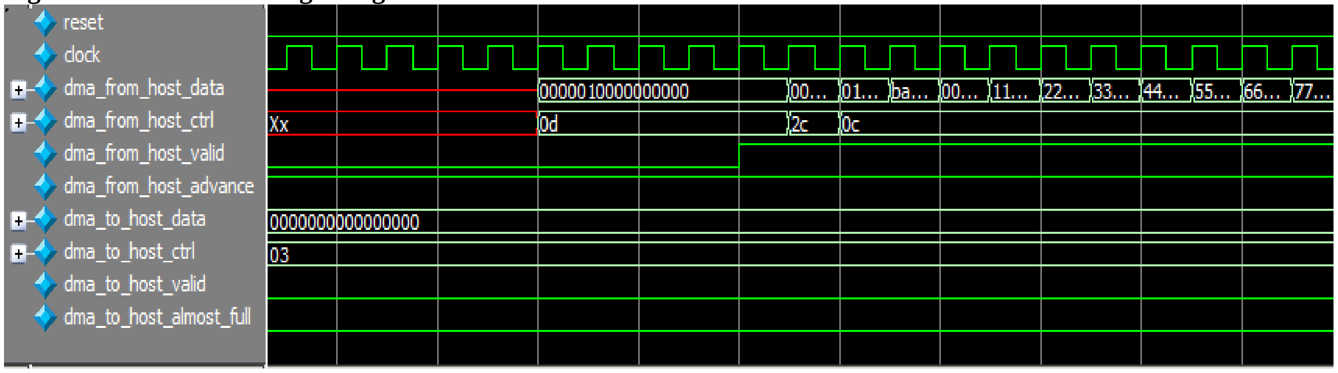
2.1.3 Gotchas

- Target reads must complete within 4K clock cycles. The host processor will stall while the read is outstanding, and to prevent a permanent system hang, the configFPGA will return a timeout value after 4K clock cycles.
- The exact amount of data requested should be returned. Returning too much or too little data will result in unexpected/unsupported behavior.
- The common use for the target interface is a register interface that can always accept accesses; it is recommended that target_write_accept and target_read_accept always be asserted.
- Only one target read is issued at a time, and the host processor will hang while it waits for a response – it is encouraged to use this interface for low-latency accesses.
- target_request_tag and target_read_ctrl should be returned as target_read_request_tag and target_read_data_ctrl.

2.2 DMA Interface Detail

2.2.1 DMA Writes

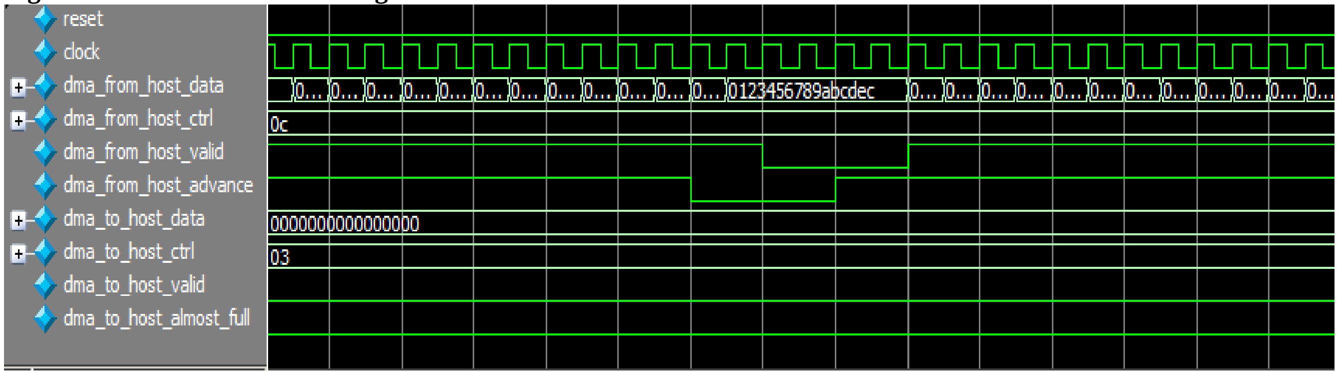
Figure 2 – DMA Write Beginning



- dma_from_host_valid is asserted at start of transfer, and remains asserted while transfer data/control is active.
- 64-bit address is transferred on first valid cycle (dma_from_host_ctrl[5,0]=2'b01)
- dma_from_host_ctrl[4] should be 0 to indicate write.
- Length data is transferred on second valid cycle (dma_from_host_ctrl[5,0]=2'b10).

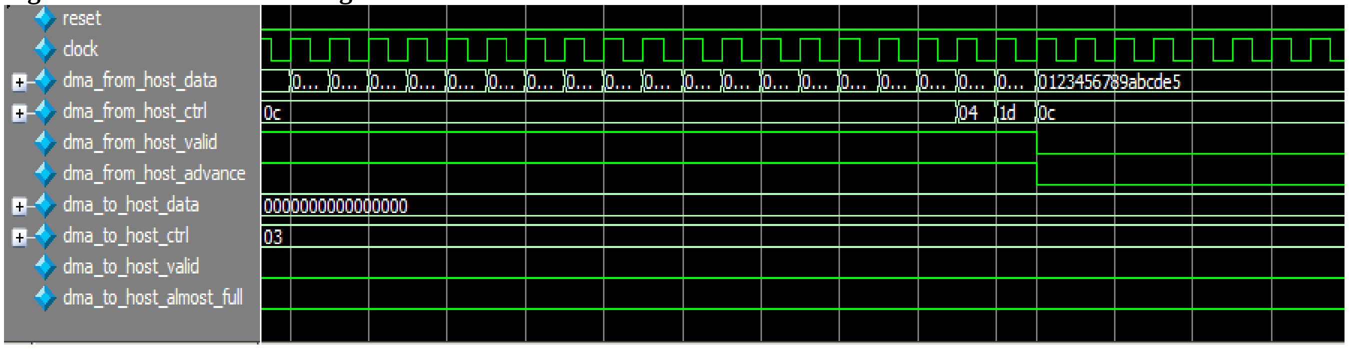
- Data is transferred from third valid cycle onward.

Figure 3 – DMA Write Throttling



- dma_from_host_advance allows the write to continue; deasserting this signal prevents write data from advancing.

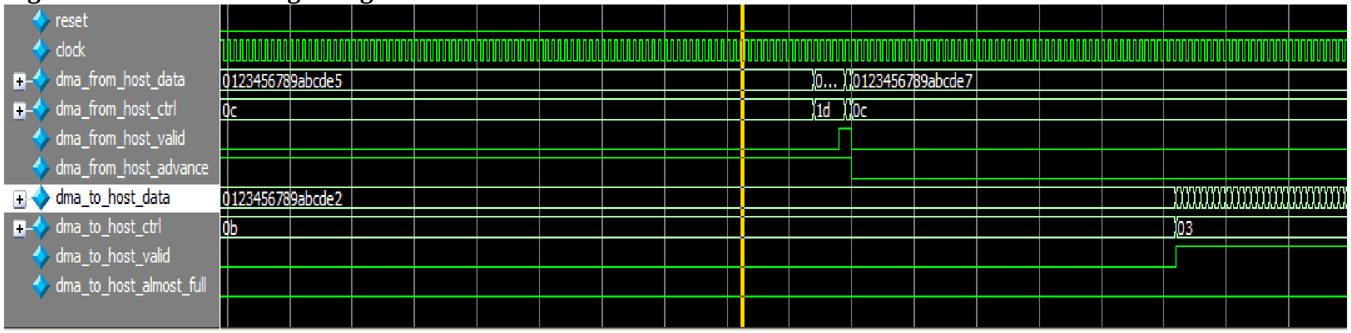
Figure 4 – DMA Write Ending



- The write transaction is considered officially over at the stat of the next transaction (read or write, read pictured here); otherwise the write may continue when dma_from_host_valid is reasserted.
- Note that the last cycle of data may have a different dword enable value (pictured above) depending on the length of data to be transferred.

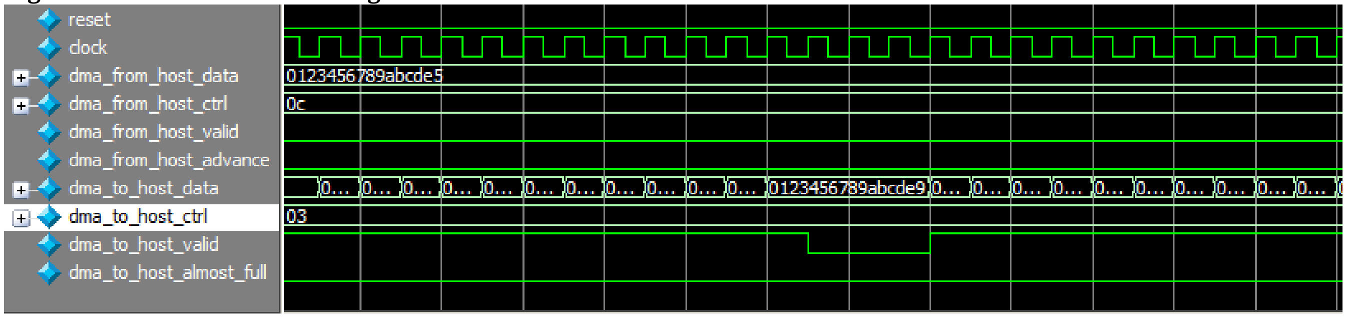
2.2.2 DMA Reads

Figure 5 - DMA Read Beginning



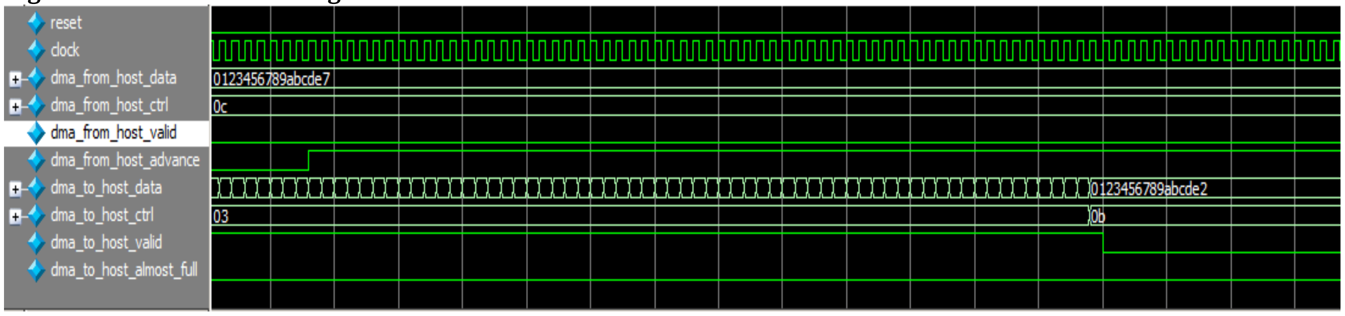
- dma_from_host_valid is asserted at start of transfer, and remains asserted while transfer data/control is active.
- 64-bit address is transferred on first valid cycle (dma_from_host_ctrl[5,0]=2'b01).
- dma_from_host_ctrl[4] should be 0 to indicate write.
- Length data is transferred on second valid cycle (dma_from_host_ctrl[5,0]=2'b10).
- Return data will begin some time later – there is no latency requirement on reads.
- dma_to_host_valid will be asserted when read return data is valid.

Figure 6 – DMA Read Throttling



- dma_to_host_valid shoud be deasserted when read return data is invalid; return data does not need to be continuous.
- If dma_to_host_almost_full is asserted, read return data should not advance and dma_to_host_valid should be deasserted.

Figure 7 - DMA Read Ending



- dma_from_host_valid is deasserted after last valid data.
- dma_to_host_ctrl[3] should be asserted on last valid data.

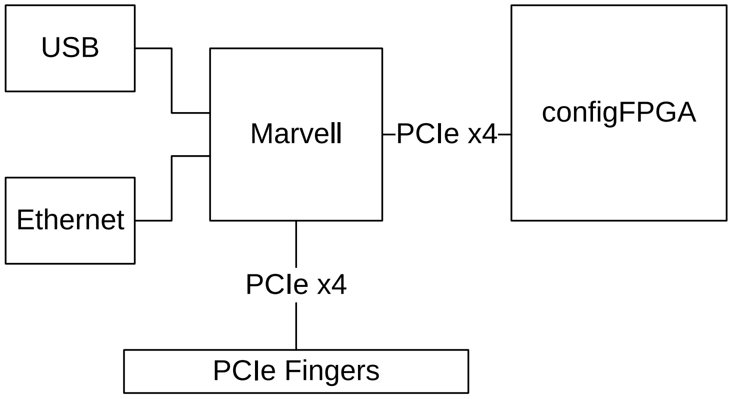
2.2.3 Gotchas

- Allowable latencies for throttling are dependent on the depth of the FIFOs used in the design and the actual latencies are dependent on the pipeline length of signals. Ideally, the response to deasserting from_host_advance or asserting dma_to_host_almost_full would be immediate and would thus require a smaller FIFO depth. Longer pipeline logic requires deeper FIFOs.
- dma_from_host_advance needs to be asserted for a DMA transfer to start.
- DMA reads should return the exact amount of data requested.

3 ConfigFPGA PCIe Interface

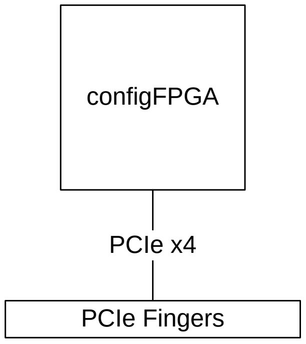
On Marvell-based Dinigroup boards, the configFPGA connects via PCIe to the Marvell; the host computer is not able to directly see the configFPGA's DMA engines or register space. The software/driver interface to the configFPGA runs on the Marvell, and a separate set of software/drivers runs on the host PC to communicate with the Marvell (via PCIe, USB, or Ethernet). For more information, please see the EMU and DiniCMOS documentation.

Figure 8 – Marvell-based Board Configuration



On non-Marvell boards (and additionally on Marvell-based boards with alternate configuration options), the configFPGA is connected directly to the PCIe fingers, and its BAR space and DMA engines are directly accessible from the host PC. For more information, please see the AETest documentation.

Figure 9 – Non-Marvell-based Board Configuration



3.1 ConfigFPGA Simulation

The manner in which the configFPGA connects to the field (user) FPGAs varies depending on the specific Dinigroup board. Each physical interface has a supplied IO module that connects to the configFPGA across a common bus interface. The common bus interface is described below:

Table 4 – PCIE tohost/fromhost Interface

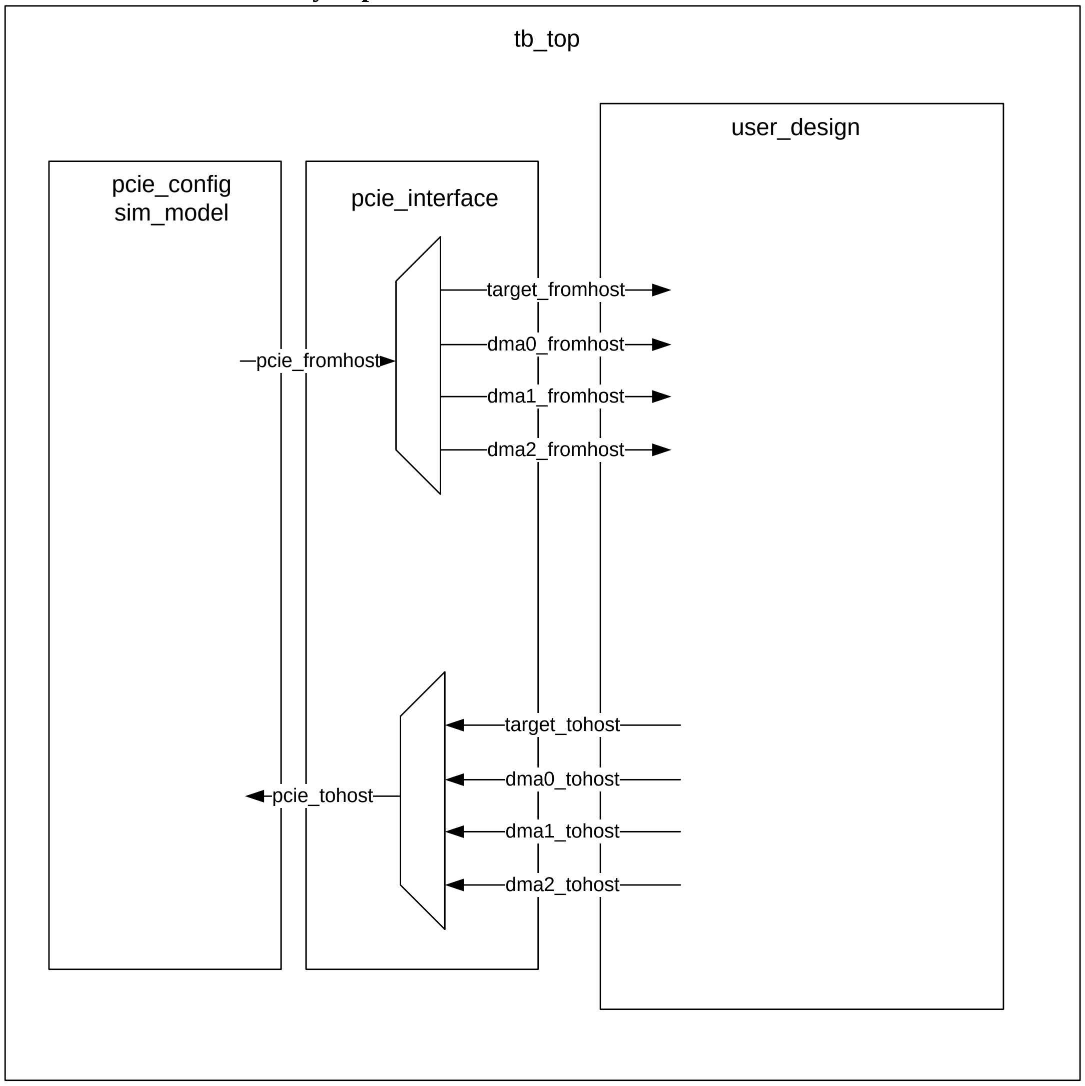
Signal Name	Direction	Description
clk	Output	Main clock
clk_2x	Output	2x main clock.
clk_5x	Output	5x main clock; serial clock on NMB.
reset	Output	Reset supplied to user design.
pcie_fromhost_data[63:0]	Output	Muxed target/DMA address/control/data
pcie_fromhost_transaction_type[3:0]	Output	[0] = Target transaction [1] = DMA0 transaction [2] = DMA1 transaction [3] = DMA2 transaction
pcie_fromhost_isaddress[1:0]	Output	[0] = Address / start of transaction [1] = end of transaction
pcie_fromhost_info[1:0]	Output	[0] = Read transaction / write_n [1] = 64-bit target access –OR- DMA length data
pcie_fromhost_valid[1:0]	Output	[0] = data[31:0] valid [1] = data[63:32] valid
pcie_fromhost_almost_full[4:0]	Output	[0] = Target interface almost full [1] = DMA0 interface almost full [2] = DMA1 interface almost full [3] = DMA2 interface almost full
pcie_tohost_data[63:0]	Input	Muxed target/DMA data
pcie_tohost_transaction_type[3:0]	Input	[0] = Target transaction [1] = DMA0 transaction [2] = DMA1 transaction [3] = DMA2 transaction
pcie_tohost_info[4:0]	Input	When Target transaction: [0] = 64-bit access [4:1] = Target return tag

		When DMA transaction: [0] = data[31:0] valid [1] = data[63:32] valid [2] = DMA demand mode [3] = DMA EOF
pcie_tohost_valid[1:0]	Input	[0] = data[31:0] valid [1] = data[63:32] valid
pcie_tohost_almost_full[4:0]	Input	[0] = Target interface almost full [1] = DMA0 interface almost full [2] = DMA1 interface almost full [3] = DMA2 interface almost full

This internal interface is subject to change as new features are added, and the signal definitions listed here should not be considered static.

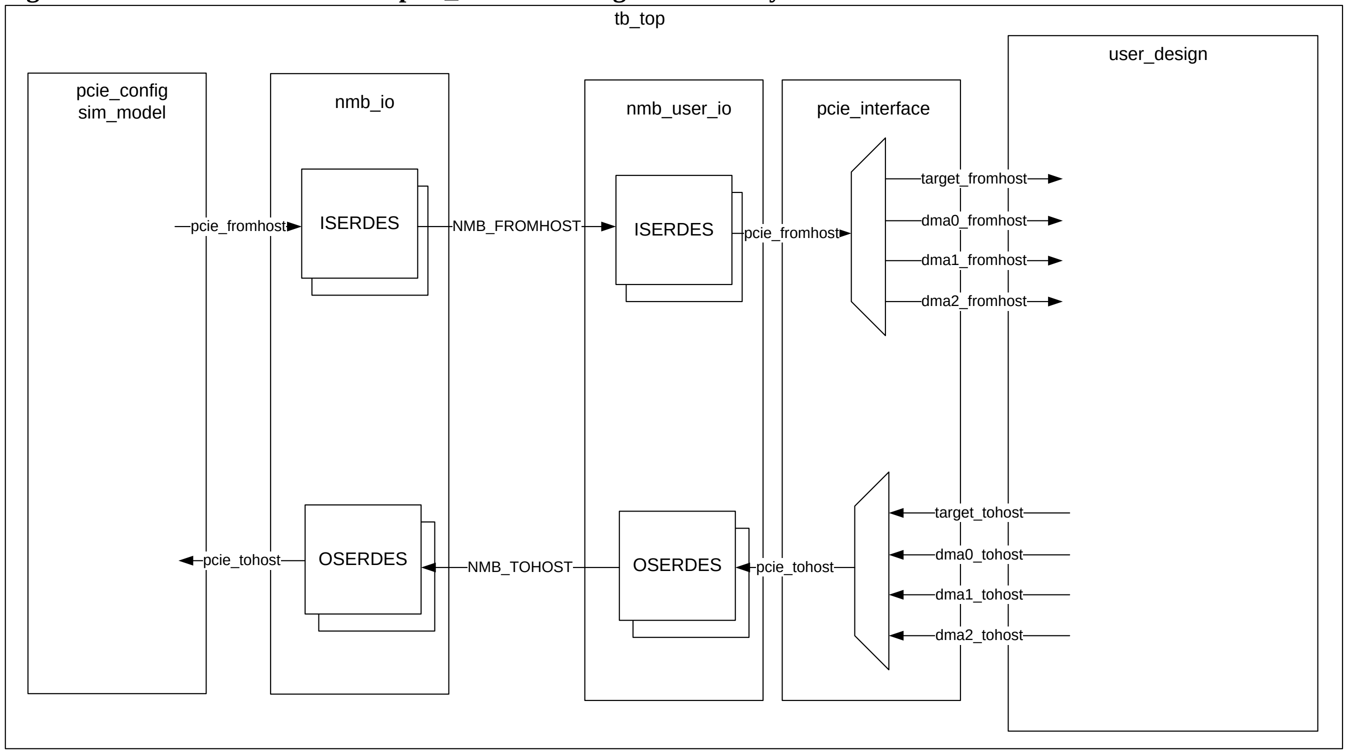
It is NOT recommended to use this interface directly. This interface is intended to connect directly to the pcie_interface module, which will translate the tohost/fromhost interface into the recommended user interface, with separate ports for target, DMA0, DMA1, and DMA2, as described in section 2.

Figure 10 – Connect Sim Model directly to pcie_interface



If the user desires to include the physical interface in simulation, then the IO modules will be placed between the pcie_config_sim_model and pcie_interface modules. The pcie tohost/fromhost bus will not be modified by the physical IO modules.

Figure 11 – Connect Sim Model to pcie_interface using intermediary IO modules

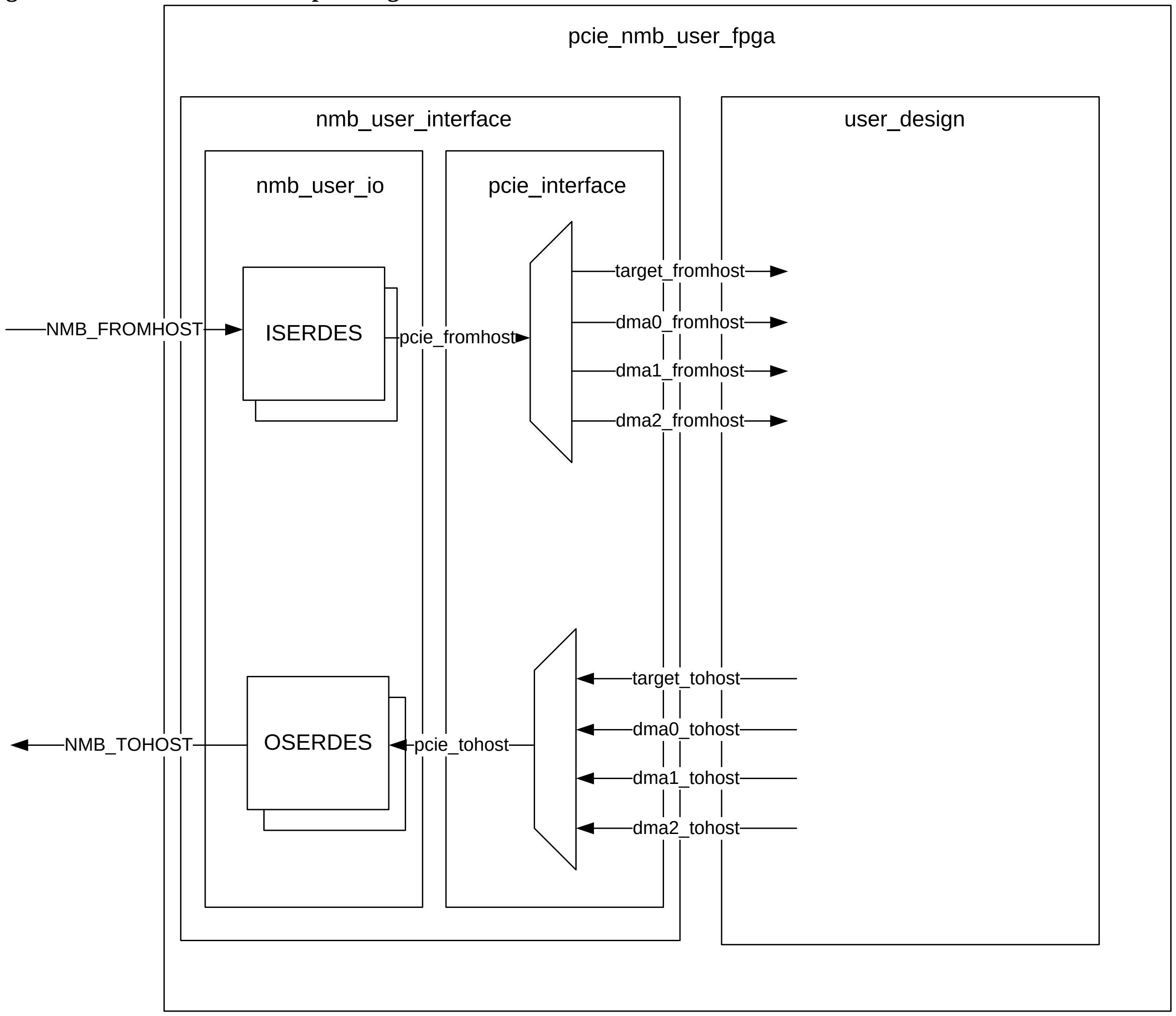


4 User Interface Modules

4.1 IO Modules/Interfaces

Each connection between the configFPGA and a field FPGA uses some type of physical interface, eg NMB, DDR-PCIe, SFB, etc. Each one of those physical interfaces has a related set of IO modules, one for the configFPGA and one for the user FPGA. A wrapper is provided for the user IO module and the pcie_interface module so that the user can have a single module that translates between the physical signals and the recommended Target/DMA interface. An example top-level design is also provided showing how to connect a user design to the interface module.

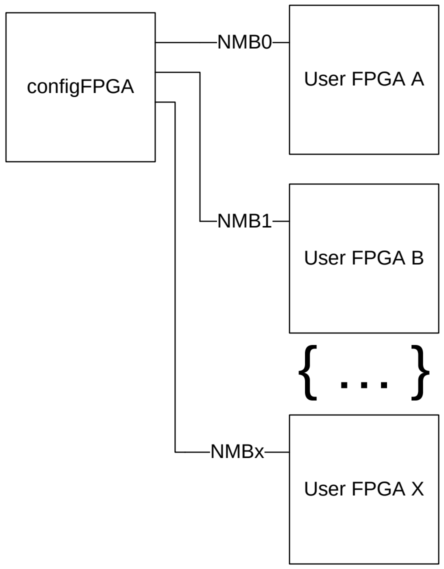
Figure 12 – Detail of NMB example design



4.2 NMB Interface

The most common bus architecture for the Marvell-based boards uses the NMB bus. NMB is a high-speed LVDS SERDES point-to-point bus for communication between the configFPGA and the user FPGAs. NMB is not shared with configuration signals. NMB uses 10 signals (8-bit data + 1-bit control + 1 source-synchronous clock) in each direction with a maximum speed of 1 Gbps per signal, for a total data throughput of 8Gbps in each direction (full-duplex). Each user FPGA has its own independent NMB bus.

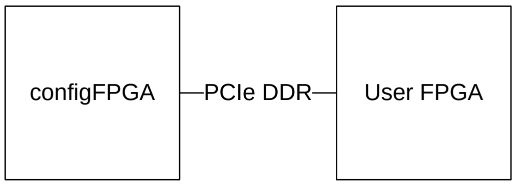
Figure 13 – NMB Bus



4.3 DDR-PCIe Interface

The PCIe DDR bus is a 64-bit data + 13-bit control + source-synchronous clock DDR bus. Because of the high pin count, there is usually only one of these busses attached to the configFPGA, and it is commonly combined with SFB interfaces on Dinigroup boards. The PCIe DDR bus will run ~100Mhz, for a total data throughput of 6.4 Gbps in each direction (full-duplex).

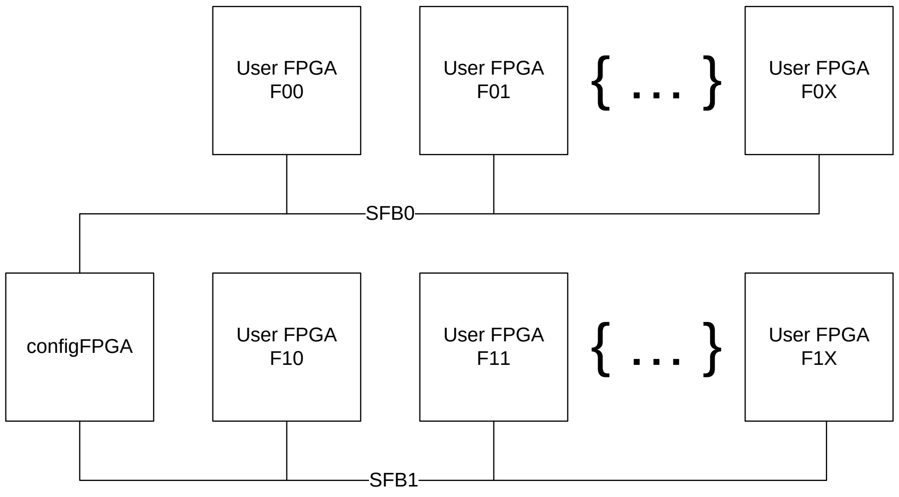
Figure 14 – PCIe DDR Bus



4.4 SFB Interface

The SFB bus is a 8-bit data + 4-bit control + global clock bus that shares its signals with the SelectMAP configuration bus. Because of its low pin count, it is ideal to combine with the PCIe DDR bus on Dinigroup boards. Multiple FPGAs can be connected to a single SFB bus, but communication is half-duplex, and so only one FPGA may communicate on the bus at a time. The SFB bus will run ~60MHz, for a total data throughput of 480 Mbps.

Figure 15 – SFB Bus



5 Description/Location of Files

All file locations are relative from the FPGA_Reference_Designs folder in the board support package.

5.1 Files for configFPGA

common/pcie/pcie_dma/pcie_fpga/testbench/pcie_config_sim_model.v

Simulation model of the configFPGA design. Tasks are used for stimulating the user interfaces; see section 5.3 for further description. The testbench is configurable to connect to a user-specified number of interfaces.

common/pcie/pcie_dma/pcie_fpga/testbench/pcie_board.v

Board-level testbench connections between configFPGA, appropriate IO modules, and the user design.

common/pcie/pcie_dma/pcie_fpga/testbench/tb_top.v

Top-level testbench that instantiates the board-level testbench and calls the Target/DMA tasks.

common/pcie/pcie_dma/pcie_fpga/testbench/bar_tests.v

Example list of Target interface tests to run.

common/pcie/pcie_dma/pcie_fpga/testbench/dma_tests.v

Example list of DMA interface tests to run.

common/NMB/nmb_io.v

NMB IO module; connects to nmb_user_io/nmb_user_interface modules.

common/SuperFastBus/sfb_io.v
SFB IO module; connects to sfb_user_io/sfb_user_interface modules

common/pcie/pcie_dma/user_fpga/pcie_ddr_io.v
PCIE DDR IO module; connects to pcie_ddr_user_io/pcie_ddr_user_interface modules.

5.2 Files for User FPGA

5.2.1 Common

common/pcie/pcie_dma/user_fpga/pcie_interface.v
Translates between pcie tohost/fromhost interface and recommended Target/DMA interface.

common/pcie/pcie_dma/user_fpga/pcie_user_design.v
BlockRAM attached to Target and DMA interfaces. User should use transaction logic from this design.

common/pcie/pcie_dma/user_fpga/pcie_dma_blockram.v
Condensed version of pcie_user_design that attaches BlockRAM to a single DMA engine.

common/pcie/pcie_dma/user_fpga/pcie_defines.v
Misc. defines used in the pcie_user_design module.

5.2.2 NMB

common/pcie/pcie_dma/user_fpga/pcie_nmb_user_fpga.v
FPGA-level example design that connects the nmb_user_interface and pcie_user_design modules.

common/NMB/nmb_user_interface.v
Wrapper for the nmb_user_io and pcie_interface modules.

common/NMB/nmb_user_io.v
Translates physical NMB IO signals to internal tohost/fromhost interface.

5.2.3 SFB

common/pcie/pcie_dma/user_fpga/pcie_sfb_user_fpga.v
FPGA-level example design that connects the sfb_user_interface and pcie_user_design modules.

common/SuperFastBus/sfb_user_interface.v
Wrapper for the sfb_user_io and pcie_interface modules.

common/SuperFastBus/sfb_user_io.v
Translates physical SFB IO signals to internal tohost/fromhost interface.

5.2.4 PCIe DDR

common/pcie/pcie_dma/user_fpga/pcie_ddr_user_fpga.v
FPGA-level example design that connects the pcie_ddr_user_interface and pcie_user_design modules.

common/pcie/pcie_dma/user_fpga/pcie_ddr_user_interface.v
Wrapper for the pcie_ddr_user_io and pcie_interface modules.

common/pcie/pcie_dma/user_fpga/pcie_ddr_user_io.v
Translates physical PCIe DDR IO signals to internal tohost/fromhost interface.

5.3 Simulating the Design

See section 3 for the connections necessary to attach the configFPGA simulation model to the user design. After the connections are established, the tasks in the configFPGA simulation model are used to initiate transactions to the user design.

TSK_TARGET_WRITE : 32- or 64-bit write to the target interface

input [63:0] address	: address to write to
input [31:0] bar	: BAR to simulate direct PCIe access – set to 0x1 if only simulating NMB
input [63:0] data	: data to write
input [1:0] dword_enable	: enable for write data

TSK_TARGET_READ : 64-bit read from target interface

input [63:0] address	: address to read from
input [31:0] bar	: BAR to simulate direct PCIe access – set to 0x1 if only simulating NMB
output [63:0] data	: read data returned

TSK_DMA_WRITE : Write data from file to DMA interface

input [31:0] engine	: 0x0, 0x1, 0x2; DMA engine to use
input [63:0] address	: starting address of transfer
input [255:0] file_name	: source file for DMA transfer data

TSK_DMA_READ : Read data from a DMA interface

input [31:0] engine	: 0x0, 0x1, 0x2; DMA engine to use
input [63:0] address	: starting address of transfer
input [23:0] length	: length of DMA read (num dwords)

TSK_DMA_WAIT_FINISH : Wait for all DMA transfers to finish

Note: To properly simulate hardware behavior, only one DMA read may be issued to a DMA engine at a time. *TSK_DMA_READ* will not wait for all DMA read data to return; *TSK_DMA_WAIT_FINISH* needs to be called between successive DMA reads to a given engine.

TSK_DEMANDMODE_DMA : Set up the reference design to initiate DMA transfers

input [31:0]	engine	: 0x0, 0x1, 0x2; DMA engine to use
input [7:0]	dma_interface	: 0x0 – 0xFF; Upper byte of board address
input [63:0]	address	: host address for DMA transfer
input [31:0]	tag	: tag ID for DMA transfer
input [4:0]	first_be	: first word byte enable
input [4:0]	last_be	: last word byte enable
input [23:0]	size	: length of DMA transfer
input	direction_tohost	: DMA direction; 0x0=fromhost, 0x1=tohost

Note: *TSK_DEMANDMODE_DMA* requires the use of the full reference design provided in *pcie_user_design.v*. Users implementing their own logic to initiate DMA transfers will need to modify this function accordingly.

6 Other Features

If you have a new feature request or feel that an essential function is missing, please contact support@dinigroup.com .