
Speedster22i DDR3 User Guide (UG031)

Speedster FPGAs



Copyrights, Trademarks and Disclaimers

Copyright © 2018 Achronix Semiconductor Corporation. All rights reserved. Achronix, Speedcore, Speedster, and ACE are trademarks of Achronix Semiconductor Corporation in the U.S. and/or other countries. All other trademarks are the property of their respective owners. All specifications subject to change without notice.

NOTICE of DISCLAIMER: The information given in this document is believed to be accurate and reliable. However, Achronix Semiconductor Corporation does not give any representations or warranties as to the completeness or accuracy of such information and shall have no liability for the use of the information contained herein. Achronix Semiconductor Corporation reserves the right to make changes to this document and the information contained herein at any time and without notice. All Achronix trademarks, registered trademarks, disclaimers and patents are listed at <http://www.achronix.com/legal>.

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

Table of Contents

Chapter - 1: Introduction	7
Features	7
Functional Overview	8
Chapter - 2: DDR3 Macro Interfaces	10
Internal (Core-side) Interface	10
External (Memory) Interface	13
DDR3 Macro Block Diagram	15
Chapter - 3: DDR3 Core Parameters	16
Chapter - 4: Read and Write Operations	21
Read Interface Details	21
Read Protocol 2× Clock Mode	21
Back-to-Back Read Protocol 2× Clock Mode	23
Write Interface Details	25
Write Protocol 2× Clock Mode	25
Back-to-Back Write Protocol 2× Clock Mode	29
Chapter - 5: Address Mapping	32
Chapter - 6: Board Design and Layout	34
Routing Guidelines and Trace Matching	34
Board-Level Simulation	34
DDR3 Routing Guidelines	34
High-Level Rules	35
Guidelines	37
Stack-up	45
Chapter - 7: PHY Architecture	51
Overview	51
Clock and Reset Architectures	51
Overview	51
Clocks	51
Resets	53
Example Use Cases	54

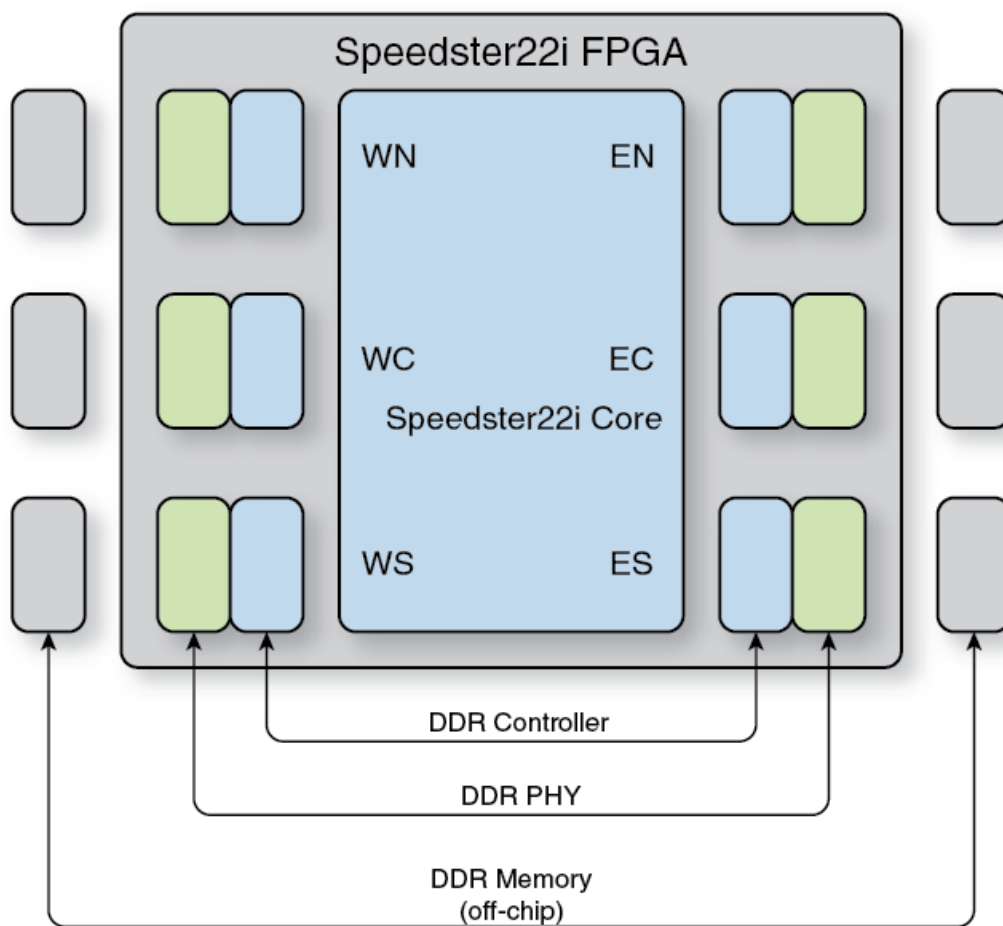
DLL and Write/Read Leveling	62
DLL Specs and Operation	62
Soft Write/Read Leveling	63
Read-Leveling Values	64
Write-Leveling Values	64
Chapter - 8: Memory Interface Latency	65
Chapter - 9: SPD Parameters and Configuration	66
Manual Setting with Parameter Calculator	66
Step 1. Collect Data	66
Step 2. Enter Data into the Spreadsheet	66
Step 3. Transfer the Calculated Values to the Macro	68
Step 4. Regenerate the Bitstream for the Revised Design	68
Automated I2C Read and Update	68
Chapter - 10: ACE Macro Generation and Integration	69
Overview	69
DDR3 IP Diagrams	69
Sections	73
DDR3 Configuration Editor Overview Page	73
DDR3 Configuration Editor Memory Timing Page	77
DDR3 Configuration Editor DLL Timing Page	80
DDR3 Configuration Editor DQS Preamble Page	82
DDR3 Controller Placement Constraints	85
Chapter - 11: Design Simulation	88
Testbenches and Memory Models	88
DDR3 Controller Model	88
DDR3 Memory Model	88
Fast Behavioral Model	89
Simulation Development Cycle	89
Chapter - 12: Debug Using the DDR3 Bring-up Design	91
DDR3 Bring-Up Designs	91
Package 1	91
Package 2	91
Bring-Up Methodology	91
Signal Integrity Analysis	91

Functional Debug and Tuning	92
Chapter - 13: Reference Designs	94
DDR3 Bring-Up Design	94
PCIe Accelerator 6D Card Multi-IP framework	94
Revision History	95

Chapter - 1: Introduction

Features

Achronix's Speedster22i FPGAs contain up to six embedded DDR3 controllers which can be used to interface with and control off-chip DDR3 memory devices, including DIMMs. Each of the DDR3 controllers supports up to 72 bits of data and speeds of up to 1600 Mbps. The embedded DDR3 controllers and PHYs are implemented as hard-IP blocks in the frame of the Speedster22i FPGAs (see the figure below).



3047977-02.2015.0920

Figure 1: Location of Speedster22i DDR Controllers and PHYs

Each of the embedded DDR controllers features:

- **1600 Mbps data rate** – The controller and PHY can run at 800 MHz to achieve an 1600 Mbps rate at the memory interface. The core interface runs at half that speed (400MHz), by using wider data.
- **Up to 72-bit interfaces** – Each controller can support up to a total data width of 72 bits, or 9 bytes. The data width is configurable and can be set to 8, 16, 32, 64 or 72 bits.

- **8:1 DQ:DQS ratio** – A 4:1 ratio can be used at the cost of half the available memory space.
- **Four chip selects per controller** – The external memory connected to each controller can comprise of up to four ranks (either four single-rank DIMMs or two dual-rank DIMMs).
- **Memory form factor independent** – The controllers are agnostic to the form factor of the memory devices on the board. These can be DIMMs, SO-DIMMs, VLP-DIMMs, MIP devices, discrete devices, etc.
- **Multi-burst mode** – Each controller supports multi-burst mode, where a single read or write command from the FPGA core is translated into multiple native read or write commands, with automatic address increments. The burst length can be up to 504 addresses.
- **Bypassable** – If the user does not require all six DDR controllers, any (or all) can be bypassed to leverage use of the designated I/O for other purposes. Moreover, if the user does not require all 72 bits of the data bus, unused byte lanes can be bypassed to leverage use of the designated I/O for other purposes.
- **Minimal LUT usage** – The embedded controllers are hard IP. Only a few features, such as read- and write-leveling, and controller configuration, consume fabric LUTs.
- **Zero license fees** – The DDR controllers are embedded (hard), and therefore, do not require licensing fee for use.

Functional Overview

The Speedster22i DDR3 solution provides a simple to use interface between the user logic, mapped to the FPGA core, and off-chip DDR3 memory. The ACE GUI IP Generator is used to configure the parameters for a DDR3 controller instance (`ACX_DDR3_INTERFACE`), which is then included in the user design. The controller instance provides the following functionality.

- **Clocks** – The controller has a single clock input, normally driven by a PLL, with a maximum frequency of 800 MHz. The controller generates divide-by 2 and divide-by 4 clocks for use in the FPGA core. DDR memory uses a double-data rate protocol (with separate data at the rising and the falling clock edges), hence an 800 MHz clock corresponds to 1600M transactions per second. The interface to the user design is synchronized to the divide-by 2 clock, with a maximum rate of 400 MHz (with data at the rising edge only). Consequently, the interface to the user design has 4 × the width of the external memory (e.g., 288 bits for 72-bit wide DIMMs). The divide-by 4 clock is provided as a convenience to the user, to reduce the rate even further.
- **Reset phase** – At reset, the controller performs the required initialization of the external memory, including programming the DDR's internal mode registers and ZQ calibration. Next, read- and write-leveling are performed, to match the PHY to the byte lane delays of the DIMM. This process is fully automatic, requiring only a reset sequence.
- **Memory read** – To perform a read, the user design signals a read request together with an address and burst size. The controller will respond with an acknowledgement two cycles before the data is available. The maximum burst size is 504 addresses. The controller translates such a burst into consecutive DDR3 transactions, of 8 addresses each (called DDR3 BL8 mode).
- **Memory write** – To perform a write, the user design signals a write request together with an address and burst size. In addition, each byte lane has a data mask bit to enable/disable writes to that byte. There is some delay between issuing the command and when the DIMM is ready to receive the data, hence the controller generates a data request (similar to an acknowledgement) two cycles before the data must be presented to the PHY. Similar to a read, the DDR3 native transaction size is 8 addresses (BL8 mode), but the controller can combine writes into bursts of up to 504 addresses.
- **Command buffering** – The DDR controller has an internal buffer for read and write commands, with a busy signal to indicate when the controller can accept new commands. The user design should provide for buffering of the actual data.

- **Refresh** – By default, the controller periodically issues commands to refresh the external memory. While no memory transactions can take place during a refresh, this does not require special actions of the user design: the command buffer and the request/acknowledgement protocol take care of this situation. Alternatively, the user can disable automatic refresh, and periodically signal the controller when a refresh should take place.
- **DDR3 interface** – The controller takes care of all the details of the the DDR3 interface, such as turning ODT on and off, rank selection, and bank precharge and activate. The controller will issue commands as closely together as possible, subject to the timing requirements of the DDR3 memory. (E.g., the DIMM may require extra time when switching between ranks, and the periodic refresh also takes time.) Combined with the command buffer, this enables operation with maximum throughput.
- **Observation and control** – While not necessary for normal operation, the controller provides access to internal configuration registers through a serial SBUS interface. The read- and write-level delays computed during the reset phase are also available for observation.

Chapter - 2: DDR3 Macro Interfaces

The DDR3 macro, generated using the ACE GUI IP generator, has a core-side interface to connect to the user design, and an external interface to connect to off-chip memory. The DDR3 macro encompasses both the controller and the PHY. The interface signals are listed below, grouped by function.

Internal (Core-side) Interface

The core-side interface connects the DDR controller to the user design. The signal names listed are the root name for the each signal. However, the ACE GUI IP generator can optionally add a suffix to indicate placement of the interface. For example, the signal `sd_a` could become `sd_a_en` for a macro located in the northeast corner of the device. These suffixes are a combination of 'e' (east) or 'w' (west), with 'n' (north), 'c' (center), or 's' (south). In the table below, `DSIZE` refers to the number of bits of the external memory, such as 72 or 64.

Table 1: Internal Interface Signals

Signal Name	Direction	Width	Description
Clock and Reset Interface			
<code>clk</code>	Input	1	Input clock, normally generated by one of the FPGA's PLLs. The memory data rate is twice the frequency of this clock. Common values are 533 MHz, 666 MHz, and 800 MHz, but other values are allowed. The maximum is 800 MHz.
<code>ddr_int_clk_div2</code>	Output	1	This clock has half the frequency of <code>clk</code> . Unless otherwise noted, all core side signals are synchronous with this half-rate clock (the <code>_align</code> suffix of some signal names refers to alignment with this clock).
<code>ddr_int_clk_div4</code>	Output	1	This clock has half the frequency of <code>ddr_int_clk_div4</code> . The serial SBUS interface is synchronous to this quarter-rate clock. This clock is provided as a convenience to the user design.
<code>reset_ddr_ctrlr_n</code>	Input	1	Active-low reset of the DDR controller. Must be driven low at startup.
<code>reset_ddr_phy_n</code>	Input	1	Active-low reset of the PHY. Must be driven low at startup. This signal is normally identical to <code>reset_ddr_ctrlr_n</code> .
<code>ddr_int_rl_reset_in</code>	Input	1	When reset sharing is enabled (<code>GENERATE_RL_RESET=0</code>), this input should be tied to the combined <code>ddr_int_rl_reset_out</code> signals. By default <code>GENERATE_RL_RESET=1</code> , and this input is ignored.

Signal Name	Direction	Width	Description
ddr_int_rl_reset_out	Output	1	When reset sharing is enabled (GENERATE_RL_RESET=0), these outputs from the sharing controllers must be ANDed and input as ddr_int_rl_reset_in.
ddr_int_rl_sm_enable	Input	1	When reset sharing is enabled (GENERATE_RL_RESET=0), the sharing controllers must be linked by connecting ddr_int_ctrlr_init_done of one controller to ddr_int_rl_sm_enable of the next controller. The first controller's ddr_int_rl_sm_enable must be tied to 1'b1, and the last controller's ddr_int_ctrlr_init_done is sent to the user design.
ddr_int_ctrlr_init_done	Input	1	After reset, this signal transitions from low to high once to indicate when the controller has completed initialization and is ready to accept reads and writes. When reset sharing is enabled (GENERATE_RL_RESET=0), this signal must be taken from the last of the linked controllers.
Read/Write Shared Interface			
ddr_int_busy_align	Output	1	Indicates that the DDR3 controller is busy and is not accepting new requests.
ddr_int_burst_size	Input	8	Indicates burst size of given read/write request. This size must be a multiple of 4, within the range 4 to 252. The value is the number of clk cycles: a value of 4 means 8 addresses are read or written (corresponding to two cycles of ddr_int_clk_div2).
ddr_int_addr	Input	34	The address of the first read/write of a request. Addresses are consecutive and range from 0 to the size of the external memory. The address must be a multiple of 8.
Read Interface			
ddr_int_rd_request	Input	1	Read request. Must be held high for just one cycle per read request and held low while ddr_int_busy_align is high.
ddr_int_rddata_valid_align	Output	9	Indicates that valid data is present on ddr_int_rddata. The valid signal precedes the data by two cycles. This signal asserts high for ddr_int_burst_size/2 cycles per request, but it is not guaranteed that these are consecutive cycles. For instance, a memory refresh may interrupt the read sequence as may other DDR timing requirements. To make routing easier there is one valid signal per byte lane, but all nine signals are in fact identical.

Signal Name	Direction	Width	Description
ddr_int_rddata	Output	4×DSIZE	Data corresponding to a read request. This output is valid two cycles after ddr_int_rddata_valid_align is high and contains the data for four consecutive memory addresses.
Write Interface			
ddr_int_wr_request	Input	1	Write request. Must be held high for just one cycle per write request, and then held low while ddr_int_busy_align is high. The address and burst size are latched with the request, but the actual data is only sampled as indicated by ddr_int_wrdata_req_early_align.
ddr_int_wrdata_req_early_align	Output	9	This signal indicates when the PHY is ready to accept the actual data to be written to memory. This signal is asserted high two cycles before the data is present on ddr_int_wrdata. This signal remains high for ddr_int_burst_size/2 cycles per request, but it is not guaranteed that these are consecutive cycles. For instance, a memory refresh may interrupt the write sequence as may other DDR timing requirements. To make routing easier there is one req signal per byte lane, but all nine signals are in fact identical.
ddr_int_wrdata_mask	Input	4×DSIZE /8	Data mask: There is one bit per byte of the request. When the bit is high, that byte will not be written, leaving the corresponding memory location unchanged. The mask bits must be valid at the same time as ddr_int_wrdata.
ddr_int_wrdata	Input	4×DSIZE	Data corresponding to a write request. This data must be valid two cycles after ddr_int_wrdata_req_early_align is asserted high. The data will be written to four consecutive addresses of the memory.
Write/Read Leveling Interface			
ddr_int_wvl_val	Output	DSIZE	The write-leveling delay values (for debug purposes). There are 8 bits per byte lane at index [byte*8+7:byte*8]. To convert 8-bit value v to an integer, use $32*v[7:6] + v[5:0]$. The result is a delay in units of 1/64 clk cycles.
ddr_int_rvl_val	Output	9×DSIZE /8	The read-leveling delay values (for debug purposes). There are 9 bits per byte lane at index [byte*9+8:byte*9]. This 9-bit value is a delay in units of 1/64 clk cycles (no conversion needed).
ddr_int_lvl_err	Input	2	If not 0, indicates an error during read-leveling (bit 0) or write-leveling (bit 1). If such an error occurs, the memory will not operate correctly. A possible cause is using the wrong DDR parameters.

Signal Name	Direction	Width	Description
SBus Interface			
ddr_int_i_sbus_data	Input	2	Disabled if SBUS_ENABLE=0 (default). Serial data input according to the SBUS protocol. Clocked by ddr_int_clk_div4. The SBUS uses a 16-bit address and 32-bit data.
ddr_int_i_sbus_req	Input	1	Disabled if SBUS_ENABLE=0 (default). High during an SBUS transaction.
ddr_int_o_sbus_ack	Output	1	Disabled if SBUS_ENABLE=0 (default). High for one cycle to acknowledge a write, or high during output of read data.
ddr_int_o_sbus_data	Output	2	Disabled if SBUS_ENABLE=0 (default). Serial data output for an SBUS read, The data is 32 bits.
Command and Control Interface			
ddr_int_cmd_ref_req	Input	1	User-initiated refresh request. By default (controller_refresh_en=1), the controller generates the necessary refresh requests using a timer. User control of the refresh is useful in cases where read/write activity occurs in bursts that must not be interrupted by a refresh.
ddr_int_ref_ack	Output	1	Refresh acknowledgement from the controller, indicating that a refresh instruction has been issued to the memory. This acknowledgement requires no special action from the user because the controller does not issue reads or writes for the duration of the refresh.

External (Memory) Interface

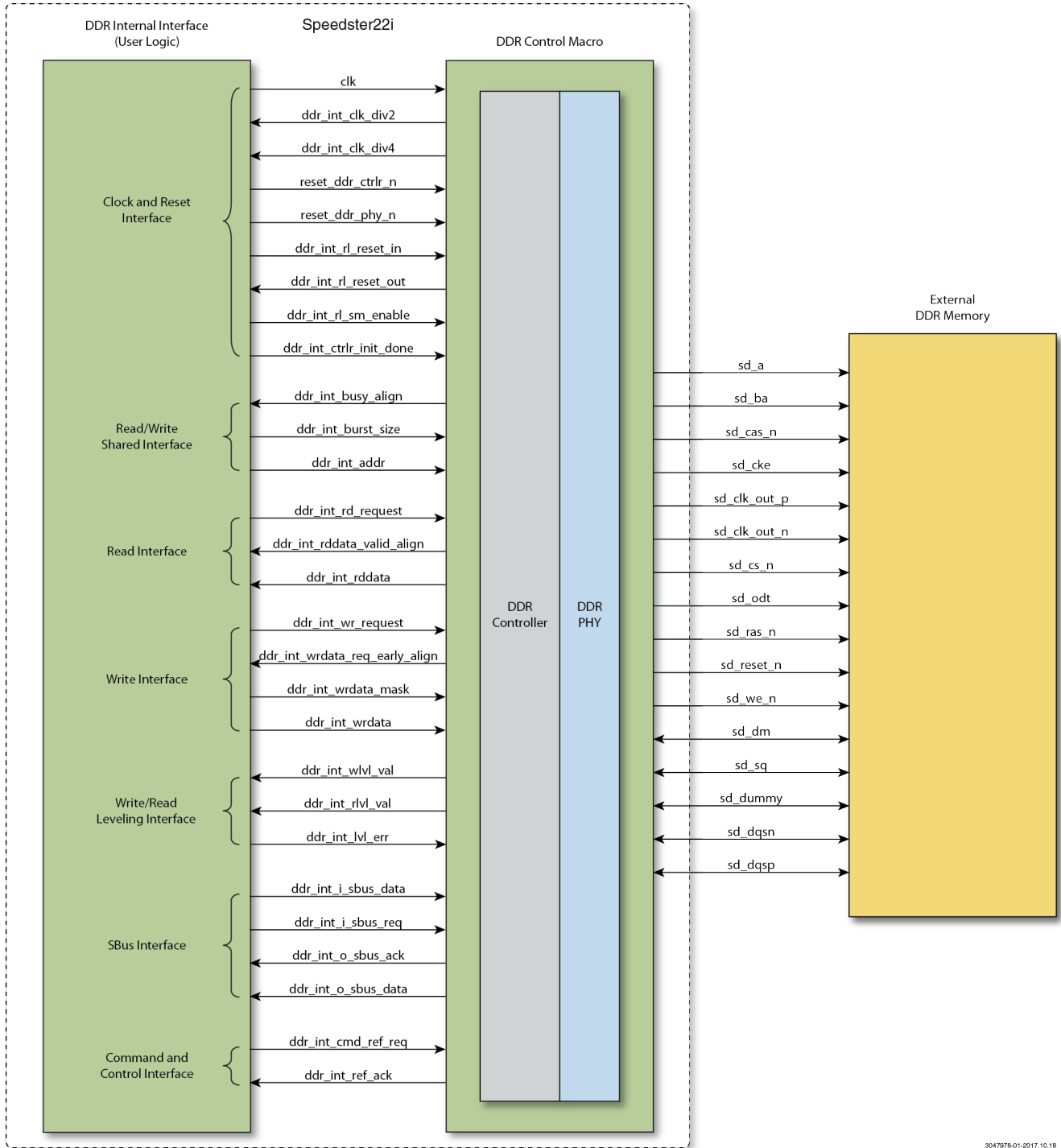
The external DDR interface signals (off-chip) from the DDR PHY to the external memory devices are listed in the table below.

Table 2: PHY/DDR Controller to Memory Interface Signals

Signal Name	Direction	Width	Description
sd_a	Output	ROW_ADDR_WIDTH	SDRAM address bus
sd_ba	Output	3	SDRAM bank select
sd_cas_n	Output	1	SDRAM CAS control signal
sd_cke	Output	NUM_RANKS	SDRAM clock enable control signal

Signal Name	Direction	Width	Description
sd_clk_out_p	Output	NUM_CLK_OUTS	SDRAM differential clock signal
sd_clk_out_n	Output	NUM_CLK_OUTS	
sd_cs_n	Output	NUM_RANKS	SDRAM chip select
sd_odt	Output	NUM_RANKS	SDRAM on-die termination control signal
sd_ras_n	Output	1	SDRAM RAS control signal
sd_reset_n	Output	1	SDRAM reset signal
sd_we_n	Output	1	SDRAM write enable control signal
sd_dm	Bidir.	DSIZE/8	SDRAM data mask
sd_dq	Bidir.	DSIZE	SDRAM data bus
sd_dummy	Bidir.	DSIZE/8	Internal use only. Leave unconnected
sd_dqsn	Bidir.	DSIZE/8	SDRAM DQS bus, which is used to clock DQ bus
sd_dqsp	Bidir.	DSIZE/8	SDRAM DQS bus, which is used to clock DQ bus

DDR3 Macro Block Diagram



3047978-01-2017.10.18

Figure 2: DDR3 Macro Interfaces and Signals

Chapter - 3: DDR3 Core Parameters

Parameters that can be set for both the DDR3 controller and PHY are shown in the table below.

Table 3: Parameter Values of the DDR3 Controller

Parameter	Default Value (hex)	Valid Values	Description
DSIZE	72	Multiples of 8 from 8–72	Local side data width
NUM_CLK_OUTS	3	1 to 4	Number of clock outputs
CONTROLLER_REFRESH_EN	1'h1	0–1	Enable controller initiated refreshes: 0 – Embedded DDR Controller automatically handles cyclic autorefreshing of memory. 1 – User manually overrides autorefresh control of memory.
DELAY_ACTIVATE_TO_PRECHARGE	5'h14	4–30 clock cycles	Minimum ACTIVE to PRECHARGE
DELAY_ACTIVATE_TO_RW	4'h7	2–12 clock cycles	Minimum time between ACTIVATE and READ/WRITE
DELAY_ACTIVATE_TO_ACTIVATE_DIFF_BANK	3'h4	2–6 clock cycles	Minimum time between ACTIVATE to ACTIVATE in different banks
DELAY_PRECHARGE_TO_ACTIVATE	4'h7	1–12 clock cycles	Minimum PRECHARGE to ACTIVATE.
DELAY_ACTIVATE_TO_ACTIVATE_SAME_BANK	6'h20	5–42 clock cycles	Minimum ACTIVATE to ACTIVATE/AUTO-REFRESH in same bank


Parameter	Default Value (hex)	Valid Values	Description
BANK_ACTIVATE_PERIOD	6'h20	7–32 clock cycles	Four bank activate period
DELAY_AUTO_REFRESH_TO_ACTIVATE_SAME_BANK	9'h03B	6–511 clock cycles	Minimum AUTO-REFRESH to ACTIVATE/AUTO-REFRESH in same bank
DELAY_READ_TO_PRECHARGE	3'h4	4–6 clock cycles	Minimum Read to precharge (DDR3 only)
DELAY_WRITE_TO_PRECHARGE	4'h8	5–12 clock cycles	Minimum time from write to PRECHARGE
DELAY_WRITE_TO_READ	3'h4	2–6 clock cycles	Minimum time from write to read.
DELAY_READ_TO_WRITE	3'h3	1–7 clock cycles	Read-to-write delay (valid values: 1
DELAY_WRITE_TO_WRITE_DIFF_BANK	3'h2	0–7 clock cycles	Minimum delay from write to write (different ranks)
DELAY_WRITE_TO_READ_DIFF_BANK	3'h1	0–7 clock cycles	Minimum delay from write to read (different ranks)
DELAY_READ_TO_READ_DIFF_BANK	3'h2	1–7 clock cycles	Minimum delay from read to read (different ranks)
DELAY_ADDITIVE_DDR3_LATENCY	3'h0	0–5 clock cycles	Additive latency value (DDR2)
CAS_LATENCY	4'h7	5–11	Cas latency
CAS_WRITE LATENCY	4'h6	5–8	Cas write latency
CONTROLLER_REFRESH_EN	1'b1	0–1	Controller refresh enable

Parameter	Default Value (hex)	Valid Values	Description
DELAY_LOADMODE_TO_ACTIVATE	3'h6	clock cycles	Minimum LOADMODE to ACTIVATE command
DELAY_LOADMODE_TO_ANY	4'hE	clock cycles	Minimum LOADMODE to ANY command
DELAY_SELF_REFRESH_TO_NON_DLL_CMD	9'h040	clock cycles	Minimum time from self-refresh to non-DLL command
DELAY_SELF_REFRESH_TO_NON_READ_CMD	8'h87	clock cycles	Minimum time from self-refresh to non-read command
DELAY_RESET_HIGH_TO_CLK_HIGH	9'h040	clock cycles	Minimum delay from memory reset high to CKE high
REFRESH_PERIOD	16'h07D0	10–65535 refresh time interval / tCK	Refresh period. This is the number of clock cycles between refresh commands
DELAY_STARTUP	19'h000c8	10–524287 clock cycles	Initialization delay after reset
NUM_COLUMN_BITS	3'h5	5, 6, 7	Number of column bits: 5 – 10 column bits 6 – 11 column bits 7 – 12 column bits
NUM_ROW_BITS	3'h3	2, 3, 4, 5	Number of row bits: 2 – 13 row bits 3 – 14 row bits 4 – 15 row bits 5 – 16 row bits
NUM_BANK_BITS	1'h1	0, 1	Number of bank bits: 0 – 2 bank bits 1 – 3 bank bits

Parameter	Default Value (hex)	Valid Values	Description
REGISTERED_DIMM	1'h0	0,1	Registered DIMM support: 0 – UDIMM 1 – RDIMM
ODT_READ_CS[0:7]	8'h00	Each bank contains 8 bits, one per DQ. ODT is enabled when set to 1	On die termination selection for reads on cs[0:7]
ODT_WRITE_CS0	8'h01		On die termination selection for writes on cs[0:7]
ODT_WRITE_CS1	8'h02		
ODT_WRITE_CS2	8'h04		
ODT_WRITE_CS3	8'h08		
ODT_WRITE_CS4	8'h10		
ODT_WRITE_CS5	8'h20		
ODT_WRITE_CS6	8'h40		
ODT_WRITE_CS7	8'h80		
READ_EN_DELAY_LANE	3'h5	0–5	Adjusts the delay of the read data out of the PHY
BYPASS_TXRX_SD	0	0 or 1	0 – 1× clock mode. Core runs at same speed as the controller 1 – 2× clock mode. Core runs at half the speed of the controller.
BYTE_LANE*_DLL_ADJ_DQ	6'h4		DQ slave adjust for byte lane ^(†)
BYTE_LANE*_DLL_ADJ_DQS	6'h16		DQS slave adjust for byte lane ^(†)
BYTE_LANE*_DLL_ADJ_DP	6'h6		DP Slave adjust for byteLane ^(†)

Parameter	Default Value (hex)	Valid Values	Description
BYTE_LANE*_WR_LVL_DQ_SELECT	4'b0000		Which DQ bit is used for write leveling ^(†)
BYTE_LANE_DLL_MADJ	8'h2a		Master adjust
BYTE_LANE_DLL_DQSx9_CLK_ADJ	8'h3f		Slave adjust for DQSx9
BYTE_LANE_CAC_DLL_ADJ_DQSN	6'h17		Slave adjust for address bytes lanes

Note

 † These parameters are available for all byte lanes. For example, 64 DQ bits are needed for 8 byte lane. Therefore, these parameters are available for all 8 lanes; replace the * with each byte number.

Chapter - 4: Read and Write Operations

Read Interface Details

The Speedster22i DDR Controller contains a very simple read interface to the DDR driver logic. Users can select between the default 2× clock mode or wide-bus mode.

Read Protocol 2× Clock Mode

2× clock mode must be used for data rates above 1066 Mbps, although can be used for lower data rates as well. When using 2× clock mode DDR drive (user) logic in core runs at half the frequency of DDR controller. The DDR controller/PHY outputs a clock (`ddr_int_clk_div2`), which the user must use to drive write data and latch read data.

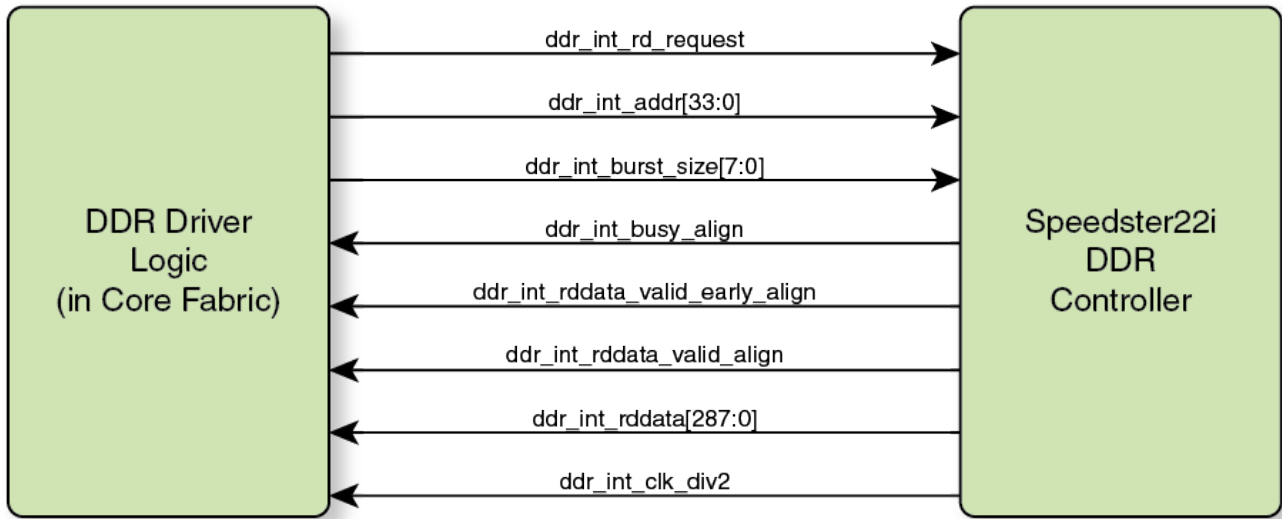
The core interface signals remain the same except for `ddr_int_busy`, `ddr_int_rddata_valid`, `ddr_int_rddata_valid_early`. Instead, the DDR driver (user) logic needs to use `ddr_int_busy_align`, `ddr_int_rddata_valid_align`, `ddr_int_rddata_valid_early_align`.

The DDR driver (user) logic must provide a read request, (`ddr_int_rd_request`), along with a corresponding address, (`ddr_int_addr`), and burst length, (`ddr_int_burst_size`).

Speedster22i DDR controller provides a read valid signal, (`ddr_int_rddata_valid_align`) for data to be read. After assertion of `ddr_int_rddata_valid_align`, 2 cycles later, the DDR driver logic receives read data (`ddr_int_rddata[287:0]`) from the Speedster22i DDR controller. The `ddr_int_rddata[287:0]` signal represents the data read from the memory over four sequential DDR clock edges (72 bits at a time). The data contained in `ddr_int_rddata[71:0]` is read from the specified column address, and that contained in `ddr_int_rddata[143:72]` is read from the specified column address + 1. Data from `ddr_int_rddata[215:144]` is read from specified column address+2 and data from `ddr_int_rddata[287:216]` is read from specified column address+3.

The read requests are accepted when the controller is not busy, indicated by `ddr_int_busy_align`.

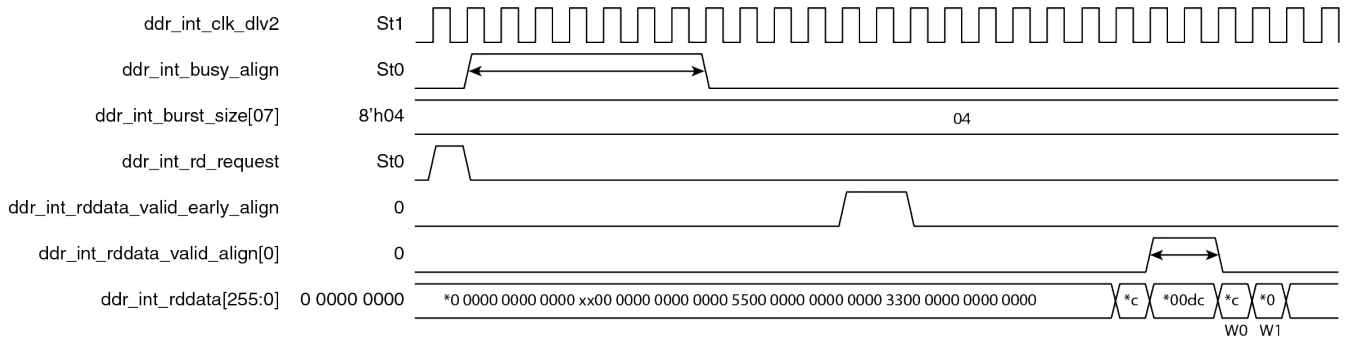
The DDR controller supports burst length option BL8. Each burst will contain 2 local side transfers, which is equivalent to 8 transfers to the DDR memory.



3047981-03.2018.09.05

Figure 3: Read Interface 2x Clock Mode

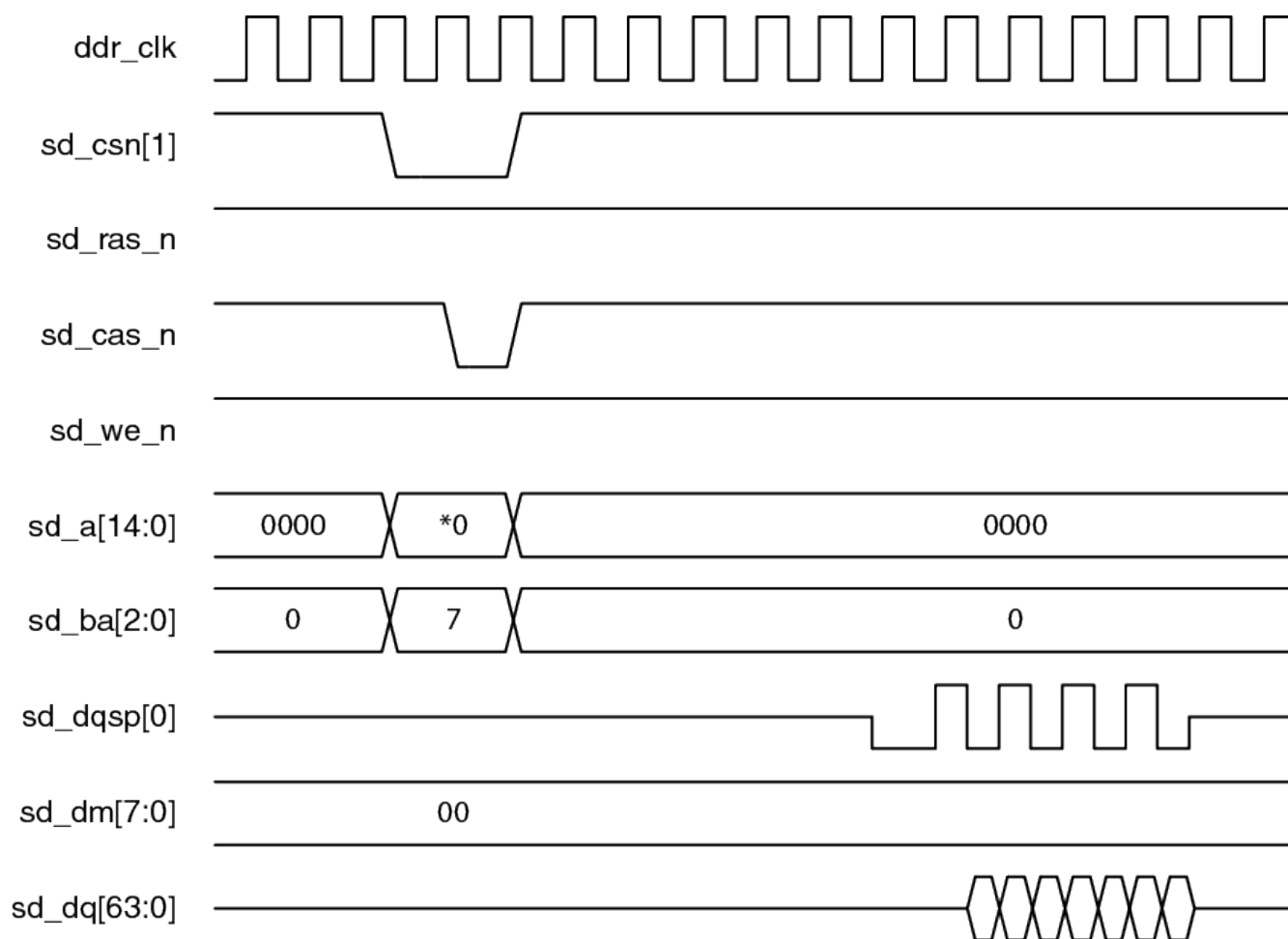
The following timing diagram illustrates a single read command of burst length 4. The signals shown in the following diagrams are ports at the (`ddr3_xSIZE1_LOCATION2` . Where 1: SIZE = 72, 64, 32, 16, 8 and 2: LOCATION=EN, EC, ES, WN, WC, WS).



3047981-04.2018.09.05

Figure 4: Internal Interface Read Protocol Timing Diagram

The Corresponding external (off-chip) interface timing signals are shown in the figure below



3047981-01-2015.10.27

Figure 5: External Interface Read Protocol Timing Diagram

To request a read data transaction, the DDR driver (user) logic must assert `ddr_int_rd_request` along with a corresponding address (`ddr_int_addr[33:0]`) and burst length (`ddr_int_burst_size`).

A *valid* read request (ie. one which is successfully posted to the Speedster22i DDR controller and propagated to the DDR Memory) is one in which ALL of the following conditions are met:

- `ddr_int_rd_request` is asserted (active high)
- `ddr_int_addr [33:0]` is driven
- `ddr_int_burst_size [7:0]` is driven to a valid value for the given protocol
 - `8'd4` → `8'd252` for DDR3 (multiple of 4)
- `ddr_int_busy_align` is not asserted (active high)

Back-to-Back Read Protocol 2× Clock Mode

The first word is provided on `ddr_int_rddata[143:0]` four cycles after `ddr_int_rddata_valid` is received from the Speedster22i DDR controller.

The following timing diagrams illustrate two cascaded, back-to-back read commands. Each *valid* read request (and corresponding data) is highlighted in a different color. The testing is done with respect to 2×-clock mode.

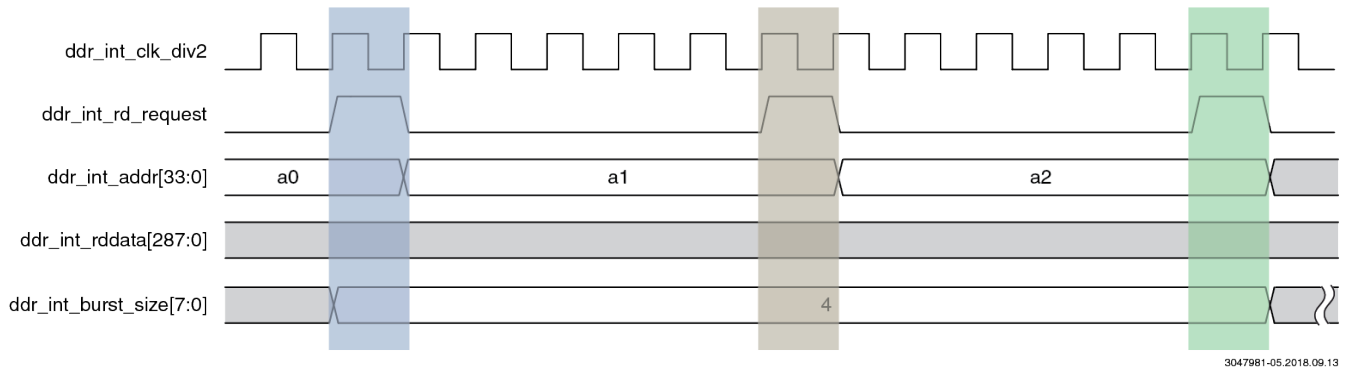


Figure 6: Read Protocol Timing Diagram (with valid read request highlighted)

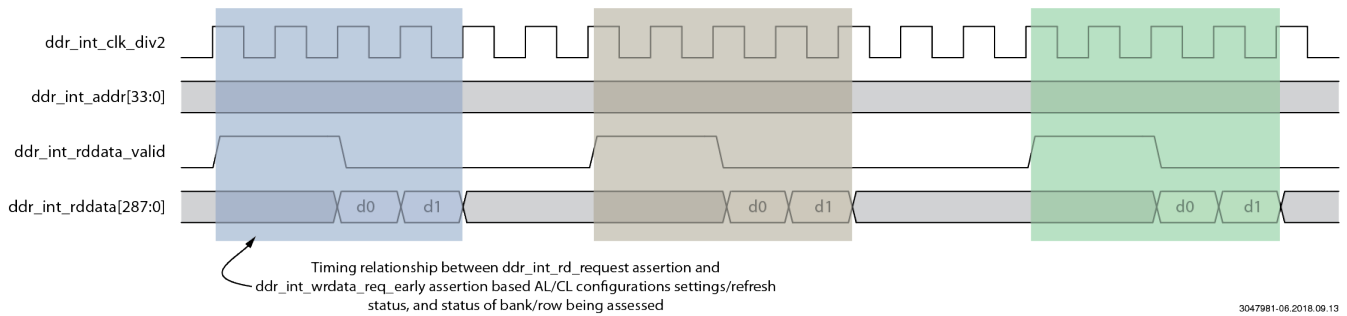


Figure 7: Read Protocol Timing Diagram (with `ddr_int_rddata_valid` highlighted)

If a read request is not valid (i.e. `ddr_int_busy` and `ddr_int_rd_request` are asserted simultaneously), `ddr_int_rd_request`, `ddr_int_addr [33:0]`, and `ddr_int_burst_size [7:0]` signal values must be latched until such time as `ddr_int_busy` is de-asserted and a valid read request can be posted as shown in Figure 21. There should be 5-clock cycle gap required for the back-to-back read request (`ddr_int_rd_request`).

The `ddr_int_rd_request` signal may remain asserted for any number (with 5-cycles apart always) of clock periods to generate any number of follow-on read transactions (in cascaded bursts).

The read data corresponding to a given read request is returned a deterministic number of clock cycles after the read request. This deterministic amount of time represents the round-trip delay of accessing as well as actually reading from the memory address. The read data is accompanied by a valid signal, denoted `ddr_int_rddata_valid`.

The Speedster22i DDR Controller will ensure that the `ddr_int_rddata_valid` signal is asserted for the correct number of cycles (based on the `ddr_int_burst_size` value specified for the corresponding read request) as well as at the correct time (in accordance with DDR latency requirements based on user-specified values of AL and CL parameters and the total round-trip latency of the given memory access).

The burst length (`ddr_int_burst_size`) corresponding to a single given read request must be set to a valid value based on the given DDR protocol. For DDR3 has a range of 8'd4 to 8'd252. Note that the `ddr_int_burst_size` value translates directly to the number of cycles for which `ddr_int_rddata_valid` is asserted.

As with writes, while bank and row addresses are derived directly from the address provided by the DDR driver (user) logic for a given read request, the column address is incremented automatically within the Speedster22i DDR Controller, starting with the column address provided by the user. For DDR3, since `ddr_int_burst_size` is set as a multiple of 4, the user should always provide a column address with a modulo-8 value.

The `ddr_int_rddata[287:0]` signal represents the data read from the memory over two sequential DDR clock edges (144 bits at a time). The data contained in `ddr_int_rddata[71:0]` is read from the specified column address, that contained in `ddr_int_rddata[143:72]` is read from the specified column address + 1, that contained in `ddr_int_rddata[215:144]` is read from the specified column address + 2, and that contained `ddr_int_rddata[287:216]` is read from the specified column address + 3.

To request a read data transaction, the DDR driver (user) logic must assert `ddr_int_rd_request` along with a corresponding address (`ddr_int_addr[33:0]`) and burst length (`ddr_int_burst_size`).

Write Interface Details

The Speedster22i DDR3 Controller contains a simple write interface to the DDR3 driver logic. Users can select between the default 2× clock mode or wide-bus mode.

Write Protocol 2× Clock Mode

When generating a DDR3 macro for Speedster22i Devices, users have the option to choose 2× clock modes for higher off-chip data rates. In 2× clock mode, the DDR3 driving logic (user RTL implemented in the FPGA fabric) runs at half the frequency of DDR3 controller. The DDR3 controller/PHY outputs a Clock (`clk_div2`), which the user must use to drive write data and latch read data. For example, the 2× mode allows the internal interface to the fabric to operate at 400 MHz, resulting data rate of 1600 Mbps.

In 2× mode, the core interface signals remain the same except for `ddr_int_busy`, `ddr_int_wrdata_req`, `ddr_int_wrdata_req_early`. Instead of the above signals, users must interface with the following signals; `ddr_int_busy_align`, `ddr_int_wrdata_req_align`, `ddr_int_wrdata_req_early_align`.

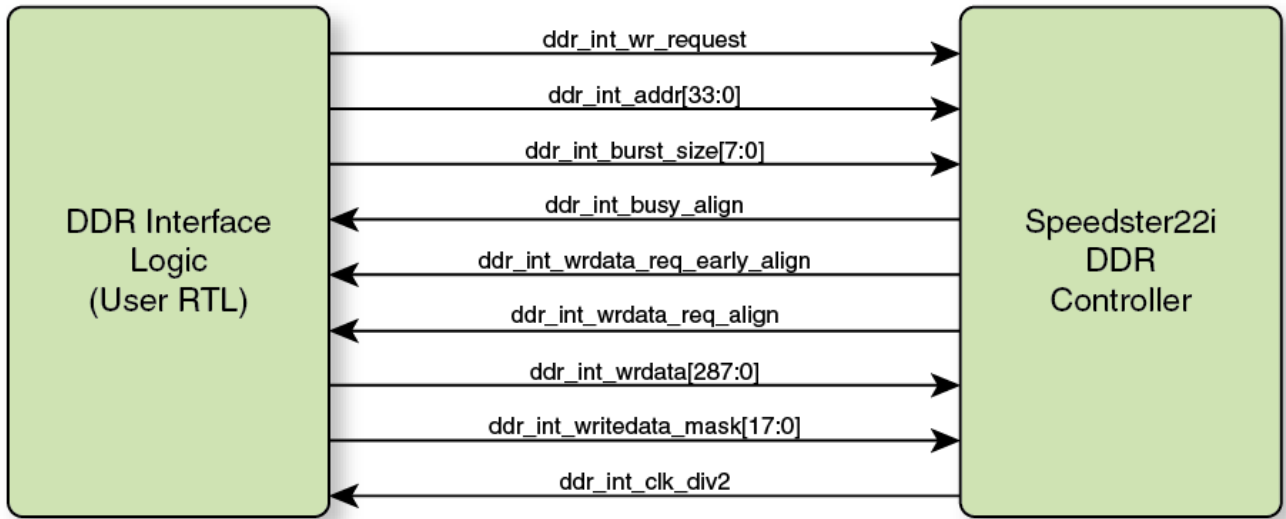
The DDR driver logic (user RTL) provides a write request (`ddr_int_wr_request`) along with a corresponding address (`ddr_int_addr`) and burst length (`ddr_int_burst_size`).

This logic then needs to provide data (`ddr_int_wrdata`) 2 clock cycles after a data request (`ddr_int_wrdata_req_early_align`) is received from the Speedster22i DDR3 Controller. The logic can also use `ddr_int_wrdata_req_align`. The `ddr_int_wrdata[287:0]` signal represents the data to be written to the memory over four sequential DDR3 clock edges (72 bits at a time). The data contained in `ddr_int_wrdata[71:0]` is written to the specified column address, and that contained in `ddr_int_wrdata[143:72]` is written to the specified column address + 1. Data from `ddr_int_wrdata[215:144]` will be written to specified column address+2 and data from `ddr_int_wrdata[287:216]` will be written to specified column address+3.

The write requests are subject to the controller not being busy, indicated by `ddr_int_busy_align`.

During each write cycle, `ddr_int_wrdata_mask` must have the appropriate bits set to indicate which bytes of each transfer are being written.

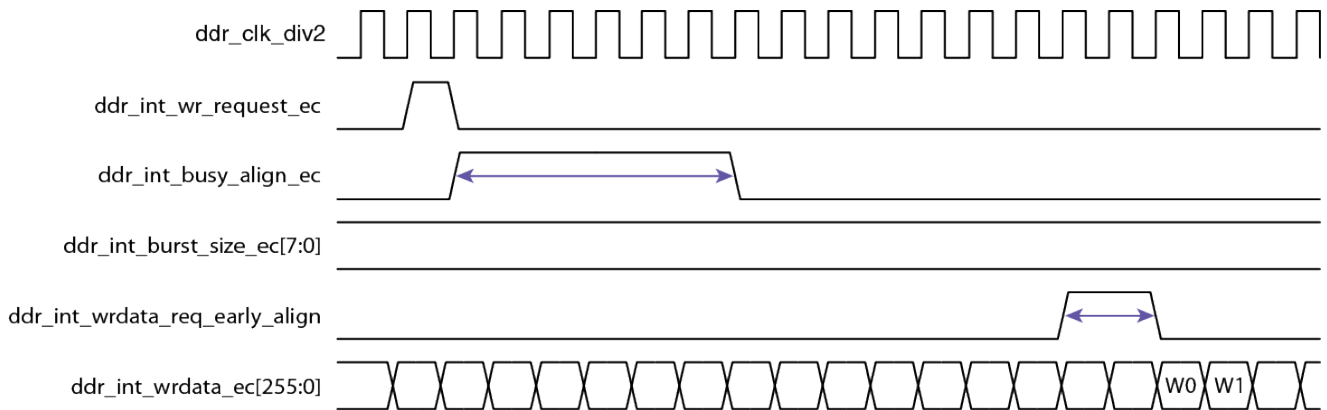
The DDR3 controller supports burst length option BL8. Each burst will contain 2 local side transfers, which is equivalent to 8 transfers to the DDR3 memory.



3047980-07.2018.10.28

Figure 8: Write Interface 2× Clock Mode

The following timing diagram illustrates a single write command of burst length 4. The signals shown in the following diagrams are ports at the (ddr3_xSIZE1_LOCATION2 . Where 1: SIZE = 72, 64, 32, 16, 8 and 2: LOCATION=EN, EC, ES, WN, WC, WS).



3047980-01-2015.10.23

Figure 9: Internal Interface Write Protocol Timing Diagram

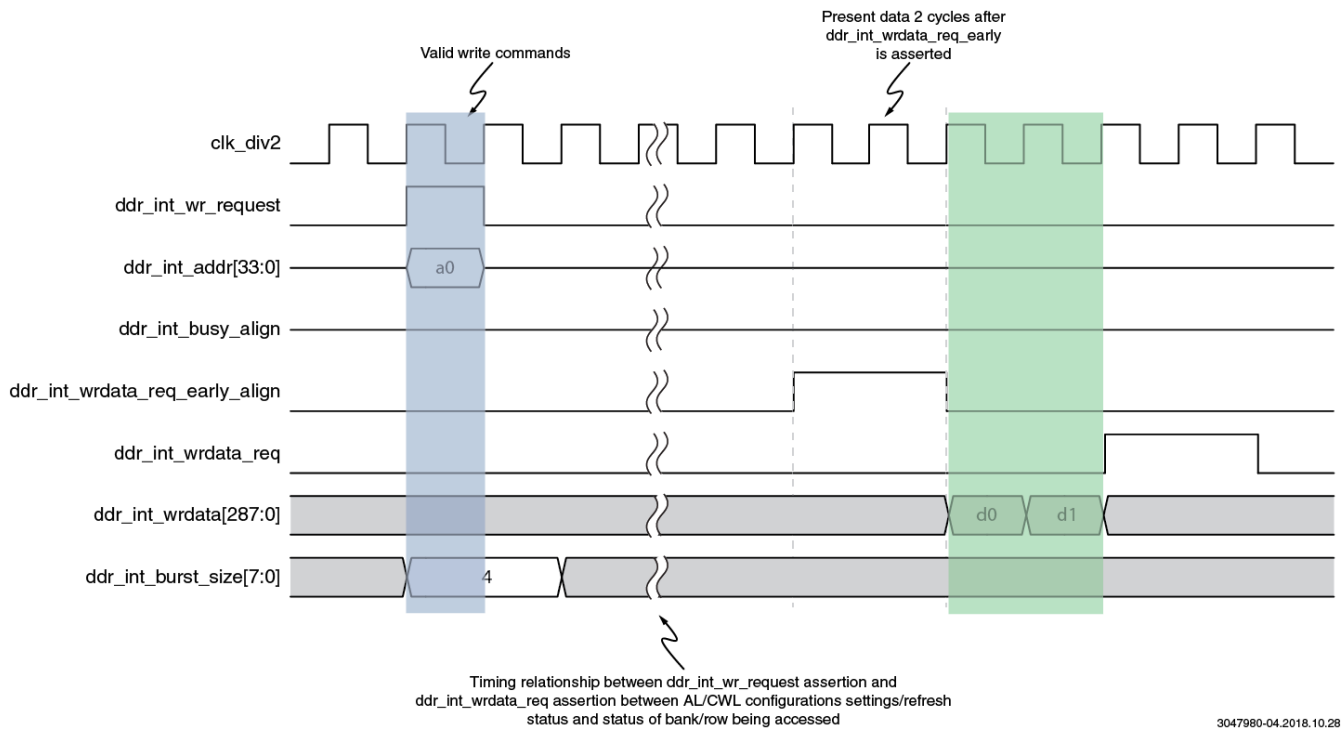


Figure 10: Internal Interface Write Protocol Timing Diagram with `addrcmd_delay` set to 9

This 2-cycles delay between the `ddr_int_wrd_data_req_early_align` and `ddr_int_wrd_data[287:0]` is optional. This is implemented using a parameter `addrcmd_delay` value setting to "4'h9". This extra cycle provides an advantage to the user to latch the data to be written to the PHY. This implementation is optional to the user. If the user wants to use 1-cycle delay between `ddr_int_wrd_data_req_early_align` and `ddr_int_wrd_data[287:0]`, in this case let keep the default value of the parameter `addrcmd_delay`. The default values for this parameter `addrcmd_delay` is "4'h8". The user needs to be very careful about their implementation while using Achronix DDR3-controller implementing 1-cycle delay implementation for latching the write-data to be written to PHY. For the 2-cycle delay, the user has enough time to latch the data.

The diagram above shows the signals between the FPGA fabric core and DDR3 controller. (`ddr_int_wrd_data_req_early_align`) signal is 3 cycles earlier than the `ddr_int_wrd_data_req` signal. The user can use `ddr_int_wrd_data_req_early_align` signal to be ready to write the data and at the assertion of `ddr_int_wrd_data_req` signal to drive user data on `ddr_int_wrd_data`.

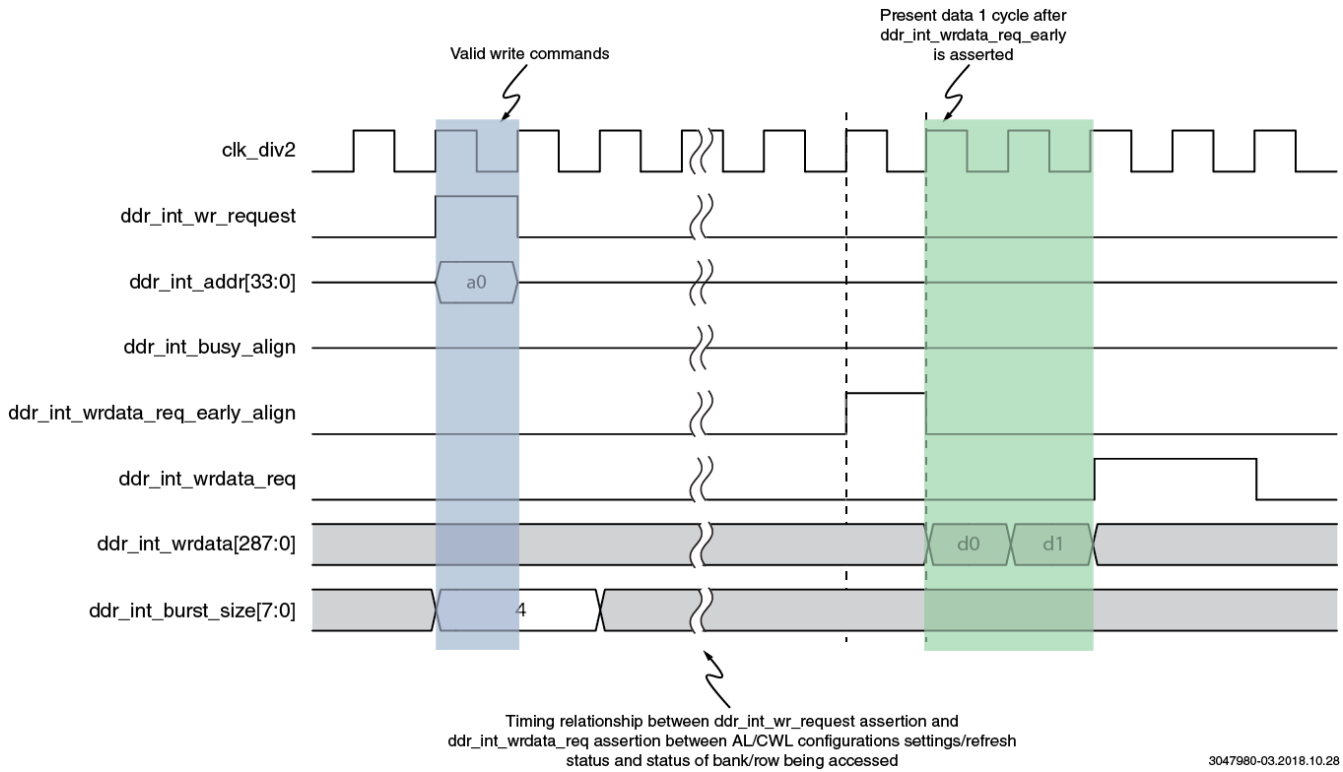


Figure 11: Internal Interface Write Protocol Timing Diagram with default value `addrcmd_delay` to 8

This logic then needs to provide data (`ddr_int_wrd_data`) one clock cycle after a data request (`ddr_int_wrd_data_req_early_align`) is received from the Speedster22i DDR3 Controller. The logic can also use `ddr_int_wrd_data_req_align`. The `ddr_int_wrd_data[287:0]` signal represents the data to be written to the memory over four sequential DDR3 clock edges (72 bits at a time). The data contained in `ddr_int_wrd_data[71:0]` is written to the specified column address, and that contained in `ddr_int_wrd_data[143:72]` is written to the specified column address + 1. Data from `ddr_int_wrd_data[215:144]` will be written to specified column address+2 and data from `ddr_int_wrd_data[287:216]` will be written to specified column address+3.

The write requests are subject to the controller being not being busy, indicated by `ddr_int_busy_align`.

The DDR3 controller supports burst length option BL8. Each burst will contain 2 local side transfers, which is equivalent to 8 transfers to the DDR3 memory.

The diagram above shows the signals between the FPGA fabric core and DDR3 controller; `ddr_int_wrd_data_req_early_align` signal is 3 cycles earlier than `ddr_int_wrd_data_req` signal. The user can use `ddr_int_wrd_data_req_early_align` signal to be ready to write the data and at the assertion of `ddr_int_wrd_data_req` signal to drive user data on `ddr_int_wrd_data`.

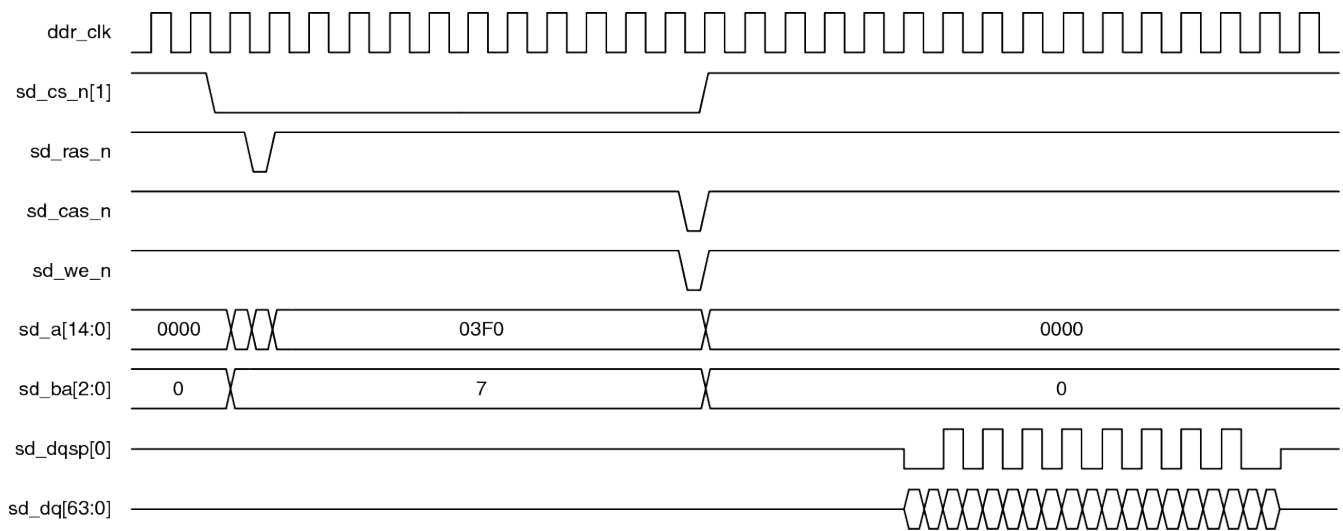


Figure 12: Write Protocol Timing Diagram (SDRAM Interface)

To request a write data transaction, the DDR3 driver logic (user RTL) must assert `ddr_int_wr_request` along with a corresponding address (`ddr_int_addr[33:0]`) and burst length (`ddr_int_burst_size`).

A *valid* write request (ie. one which is successfully posted to the Speedster22i DDR3 Controller and propagated to the DDR3 Memory) is one in which ALL of the following conditions are met:

- `ddr_int_wr_request` is asserted (active high)
- `ddr_int_addr[33:0]` is driven
- `ddr_int_burst_size[7:0]` is driven to a valid value for the given protocol: `8'd4` → `8'd252` for DDR3 (multiple of 4)
- `ddr_int_busy_align` is not asserted (active high)
- Latency of the DDR3-controller provided data (`ddr_int_wrdata`) to the external memory is 4-cycle.

Back-to-Back Write Protocol 2× Clock Mode

The following timing diagram (Figure (see page 30) below) illustrates the same three cascaded, back-to-back write commands. Each *valid* write request (and corresponding data) is highlighted in a different color. For back-to-back write there are 5 clock-cycles gap required between the write requests. This is done with respect to 2× clock mode.

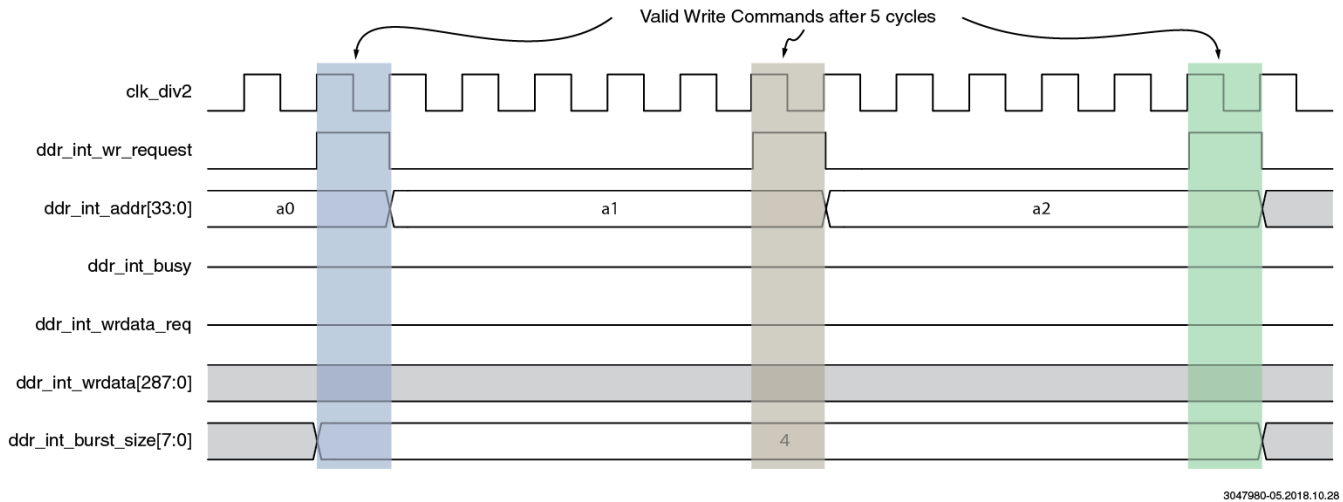


Figure 13: Write Protocol Timing Diagram (Write requests with valid writes highlighted)

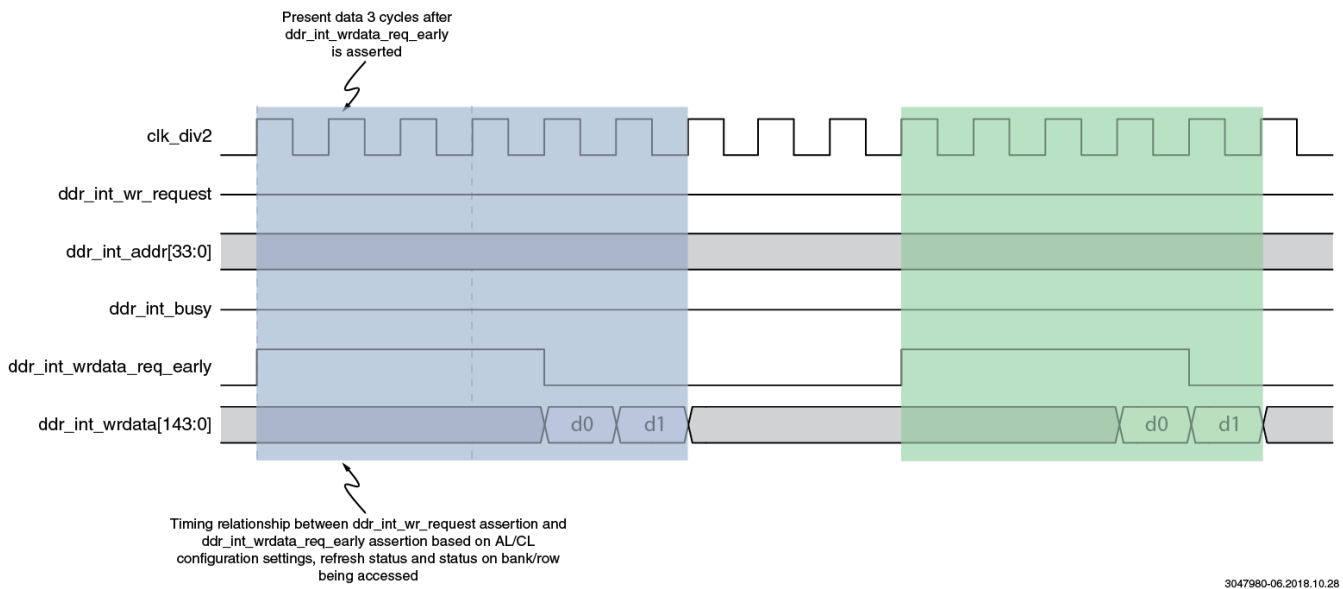


Figure 14: Write Protocol Timing Diagram (ddr_int_wrdata_req Corresponding to Respective Writes Highlighted)

If a write request is not valid (ie. `ddr_int_busy` and `ddr_int_wr_request` are asserted simultaneously), `ddr_int_wr_request`, `ddr_int_addr[33:0]`, and `ddr_int_burst_size[7:0]` signal values must be latched until `ddr_int_busy` is de-asserted and a valid write request can be posted as shown in [Figure: Write Protocol Timing Diagram \(SDRAM Interface\)](#) (see page 29). There should be 5-cycles delay between the back-to-back `ddr_int_wr_request` signal as shown in [Figure: Write Protocol Timing Diagram \(SDRAM Interface\)](#) (see page 29). Corresponding `ddr_int_wrdata_req` and `ddr_int_wrdata` timing is shown in the [figure](#) (see page 30) immediately above.

The `ddr_int_wr_request` signal may remain asserted for any number (with 5-cycles apart always) to generate any number of follow-on writes transactions (in cascaded bursts).

The data request (`ddr_int_wrd_data_req`) signal will be asserted two cycles prior to when the DDR3 driver logic (user RTL) must present data at the `ddr_int_wrd_data` bus and mask information at the `ddr_int_wrd_data_mask` bus.

The Speedster22i DDR3 Controller will ensure that the `ddr_int_wrd_data_req_early` signal is asserted for the correct number of cycles (based on the `ddr_int_burst_size` value specified for the corresponding write request) as well as at the correct time (in accordance with DDR3 latency requirements based on user-specified values of AL and CWL parameters in the case of DDR3).

The burst length (`ddr_int_burst_size`) corresponding to a single given write request must be set to a valid value based on the given DDR3 protocol. For DDR3 has a range of 8'd4 to 8'd252. Note that the `ddr_int_burst_size` value translates directly to the number of cycles for which `ddr_int_wrd_data_req` is asserted.

While bank and row addresses are derived directly from the address provided by the user (ie. DDR driver logic) for a given write request, the column address is incremented automatically within the Speedster22i DDR3 Controller for a given burst, starting with the column address provided by the user for the given write request. For DDR3, since `ddr_int_burst_size` is set as a multiple of 4, the user should always provide a column address with a modulo-8 value.

The `ddr_int_wrd_data[287:0]` signal represents the data to be written to the memory over two sequential DDR3 clock edges (144 bits at a time). The data contained in `ddr_int_wrd_data[71:0]` is written to the specified column address, that contained in `ddr_int_wrd_data[143:72]` is written to the specified column address + 1, that contained in `ddr_int_wrd_data[215:144]` is written to the specified column address + 2, and that contained in `ddr_int_wrd_data[287:216]` is written to the specified column address + 3.

The `ddr_int_wrd_data_mask[17:0]` represents the data mask. Each bit within this bus corresponds to 8bits of the `ddr_int_wrd_data[287:0]` signal bus. Asserting a given bit within the `ddr_int_wrd_data_mask` bus ensures that the corresponding 8 bits of the `ddr_int_wrd_data` bus do NOT get written to memory, overwriting its previous contents.

Chapter - 5: Address Mapping

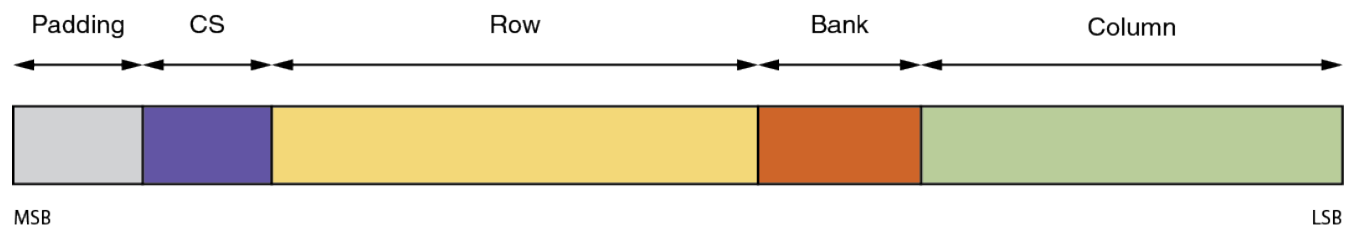
The Speedster22i DDR3 controller contains a specific address bus mapping is described in the following table. The exact bit widths of the mapping vary depending on the values of the configuration parameters defined by the user in the core instantiation (see [DDR3 Core Parameters](#) (see page 16)). The user will set these values based on the component memory used in the DIMM or MIP configuration.

Table 4: Address Mapping (LSB to MSB) for *ddr_int_addr*

Field	Core Parameter	Width (Bits)	Description
Column	NUM_COLUMN_BITS	10 to 12	Value determined by the user and set in the core parameters based on the component memories being used.
Bank	NUM_BANK_BITS	3	Always 3 bits as the number of banks is 8 for DDR3.
Row	NUM_ROW_BITS	12 to 16	Value determined by the user and set in the core parameters based on the component memories being used.
Chip Select	–	0 to 2	Zero bits for a single rank; one bit for two ranks; two bits for four ranks.
Padding	–	Varies	Address is padded with zeros to reach the required 34 bits.

Speedster22i devices support up to four ranks per DDR3 interface (i.e., four single-rank DIMMs, or two dual-rank DIMMs). The chip select bits are encoded in the address mapping and are decoded when signaling to the external memory interfaces. Single-rank DIMM applications have the chip select hard coded to 0 — `sd_cs_n[0]` is always low to indicate selection of that rank. Dual-rank DIMM applications use one `CS[0]` bit in the address map to toggle assertion between `sd_cs_n[0]` and `sd_cs_n[1]`. Four-rank implementations use two bits in the address map: `CS[1:0]`, to decode into `sd_cs_n[3:0]`.

In configurations where the sum total of the column bits, row bits, bank bits and chip select bits do not add up to the total 34-bit length required for `ddr_int_addr`, the upper bits are padded with zeros.



LSB
3047979-01_2016.08.04

Figure 15: Address Mapping of *ddr_int_addr* Signal

**Important!**

The HD1000 DDR3 interface uses BL8 for write and read operations, which necessitates the 34-bit address being a multiple of 8. Every burst writes to or reads from eight consecutive addresses. As a result, `ddr_int_addr[2:0]` must always be set to `3'b000`.

Chapter - 6: Board Design and Layout

Routing Guidelines and Trace Matching

A successful DDR3 implementation able to achieve data rates of 1600 Mbps or higher requires careful board planning and design. Signal routing on the board must be completed in a way that ensures power-noise immunity, minimal crosstalk and trace matching, taking medium propagation delay into account and looking beyond simple length calculations. Given that the HD1000 provides the capability of interfacing with six DDR3 controllers, each capable of supporting a data width of 72 and up to four ranks of memory, there is a significant set of board traces that need to be meticulously routed.

At 1600 Mbps, the 1T period time is $1/1600 \text{ Mbps} = 625 \text{ ps}$. The general guideline is that users should ensure that signals are matched to within 1% of their respective period times.

- Data byte-lanes should be matched to within ~6ps.
- Control-address-command (CAC) byte-lanes, which have a 2T period, should be matched to within ~12ps.

These delays should be calculated from the starting point (the die) to the endpoint (DIMM) and should include all delays in the system including those in the package, the board and the DIMM itself. The analysis should be done for a worst-case device, board and DIMM combination.

Board-Level Simulation

It is important to do a complete board-level simulation with all of the components in the system, to ensure signal integrity and adequate timing margins. Achronix provides SSTL15 I/O buffer IBIS models to help with these simulations. These models support I/O buffer options such as drive strength and termination, as well as package-level parameters.

The user should combine the IBIS models from Achronix with their own board model and source IBIS models from their memory vendor. They will then be able to perform a full-system analog characterization and simulation to ensure that their desired margins for signal integrity are met.

DDR3 Routing Guidelines

Achronix's Speedster22i FPGAs contain up to six embedded DDR controllers (or channels) which can be used to interface with and control off-chip DDR3 memory devices, including DIMMs. The embedded DDR3 controllers and PHYs are implemented as hard-IP blocks in the frame of the Speedster22i FPGAs.

Each of these channels is labeled EN (east-north), WN (west-north), EC (east-center), WC (west-center), ES (east-south), or WS (west-south). For sake of convenience, each of these channels will be termed as [all]. Each channel is associated with a group: address, control, and data signal. For example, the EN channel is composed of the signals in the [table \(see page 39\)](#) below. Additionally, there are several less critical nets assigned to a group called DDR_MISC.

All the signals in the address group must be length matched together and routed on the same layer and can be connected to two DIMM sockets. In the case of dual DIMM sockets, all trace lengths must be matched, as well as the distance from the I/C (FPGA) to the first DIMM socket. Exceptions to this requirement are the signals CKE, ODT, CS and CLK_DP/N. These signals connect to a single DIMM and should be on the same routing layer as the other address group signals, but simply minimized in length. Some of the CS nets can have two destinations and are treated similar to the address lines.

Each of the data groups are composed of sets of nine subgroups. Each of these subgroup is composed of eight data signals plus a differential clock as color coded in the [table \(see page 39\)](#) below. Therefore there are 11 nets in a data subgroup. Data nets can connect to 2 DIMMS, therefore matching between DIMMs is required as well as matching from the IC to the 1st DIMM. Each colored subgroup must be routed on the same layer.

High-Level Rules

1. Nets that use a transition via that traverses the entire thickness of the PCB should be routed to the closest DIMMs. When breaking out on the top layer, immediately transition to the bottom layer and route to the closest DIMM.
2. Set the DIMM-to-DIMM mechanical pitch to 400 mils (10 mm).
3. DDR3 signals routed on a microstrip or stripline layer must not be routed over plane splits for near a referencing plane. Care must be taken to ensure that the DDR3 DQ/DQS signals are routed over a continuous return path.
4. It is recommended to keep DDR3 routing away from components used in a switching regulator circuit and any non-DDR related routing and components. DDR3 routing near a high-noise voltage regulator via and shape requires at least 50 mils of spacing and at least 100 mils of spacing near high-noise voltage regulator traces .
5. At least 300 mils of spacing is needed between a DDR3 via and high-noise voltage regulator via. But if additional ground vias are added in between, this spacing can be reduced to 100 mils.
6. It is recommended to leave at least 8 mils of spacing for DDR3 routing and a referencing edge or via clearance.
7. Routing over a void or anti-pad is not allowed.
8. It is recommended that long memory channels should be routed on stripline layers and short channels on microstrip layers with only one channel on two layers on each side of the FPGA, for example, channel 1 routed on the top and bottom layers on one side of the FPGA and channel 3 on the top and bottom layers on the other side of the FPGA.
9. When using micro and/or blind vias, assuming no stubs are created, the above layer transition rules can be ignored.
10. Ground vias should be within 250 mils (or closer) of signal transition vias regardless of the via technology.

Table 5: Signals for the Channel East-North

Addr / Clk Group Chan EN	Data Group for Channel EN, Dimm Sockets Jxx and Jyy		
DDR_EN_A<0> DDR_EN_A<1> DDR_EN_A<2> DDR_EN_A<3> DDR_EN_A<4> DDR_EN_A<5> DDR_EN_A<6> DDR_EN_A<7> DDR_EN_A<8> DDR_EN_A<9> DDR_EN_A<10> DDR_EN_A<11> DDR_EN_A<12> DDR_EN_A<13> DDR_EN_A<14> DDR_EN_A<15> DDR_EN_BA<0> DDR_EN_BA<1> DDR_EN_BA<2> DDR_EN_CAS_N DDR_EN_RAS_N DDR_EN_WE_N DDR_EN_PAR_IN DDR_EN_CKE<0:3> DDR_EN_CS_N<0:3> DDR_EN_CLK_P /N<0:3> DDR_EN_ODT<0:3>	DDR_EN_DQ<0> DDR_EN_DQ<1> DDR_EN_DQ<2> DDR_EN_DQ<3> DDR_EN_DQ<4> DDR_EN_DQ<5> DDR_EN_DQ<6> DDR_EN_DQ<7> DDR_EN_DM<0> DDR_EN_DQS_DP<0> DDR_EN_DQS_DN<0> DDR_EN_DQ<8> DDR_EN_DQ<9> DDR_EN_DQ<10> DDR_EN_DQ<11> DDR_EN_DQ<12> DDR_EN_DQ<13> DDR_EN_DQ<14> DDR_EN_DQ<15> DDR_EN_DM<1> DDR_EN_DQS_DP<1> DDR_EN_DQS_DN<1> DDR_EN_DQ<16> DDR_EN_DQ<17> DDR_EN_DQ<18> DDR_EN_DQ<19> DDR_EN_DQ<20> DDR_EN_DQ<21> DDR_EN_DQ<22> DDR_EN_DQ<23> DDR_EN_DM<2> DDR_EN_DQS_DP<2> DDR_EN_DQS_DN<2>	DDR_EN_DQ<24> DDR_EN_DQ<25> DDR_EN_DQ<26> DDR_EN_DQ<27> DDR_EN_DQ<28> DDR_EN_DQ<29> DDR_EN_DQ<30> DDR_EN_DQ<31> DDR_EN_DM<3> DDR_EN_DQS_DP<3> DDR_EN_DQS_DN<3> DDR_EN_DQ<32> DDR_EN_DQ<33> DDR_EN_DQ<34> DDR_EN_DQ<35> DDR_EN_DQ<36> DDR_EN_DQ<37> DDR_EN_DQ<38> DDR_EN_DQ<39> DDR_EN_DM<4> DDR_EN_DQS_DP<4> DDR_EN_DQS_DN<4> DDR_EN_DQ<40> DDR_EN_DQ<41> DDR_EN_DQ<42> DDR_EN_DQ<43> DDR_EN_DQ<44> DDR_EN_DQ<45> DDR_EN_DQ<46> DDR_EN_DQ<47> DDR_EN_DM<5> DDR_EN_DQS_DP<5> DDR_EN_DQS_DN<5>	DDR_EN_DQ<48> DDR_EN_DQ<49> DDR_EN_DQ<50> DDR_EN_DQ<51> DDR_EN_DQ<52> DDR_EN_DQ<53> DDR_EN_DQ<54> DDR_EN_DQ<55> DDR_EN_DM<6> DDR_EN_DQS_DP<6> DDR_EN_DQS_DN<6> DDR_EN_DQ<56> DDR_EN_DQ<57> DDR_EN_DQ<58> DDR_EN_DQ<59> DDR_EN_DQ<60> DDR_EN_DQ<61> DDR_EN_DQ<62> DDR_EN_DQ<63> DDR_EN_DM<7> DDR_EN_DQS_DP<7> DDR_EN_DQS_DN<7> DDR_EN_DQ<64> DDR_EN_DQ<65> DDR_EN_DQ<66> DDR_EN_DQ<67> DDR_EN_DQ<68> DDR_EN_DQ<69> DDR_EN_DQ<70> DDR_EN_DQ<71> DDR_EN_DM<8> DDR_EN_DQS_DP<8> DDR_EN_DQS_DN<8>

Guidelines

Table 6: Data Subgroup Guidelines

For the DATA Subgroups	Topology: Daisy chain
Max trace length	4.2" (microstrip); 5.7" (stripline)
Reference	Ground referenced
Layer assignment	Microstrip or stripline (preferred)
Layer changes	Layer change only allowed from microstrip to stripline or to other microstrip at CPU pin-field
Layer transitions	Signals within a group DQ/DQS must be routed on the same layer
DQ/DM impedance	40Ω ±15% (microstrip), 38Ω ±10% (stripline)
DQ/DM trace spacing (channel-to-channel boundaries)	36 mils (microstrip); 42 mils (stripline)
DQ/DM trace spacing between DIMMs	12 mils (microstrip); 9 mils (stripline)
DQS trace spacing	5 mils
DQS single-ended impedance	40Ω ±15% (microstrip), 38Ω ±10% (stripline)
DQ/DM/DQS trace width	See Stack-up (see page 45) .
DQS to DQ/DM Trace Spacing:	28 mils (microstrip) 18 mils (stripline)
DQ/DM Trace Spacings	

For the DATA Subgroups	Topology: Daisy chain
Intra-byte lane in an open field between the FPGA and the closest DIMM slot	22.5 mils (microstrip); 18 mils (stripline),
Intra-byte lane in an open field	30 mils (microstrip); 33.6 mils (stripline)
Intra-byte lane in an open field	21 mils (microstrip); 18 mils (stripline)
At intra-byte lane boundaries	30 mils (microstrip); 33.5 mils (stripline)
Trace Lengths	
FPGA signal pad to first DIMM1 pin (includes FPGA package length)	Min = 1.5" (microstrip), Min = 2.0" (stripline). Max = 3.6" (microstrip), Max = 5.0" (stripline).
DIMM1 pin to DIMM2 pin length structure (length "B")	Min = 0.375", Max = 0.525". Impedance: 40Ω ±15% (microstrip) 38Ω ±15% (stripline). Nominal trace width: see Stack-up (see page 45) .
Length Matching	
DQ to DQS (includes FPGA package length from FPGA pad to each DIMM pin)	DQ signals must match to relevant DQS pair within 50 mils.
DQS pair	P and N signals must match to each other within 5 mils.
DQ to DQ (includes FPGA package length from FPGA pad to each DIMM pin)	Within a data subgroup, DQ signals should match to each other within 5 mils.
DIMM to DIMM (fly-by topology)	Routing from the first to second DIMM must be matched to within 5 mils. No length matching across DQ groups.

For the DATA Subgroups	Topology: Daisy chain
Length matching note	Should not be done in the FPGA breakout region, and that signals route directly from their pin to the exit of the pin field, where they attain the normal width/spacing. Avoid serpentine routing in this region. Spacing violation in breakout region cannot exceed 100 mils in length.
Breakout Requirements	
FPGA breakout requirements	See Stack-up (see page 45) for breakout areas. Micro and/or blind vias are permissible. If not specified in stack-up, use 4-mil width/4-mil spacing with two-track routing (track-to-track minimum of 17 mils) within BGA area. After BGA area, maintain two-track routing or spread with 4-mil width /8-mil spacing. 4-mil width/8-mil spacing is allowed for no more than 1 inch (stripline) or 500 mils (microstrip).
Total breakout routing	Cannot exceed 1000 mils for stripline and 500 mils for microstrip. This length should be minimized.

Below are detailed guidelines for the Source Clocked Address Signal Group. This group uses a Daisy Chain Topology.

Table 7: Source Clocked Address Signal Group Guidelines

For the Address Subgroups	Topology: Daisy chain
Max trace length	3.525"
Reference	DDR_VDD
Layer Assignment	microstrip or stripline (preferred)
Layer Changes	Signals within a channel must be routed on the same layer. Layer change only allowed from microstrip to stripline or to other microstrip at CPU pin-field
Address group Impedance	40Ω ±15% (microstrip), 38Ω ±10% (stripline)
Address group Single Ended Impedance	40Ω ±15% (microstrip), 38Ω ±10% (stripline)
Address group Trace Width	see stackup
Layer transitions	Signals within an address group are routed on the same layer.
Address group Spacing (intra-byte lanes) in Open Field (between FPGA and Closest DIMM Slot)	13.5 (microstrip) 12 (stripline) mils, 20 mils preferred.

For the Address Subgroups	Topology: Daisy chain
Address group Spacing (inter-byte lanes) in Open Field	30 mills (microstrip), 33.6 mills (stripline).
Address group Spacing (at intra-byte lane boundaries)	30 mills (microstrip) 33.5 mills (stripline).
Nominal Trace Spacing within the DIMM field	12 mills (stripline) 9 mills (microstrip).
Address group Spacing between DIMMs	12 miles (microstrip) 9 mills (stripline).
Trace Lengths	
FPGA Signal Pad to first DIMM1 Pin (includes FPGA package length) denoted as length "A"	3.0" max, 1" min
DIMM1 pin to DIMM2 pin length structure (length "A")	0.525" max, 0.375" minimum
Length Matching (includes FPGA package length from FPGA pad to each DIMM pin)	<ul style="list-style-type: none"> • Within an Address Group signals need to match the associated CLK signal diff pair within 50 mills. • Address group signal length to the 1st DIMM must match each other to within 20 mills.
DIMM to DIMM length matching(Fly by Topology)	<ul style="list-style-type: none"> • Routing from the 1st DIMM to the 2nd DIMM matched to within 5mills. • No length matching across different Address groups.
FPGA Breakout Requirements	See stackup for breakout areas. uVias or Blind vias are permissible. If not specified in stack-up, use 4-mil width/4-mil spacing with 2 track routing (track to track minimum of 17 mills) within BGA area. After BGA area maintain 2 track routing or spread with 4-mil width/8-mil spacing. 4-mil width/8-mil spacing is allowed for no more than 1 inch (stripline) or 500 mills (microstrip).
Total breakout routing	Cannot exceed 1000 mills for stripline and 500 mills for microstrip. This length should be minimized.

For the Address Subgroups	Topology: Daisy chain
Length matching note	<ul style="list-style-type: none"> • Should not be done in the FPGA breakout region, and that signals route directly from their pin to the exit of the pin field where they attain the normal width/spacing. Avoid serpentine routing in this region. • Spacing violation in breakout region cannot exceed 100mils in length • Length matching offsets caused by the IC Package delays must be accommodated.

Table 8: Address Bus Control Related Nets: DDR_[all] BAx, DDR[all] CKEx, DDR[all] CSx_N, DDR[all] _ODTx

For the BA, CKE, CS, ODT & Subgroups	Topology: Daisy chain
Max trace length	3.525"
Reference	DDR_VDD
Layer Assignment	microstrip or stripline (preferred)
Layer Changes	Signals within a channel must be routed on the same layer. Layer change only allowed from microstrip to stripline or to other microstrip at CPU pin-field
Address group Impedance	40Ω ± 15% (microstrip), 38Ω ± 10% (stripline)
Address group Single Ended Impedance	40Ω ± 15% (microstrip), 38Ω ± 10% (stripline)
Address group Trace Width	see stackup
Layer transitions	Signals within an address group are routed on the same layer
Address group Spacing (intra-byte lanes) in Open Field (between FPGA and Closest DIMM Slot)	13.5 (microstrip) 12 (stripline) mills, 20 mills preferred
Address group Spacing (inter-byte lanes) in Open Field	30 (microstrip), 33.6 (stripline) mills
Address group Spacing (at intra-byte lane boundaries)	30 (microstrip) 33.5 (stripline) mills

For the BA, CKE, CS, ODT & Subgroups	Topology: Daisy chain
Nominal Trace Spacing within the DIMM field	12 mils (stripline) 9 mils (microstrip)
Address group Spacing between DIMMs	12 miles (microstrip) 9 mils (stripline)
Trace Lengths	
FPGA Signal Pad to first DIMM1 Pin (includes FPGA package length) denoted as length "A"	3.0 inches max, 1 inch min
DIMM pin to DIMM pin denoted as "B"	0.525 inches max, 0.375 inches minimum
Length Matching (includes FPGA package length from FPGA pad to each DIMM pin)	<ul style="list-style-type: none"> • Considered as part of the Address Group signals & need to match the associated CLK signal diff pair within 50 mills • Considered as part of the Address group signals & must match each other to within 25 mills
DIMM to DIMM length matching(Fly by Topology where 2 DIMMs are involved)	<ul style="list-style-type: none"> • Routing from the 1st DIMM to the 2nd DIMM matched to within 5 mills • No length matching across Address groups
FPGA Breakout Requirements	See stackup for breakout areas. uVias or Blind vias are permissible. If not specified in stack-up, use 4-mil width/4-mil spacing with 2 track routing (track to track minimum of 17 mils) within BGA area. After BGA area maintain 2 track routing or spread with 4-mil width/8-mil spacing. 4-mil width/8-mil spacing is allowed for no more than 1 inch (stripline) or 500 mils (microstrip).
Total breakout routing	Cannot exceed 1000 mils for stripline and 500 mils for microstrip. This length should be minimized
Length matching note	should not be done in the FPGA breakout region, and that signals route directly from their pin to the exit of the pin field, where they attain the normal width/spacing. Avoid serpentine routing in this region. Spacing violation in breakout region cannot exceed 100mils in length

These nets coincide with the Address Groups and are point to point: `DDR_[all]CLKx_P/N`, `DDR[all]_SODIMMx_y`

Table 9: DDR3 Clock Signal Group Guidelines

For the Clock Differential Pairs	Topology: Point to Point
Max trace length	3.525"
Reference	DDR_VDD
Layer Assignment	microstrip or stripline (preferred)
Layer Changes	Signals within a channel must be routed on the same layer. Layer change only allowed from microstrip to stripline or to other microstrip at CPU pin-field
Clock Signal single ended net Impedance	40Ω ± 15% (microstrip), 38Ω ± 10% (stripline)
Clock Group P to N length matching	2 mills
Clock Group Trace Width	see stackup
Layer transitions for Clocks within Address Groups	Signals within an Address Group are routed on the same layer
Trace Spacing in Open Field (between FPGA and Closest DIMM Slot) amongst the other Address Group nets	15 (microstrip) 13 (stripline) mills, 20 mills preferred
Clock group Spacing (inter-byte lanes) in Open Field	30 (microstrip), 33.6 (stripline) mills
Clock group Spacing between DIMMs	30 miles (microstrip) 33.6 mils (stripline)
Trace Lengths	
FPGA Signal Pad to first DIMM1 Pin length structure (includes FPGA package length) denoted as length "A"	3.0 inches max, 1 inch min
DIMM pin to DIMM pin denoted as "B"	0.525 inches max, 0.375 inches minimum

For the Clock Differential Pairs	Topology: Point to Point
Length Matching (includes FPGA package length from FPGA pad to each DIMM pin)	<ul style="list-style-type: none"> • CLKx_[P/N] and CLKy_[P/N] should match within 5 mils for the same DIMM • CLKs to be 10 mils longer than longest Address/Ctrl trace • No length matching across Address groups
FPGA Breakout Requirements	See stackup for breakout areas. uVias or Blind Vias are permissible. If not specified in stack-up, use 4-mil width/4-mil spacing with 2 track routing (track to track minimum of 17 mils) within BGA area. After BGA area maintain 2 track routing or spread with 4-mil width /8-mil spacing. 4-mil width/8-mil spacing is allowed for no more than 1 inch (stripline) or 500 mils (microstrip).
Total breakout routing	Cannot exceed 1000 mils for stripline and 500 mils for microstrip. This length should be minimized
Length matching note	Should not be done in the FPGA breakout region, and that signals route directly from their pin to the exit of the pin field, where they attain the normal width/spacing. Avoid serpentine routing in this region. Spacing violation in breakout region cannot exceed 100mils in length

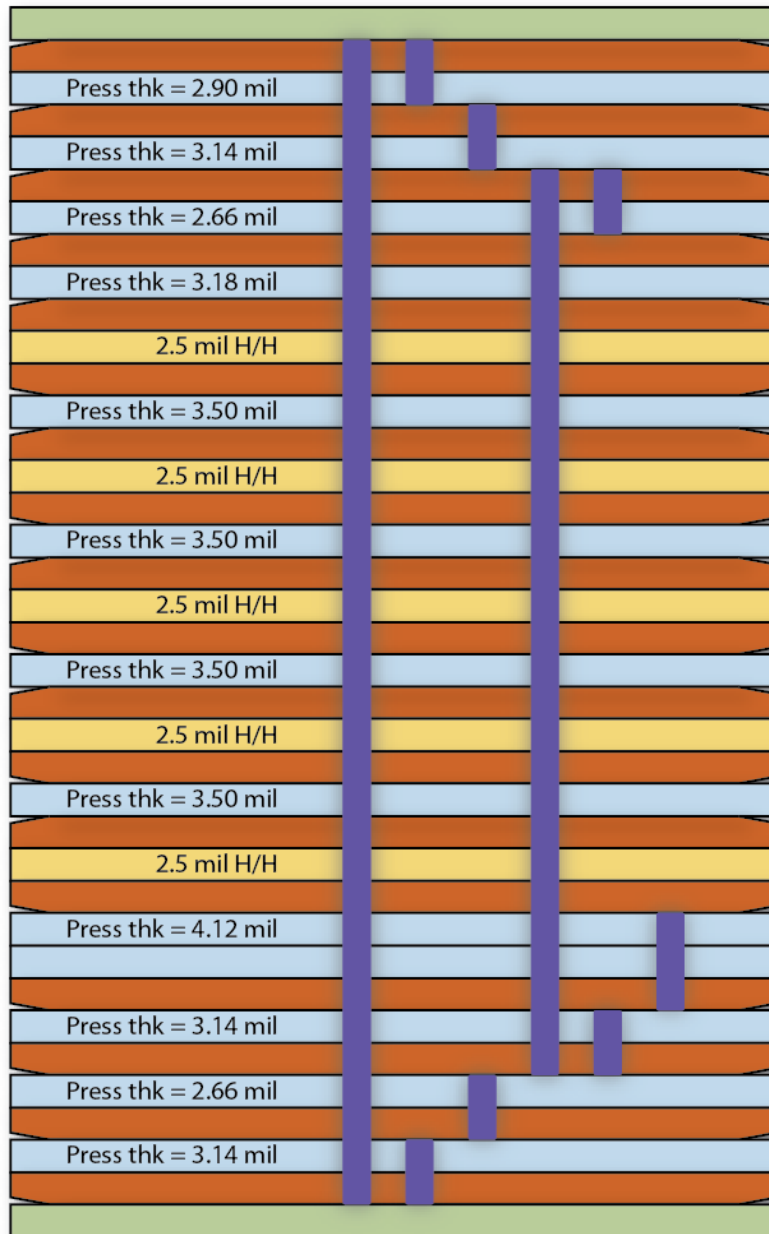
Below are listed nets that are in the DDR_MISC group for a given channel.

Table 10: DDR_MISC Group Guidelines

For Signal Group DDR_Misc	
DDR_[all]_ERR_OUT_N DDR_[all]_EVENT_N DDR_[all]_I2C_SCL DDR_[all]_I2C_SDA DDR_[all]_PAR_IN DDR_[all]_RESET_N DDR_[all]_SCL_1 DDR_[all]_SCL_2 DDR_[all]_SDA_1 DDR_[all]_SDA_2 DDR_[all]_SODIMM_SCL DDR_[all]_SODIMM_SDA	These nets should be routed on the same layer as the Address Group and simply minimized in length. Each of the channels should mimic these basic rules.

Stack-up

The following figure illustrates the recommend board stack-up. Details on each layer are provided in the table following.



3048261-01-2015.11.03

Figure 16: Board Stack-up

Table 11: Layer Stack-up Details

Layer	Type	CU Weight	CU %	Material Description	Segment	Glass Style	Material Family	Dielectric Constant @ 1GHz	Thickness After Lamination (Mils)
Soldermask									0.80
1	Plane	T	80		Foil				1.25
				Press thk = 2.90 mil	Prepreg	1080 HRC (71)	370 HR	3.77	2.90
2	Mixed	H	50		Foil				0.80
				Press thk = 3.14 mil	Prepreg	1080 HRC (71)	370 HR	3.77	3.14
3	Plane	H	80		Foil				0.80
				Press thk = 2.66 mil	Prepreg	1080 HRC (71)	370 HR	3.77	2.66
4	Signal	H	20		Foil				0.80
				Press thk = 3.18 mil	Prepreg	1080 HRC (71)	370 HR	3.77	3.18
5	Plane	H	80						0.60
				2.5 mil H/H	Core	1-1080	370 HR	4.04	2.50
6	Signal	H	20						0.60
				Press thk = 3.50 mil	Prepreg	2113 (59)	370 HR	4.02	3.50
7	Plane	H	80						0.60
				2.5 mil H/H	Core	1-1080	370 HR	4.04	2.50
8	Signal	H	20						0.60
				Press thk = 3.50 mil	Prepreg	2113 (59)	370 HR	4.02	3.50

Layer	Type	CU Weight	CU %	Material Description	Segment	Glass Style	Material Family	Dielectric Constant @ 1GHz	Thickness After Lamination (Mils)
9	Plane	H	80						0.60
				2.5 mil H/H	Core	1-1080	370 HR	4.04	2.50
10	Signal	H	20						0.60
				Press thk = 3.50 mil	Prepreg	2113 (59)	370 HR	4.02	3.50
11	Plane	H	80						0.60
				2.5 mil H/H	Core	1-1080	370 HR	4.04	2.50
12	Signal	H	20						0.60
				Press thk = 3.50 mil	Prepreg	2113 (59)	370 HR	4.02	3.50
13	Plane	H	80						0.60
				2.5 mil H/H	Core	1-1080	370 HR	4.04	2.50
14	Signal	H	20						0.60
				Press thk = 4.12 mil	Prepreg	106(75)	370 HR	3.75	4.12
						106(75)	370 HR	3.75	
15	Plane	H	80		Foil				0.80
				Press thk = 3.14 mil	Prepreg	1080 HRC (71)	370 HR	3.77	3.14
16	Signal	H	20		Foil				0.80
				Press thk = 2.66 mil	Prepreg	1080 HRC (71)	370 HR	3.77	2.66
17	Plane	H	80		Foil				0.80

Layer	Type	CU Weight	CU %	Material Description	Segment	Glass Style	Material Family	Dielectric Constant @ 1GHz	Thickness After Lamination (Mils)
				Press thk = 3.14 mil	Prepreg	1080 HRC (71)	370 HR	3.77	3.14
18	Plane	T	80		Foil				1.25
Soldermask									0.80

Table 12: Impedance Table

Layer	Impedance Spec (Ω)	Tolerance (Ω)		Type [†]	Upper Ref	Lower Ref	Designed		Finished		Impedance Sim (Ω)
		+	-				Line Width (Mils)	Spacing (Mils)	Line Width (Mils)	Spacing (Mils)	
(Ω)1	100	10	10	Diff		2	4	8.00	4	8.00	99.732
2	100	10	10	Diff	1	3	2.4	9.60	2.5	9.50	99.913
2	50	5	5	Single	1	3	2.6		2.6		49.368
2	85	8.5	8.5	Diff	1	3	3.4	8.60	3.5	8.50	84.871
2	40	4	4	Single	1	3	4		4		39.835
4	100	10	10	Diff	3	5	2.4	9.60	2.4	9.60	99.760
4	50	5	5	Single	3	5	2.6		2.6		48.346
4	85	8.5	8.5	Diff	3	5	3.4	8.60	3.4	8.60	84.286
4	40	4	4	Single	3	5	4		4		38.877
6	100	10	10	Diff	7	5	2.4	9.60	2.4	9.60	99.722
6	50	5	5	Single	7	5	2.6		2.6		48.250
6	85	8.5	8.5	Diff	7	5	3.4	8.60	3.4	8.60	83.901
6	40	4	4	Single	7	5	4		4		38.603
8	100	10	10	Diff	9	7	2.4	9.60	2.4	9.60	99.722
8	50	5	5	Single	9	7	2.6		2.6		48.250
8	85	8.5	8.5	Diff	9	7	3.4	8.60	3.4	8.60	83.901
8	40	4	4	Single	9	7	4		4		38.603
10	100	10	10	Diff	11	9	2.4	9.60	2.4	9.60	99.722
10	50	5	5	Single	11	9	2.6		2.6		48.250
10	85	8.5	8.5	Diff	11	9	3.4	8.60	3.4	8.60	83.901
10	40	4	4	Single	11	9	4		4		38.603
12	100	10	10	Diff	13	11	2.4	9.60	2.4	9.60	99.722
12	50	5	5	Single	13	11	2.6		2.6		48.250
12	85	8.5	8.5	Diff	13	11	3.4	8.60	3.4	8.60	83.901
12	40	4	4	Single	13	11	4		4		38.603

Layer	Impedance Spec	Tolerance (Ω)		Type†	Upper Ref	Lower Ref	Designed		Finished		Impedance Sim
14	100	10	10	Diff	15	13	2.4	9.60	2.625	9.38	99.722
14	50	5	5	Single	151	13	2.6		2.6		50.403
14	85	8.5	8.5	Diff	15	13	3.4	8.60	3.625	8.38	84.794
14	40	4	4	Single	15	13	4		4		40.548
16	100	10	10	Diff	17	15	2.4	9.60	2.4	9.60	99.470
16	50	5	5	Single	17	15	2.6		2.6		48.196
16	85	8.5	8.5	Diff	17	15	3.4	8.60	3.4	8.60	84.014
16	40	4	4	Single	17	15	4		4		38.738
18	100	10	10	Diff		17	4	8.00	4.25	7.75	100.309

Table Notes



† Under type, "single" mean single ended, "diff" means differential.

Chapter - 7: PHY Architecture

Overview

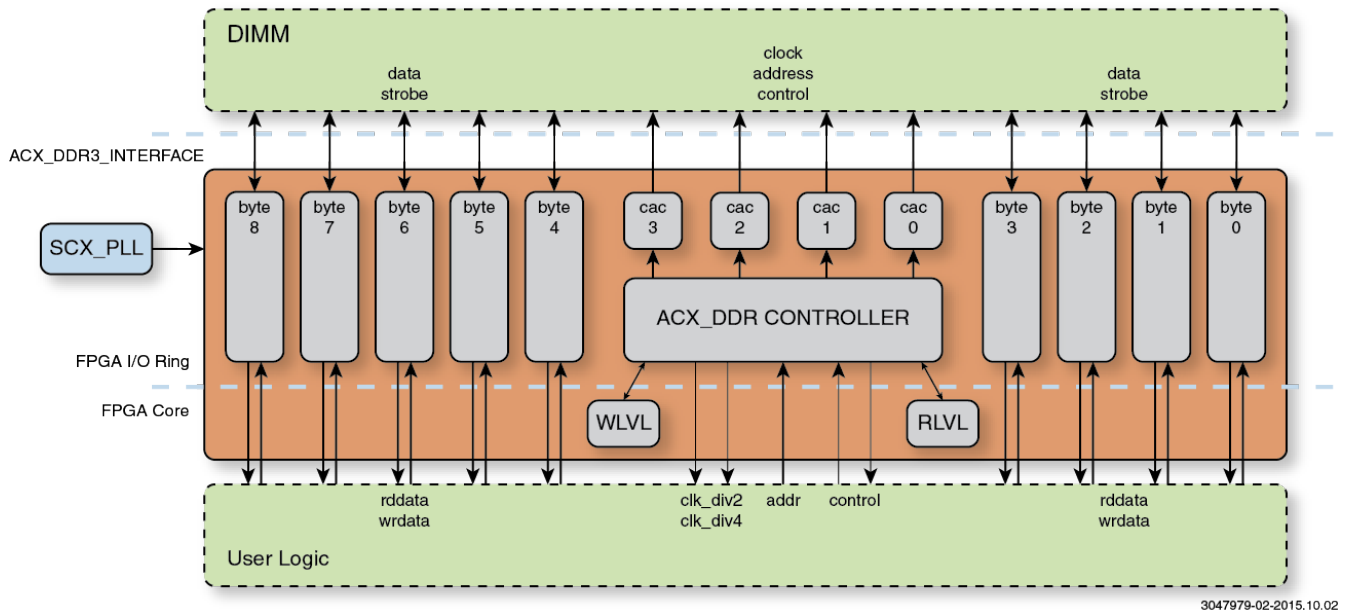


Figure 17: DDR3 PHY Architecture

Clock and Reset Architectures

Overview

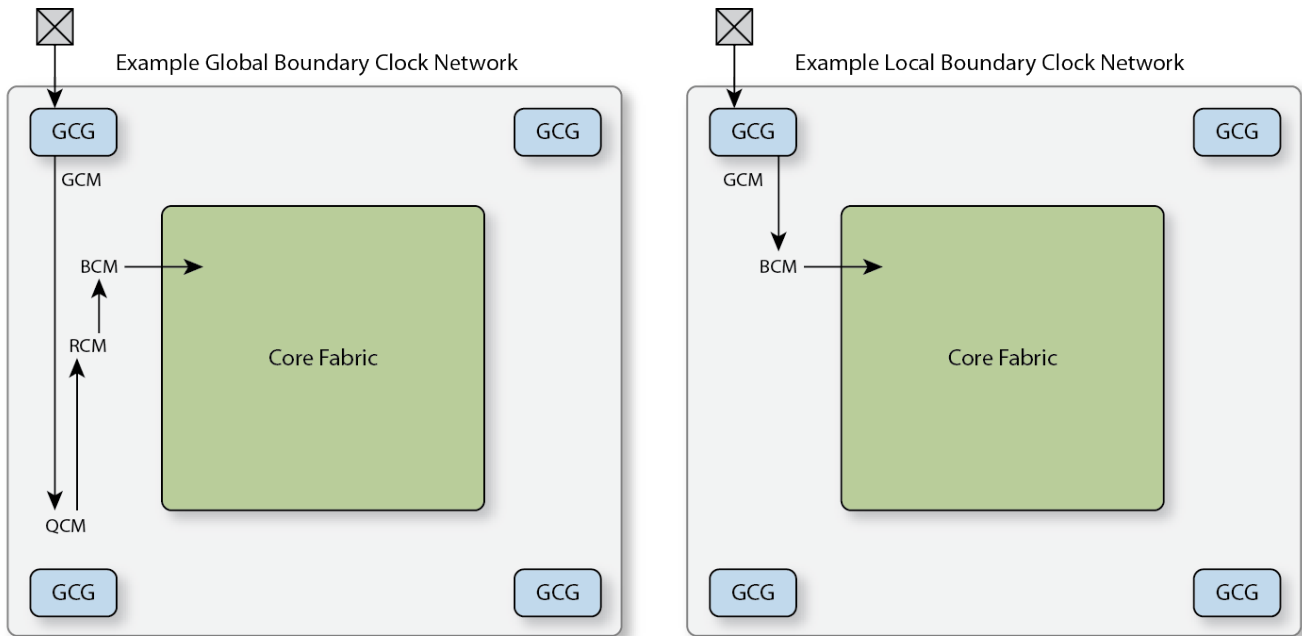
Speedster22iHD FPGAs have two hierarchical clock networks: a core clock network and a boundary clock network. The core clock network is the hierarchical network that feeds resources in the FPGA fabric. The boundary clock network is a fully programmable clock network in the I/O ring, unique to Speedster22iHD FPGAs, that provides for significant advantages when clocking I/O ring resources at high frequencies.

In addition, there is a reset network that channels signals generated internally or coming from GPIO are funneled through the reset input blocks (RIBs) in the device corners into the FPGA core and the I/O ring.

Clocks

Boundary Clock Networks

The boundary clock network is comprised of a low-skew global boundary clock network and a lower jitter local boundary clock network. The global boundary clock network ensures that clocks coming in from all four sides of the device traverse a muxing network that provides for low skew between them. The local boundary clock network is a shortened and more direct low-jitter path from the clock generators (CGs) to I/O ring resources requiring clocking.



3048342-08.2015.10.29

Figure 18: Distinction Between Global and Local Boundary Clock Networks

The table below shows which clock corners need to be used to ensure local boundary clocking for each of the six DDR3 interfaces. Essentially, north corners can provide local clocks to north and center interfaces on that side, and south corners can only provide local clocks to south interfaces on the same side.

Table 13: Local Boundary Clock Support for DDR3 Interfaces

DDR3 Interface	Clock Corner Needed for Local Boundary Clocking
West-North (WN)	Northwest (NW)
West-Center (WC)	Northwest (NW)
West-South (WS)	Southwest (SW)
East-North (EN)	Northeast (NE)
East-Center (EC)	Northeast (NE)
East-South (ES)	Southeast (SE)

DDR PHY Clock Architecture

The DDR PHY accepts a single master clock input, which is routed to the DDR.

- Inside the DDR PHY, there are clock dividers that generate the half-rate and quarter-rate clocks.
- These clocks are used for the PHY and controller logic and are also forwarded to the programmable logic fabric for the user interface. These clocks are named `ddr_int_clk_div_2` and `ddr_int_clk_div4`.

- The half-rate and quarter-rate clocks are byte-lane clocks that can only reach logic within the three clock regions that correspond to that DDR3 interface.
- All the PHY fabric interface signals are timed and synchronous to these two clock outputs.

The single master DDR clock input must not be routed to any other locations within the fabric other than the DDR PHY.

Resets

DDR PHY Reset Architecture

The DDR PHY has three reset inputs:

- `reset_dds_phy_n` – To reset the hard PHY. This signal requires one of the I/O ring reset resources. This reset should not be routed to any other locations, particularly within the fabric.
- `reset_dds_ctrlr_n` – To reset the PHY controller. In addition, this signal feeds the resets of the controller logic, including the leveling circuits, in the core fabric. This signal requires one of the I/O ring reset resources. This reset must be released within five cycles of `reset_dds_phy_n`.
- `reset_dll_n` – To reset the DLL used for I/O timings. This signal requires one of the I/O ring reset resources. In current designs this reset has been left inactive.

Speedster22i Core-Generated I/O Resets

There are a total of 16 core-generated I/O resets available in the entire device (four in each corner). As the name implies, resets provided from the core to the I/O ring must pass through one of these resources. These resources must be shared as much as possible within the I/O ring. Every DDR3 interface requires two resets: one for the general PHY/controller logic, and one specifically for the read-leveling state-machine. Since other circuitry in the user design, such as the PCIe, Ethernet, Interlaken or other SerDes interfaces also require core-generated I/O resets, minimizing this number to stay within the limit of 16 total core-generated I/O resets, may be an important consideration in architectural decisions.

PHY/Controller Reset Chains and Timing Closure

As mentioned above, the DDR3 PHY library has separate reset ports for the PHY (logic in the I/O ring) and controller (logic in the fabric). For timing closure purposes, it will be necessary to make decisions about whether these resets are shared and how many reset chains there will be in a design.

Read-Leveling Reset Sharing

Successful reset sharing and controller initialization requires:

- Appropriate setting of the `GENERATE_RL_RESET` parameter is required. It must be set to 1'b1 for reset domains with a single DDR3 interface and to 1'b0 for domains with multiple interfaces where resets should be shared.
- Read-leveling reset outputs generated in the PHY library should be ANDed and provided to the read-leveling reset input port for all reset domains with multiple DDR3 interfaces where resets need to be shared. For reset domains with a single DDR3 interface, these input and output ports/signals are ignored. The `GENERATE_RL_RESET` parameter setting determines whether the reset comes from the internally generated logic (1'b1) or whether it comes from the reset input port (1'b0).
- The read-leveling state machines in the DDR3 interfaces inside a reset domain need to be sequenced. The easiest way to perform this sequencing is to tie the enable port for the state machine in the DDR3 interface to the previous interface's controller initialization done signal.

- The user logic in the core fabric should remain in reset until all controllers have completed initialization. The very last controller initialization done signal can be used as a user logic enable signal for this purpose.

The figure below provides an illustration of how this scheme can be implemented.

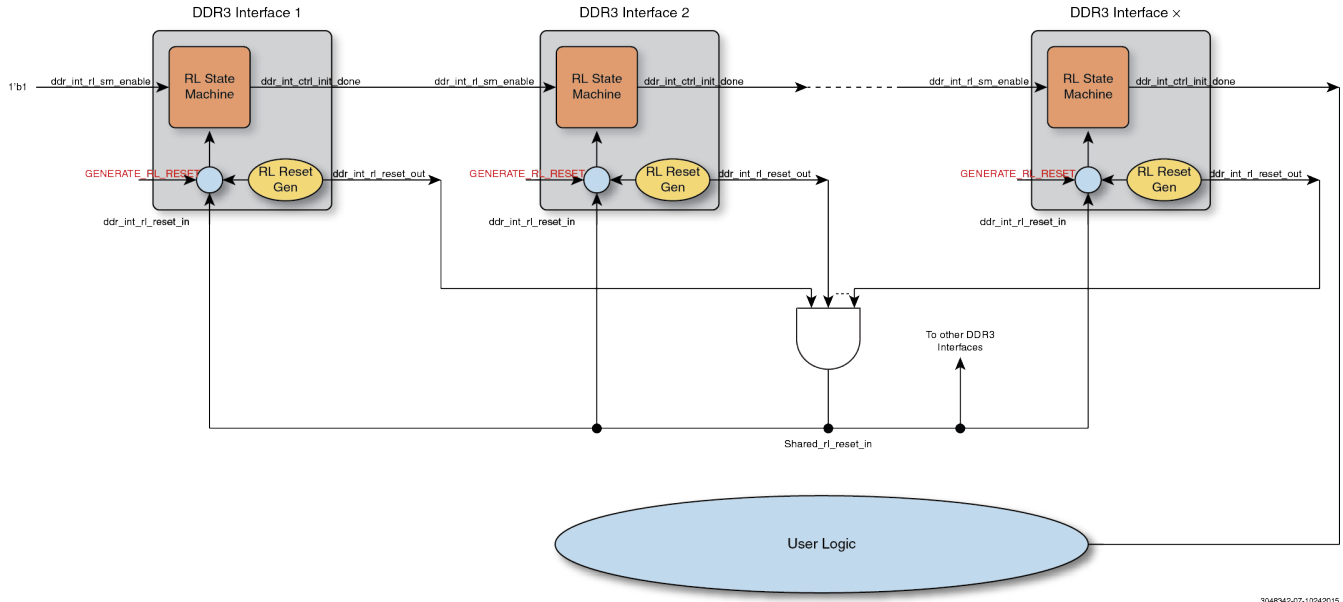


Figure 19: Read-Leveling Reset Sharing and Initialization Sequencing Scheme

For more information on the clock and reset networks, refer to *Speedster22i Clock and Reset Networks User Guide*

Example Use Cases

While a single-controller design is useful for debug, bring-up and in general, understanding the interfaces, most user designs need to use a larger number of DDR3 interfaces to satisfy the data buffering requirements. The most common cases are designs utilizing four or all six DDR3 interfaces. This article explores four variants of the multi-controller designs:

- Four-controller design with interfaces placed in the north and center (NC) locations: west-north, west-center, east-north and east-center. Local boundary (low-jitter) clocks are used for all interfaces.
- Four-controller design with interfaces placed in the south and center (SC) locations: west-south, west-center, east-south and east-center. Local boundary (low-jitter) clocks are used for all interfaces.
- Six-controller design consuming all the DDR3 interfaces and using local boundary (low-jitter) clocks.
- Six-controller design consuming all DDR3 interfaces and using a minimal number of clock and reset resources. Global boundary clocks are used for all interfaces.

Before diving into the architectures in each of the four design variants, there are five metrics that should be kept in mind at all times. A table is provided for each design variant to illustrate how these metrics differ across the designs. Since there are trade-offs when making these decisions, users need to determine which architecture works best for them, given their board and other design constraints. These five metrics are:

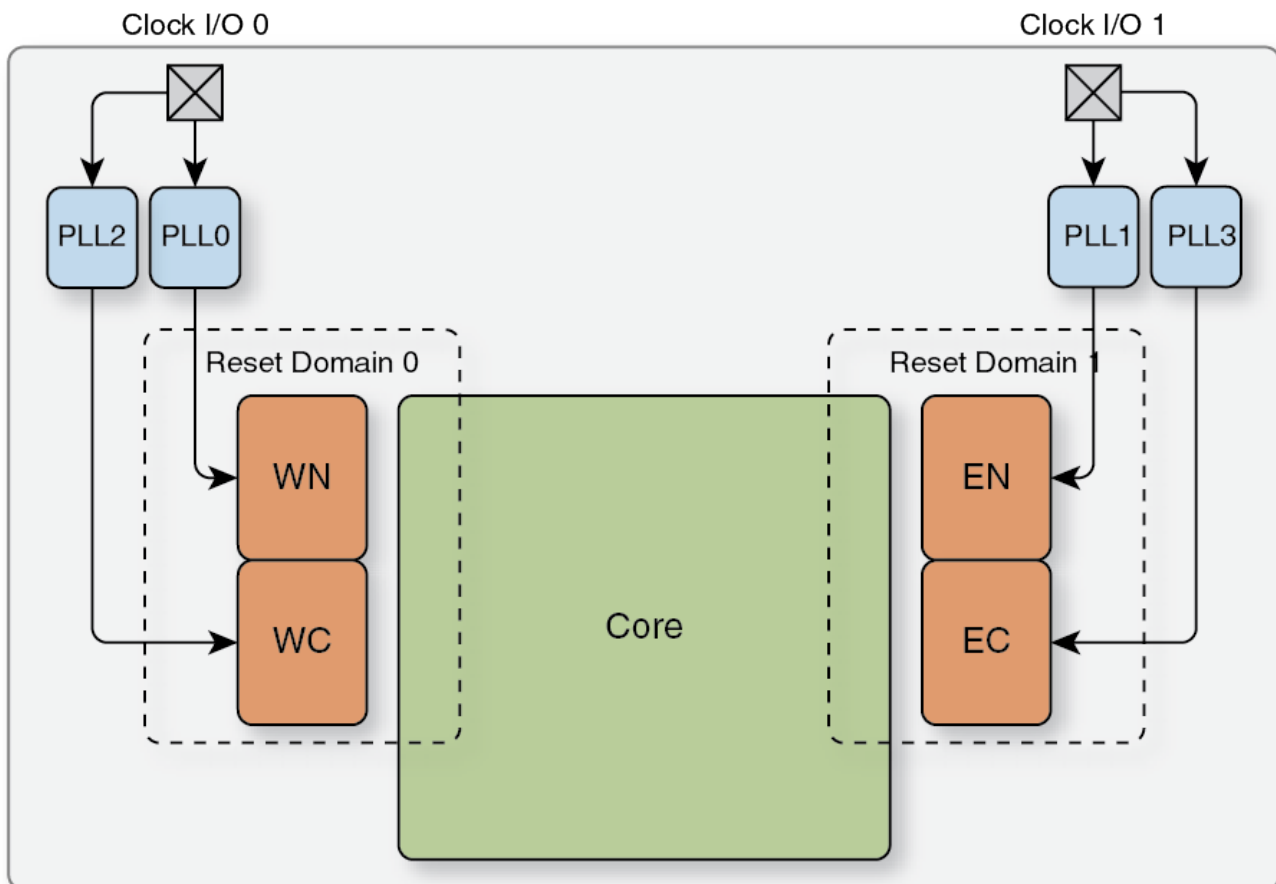
- **DDR3 interfaces** – The more controllers a design uses, the more challenging it is as far as resource consumption and timing closure.
- **Reference clock I/O** – Depending on the architecture, this can be 1, 2 or 4. These input clocks are recommended to be 100 MHz or 125 MHz. The first three designs in the list above show how use-cases

for minimal numbers of reference clock I/O can be used while still providing local boundary clocks to the DDR3 interfaces. The fourth design shows how a single reference clock and global boundary clocks can be used to provide clocks to all six DDR3 interfaces.

- **Local versus global boundary clocks** – The preference is always to use local boundary (low-jitter) clocks but this choice is dependent on clock I/O and DDR3 interface placement.
- **Core-generated I/O resets** – These resources must be shared as much as possible within the I/O ring to ensure that the design does not try to use up more than the 16 available in the HD1000 device. Minimizing this number may be an important consideration in architectural decisions.
- **PHY/controller reset chains and timing closure:** For timing closure purposes, it will be necessary to make decisions about whether these resets are shared and how many reset chains there will be in a design.

Four-Controller NC Design

A north and center placement is the preferred implementation for a four-controller DDR3 design, primarily because it minimizes the need for clock I/O and core-generated I/O resets when using local boundary clocks. The figure below illustrates the clock I/O placement, clock routing and reset sharing domains for this architecture. Each reset sharing domain requires sequencing the controller initialization and chaining/ANDing read-leveling reset signals. Refer to the [Six-Controller Design with Shared Resets](#) (see page 60) for details.



3048342-03-10272015

Figure 20: Four-Controller NC Design Architecture

For the PHY/controller reset chains, each reset domain has its own PHY reset signal, and each DDR3 interface also has its own controller reset. The full set of resource metrics in the four-controller NC design with local boundary clocks is provided in the table below.

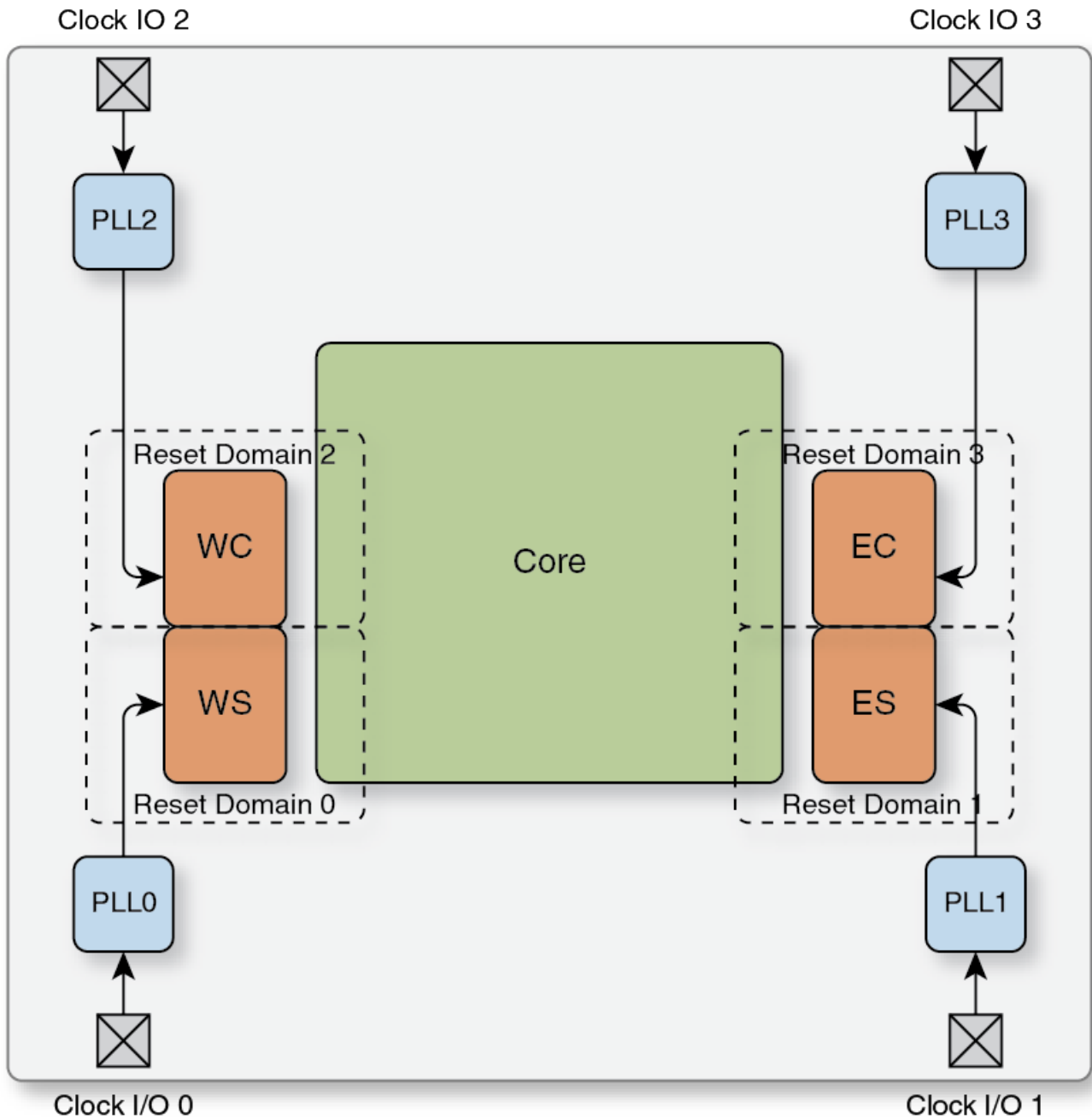
Table 14: Resource Metrics for the Four-Controller NC Design

Metric	Value
DDR3 interfaces	4
Reference clock I/O	2
Clocking	Local
Core-generated I/O resets	4
PHY/controller reset chains	Independent: PHY – 2 chains; controller – 4 chains

Four-Controller SC Design

The four-controller south and center design is similar to the north and center design in many ways; however, it is not as efficient as far as resource usage for local boundary clock usage implementations. Clocks to the center DDR3 interfaces can only be routed using local boundary paths from the north clock bank corners.

The figure below shows how the SC architecture differs from the NC one. A separate clock pin, PLL and reset domain is needed for each controller, doubling the number of core-generated I/O resets. In essence, each DDR3 interface behaves independently of the others.



3048342-04-10272015

Figure 21: Four-Controller SC Design Architecture

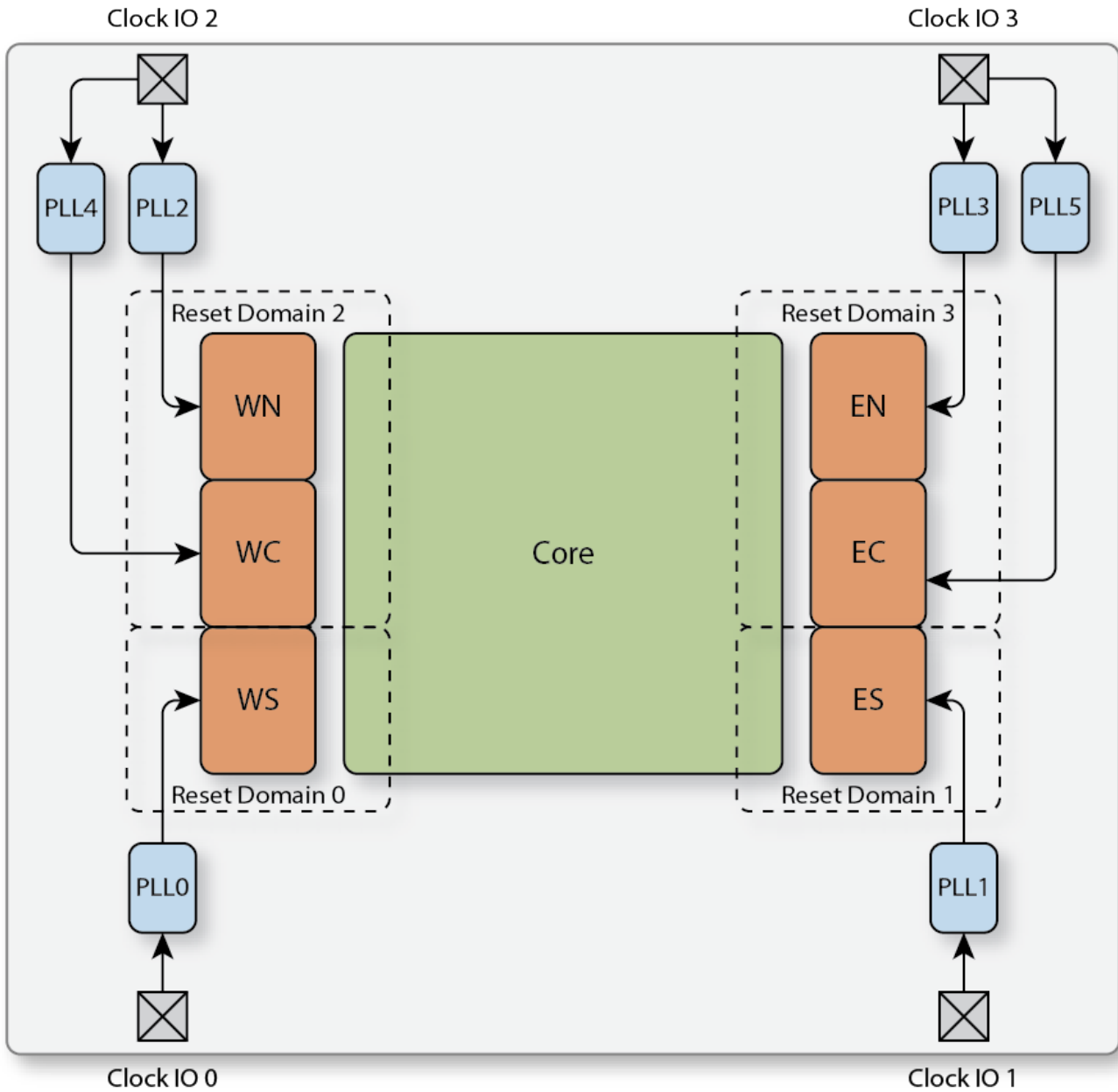
For the PHY/controller reset chains, each interface has a separate PHY reset signal and a different one for the controller reset. The full set of resource metrics in the four-controller SC design with local boundary clocks is provided in the table below.

Table 15: Resource Metrics for the Four-Controller SC Design

Metrics	Values
DDR3 interfaces	4
Reference clock I/O	4
Clocking	Local
Core-generated I/O resets	8
PHY/controller reset chains	Independent: PHY – 4 chains; controller – 4 chains

Six-Controller Design

The six-controller design combines the elements from the four-controller NC and SC designs. All six controllers are used, and in order to ensure local boundary clocks for all interfaces, four clock I/O and six PLLs are needed. Due to the north and center interfaces being able to share reset domains, the number of core-generated I/O resets does not increase in comparison to the four-controller SC design.



3048342-05-01082016

Figure 22: Six-Controller Design Architecture

For the PHY/controller reset chains, the north and center again can share PHY resets, but each of the six controllers must have its own controller reset. The full set of resource metrics in the six-controller design with local boundary clocks is provided in the table below.

Table 16: Resource Metrics for the Six-Controller Design

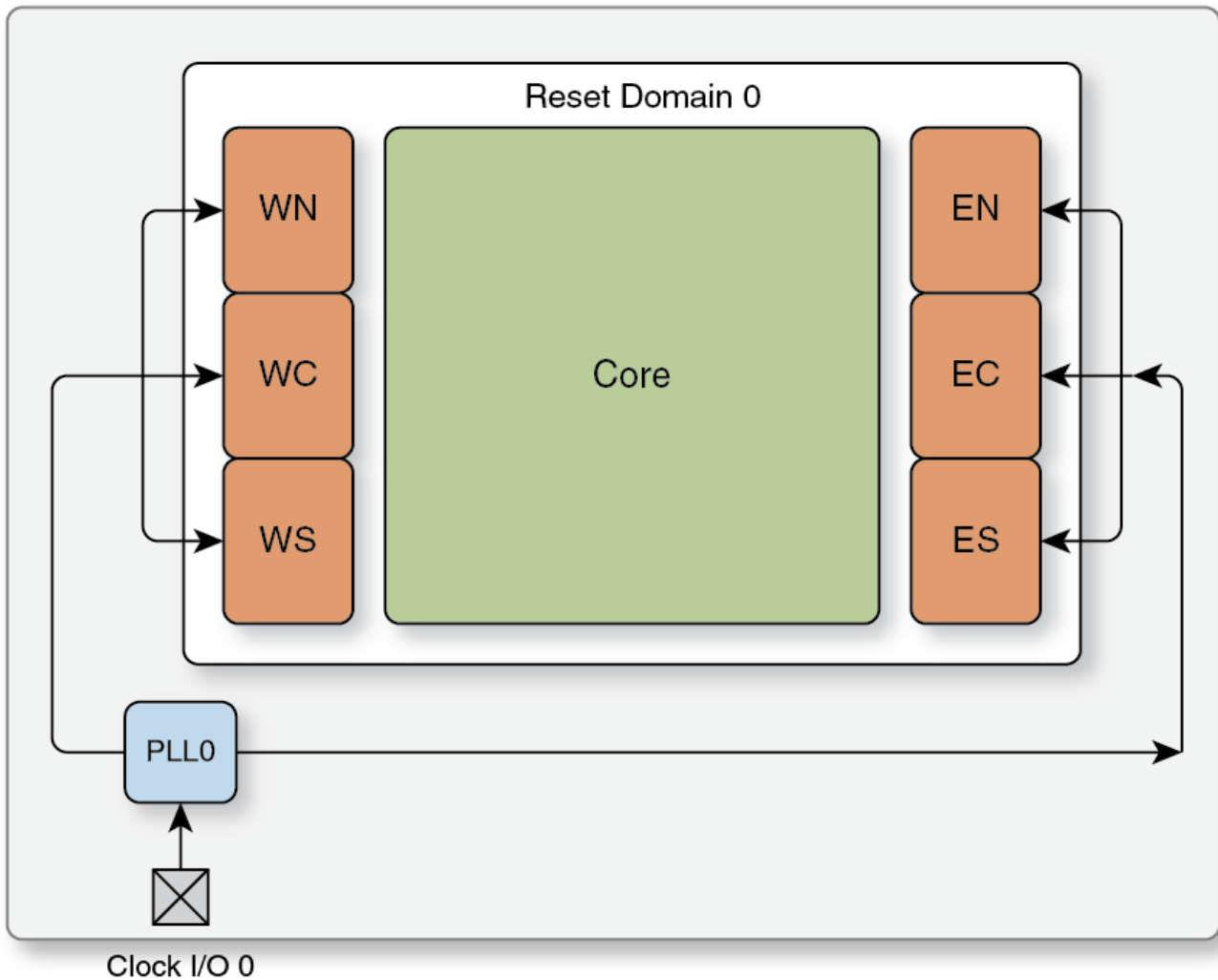
Metrics	Values
DDR3 interfaces	6
Reference clock I/O	4
Clocking	Local
Core-generated I/O resets	8
PHY/controller reset chains	Independent: PHY – 4 chains; controller – 6 chains

Six-Controller Design with Shared Resets

The six-controller design with shared resets may be the most desirable DDR3 architectural implementation to use, especially in larger user designs that require clock bank I/O and core-generated I/O resets be available for other functionality. The concepts presented here can be applied for other variants as well. In other words, there is nothing preventing a four-controller NC or SC design with shared resets. Alternatively, a four controller design could also be constructed from the north and south interfaces, with the center I/O used for other functionality.

This design brings in a single clock input and uses a single PLL. The PLL's output clock then traverses the global boundary clock network to reach all six DDR3 interfaces. Since global boundary clocking *has* to be used in this implementation, the clock bank I/O and PLL pair can be placed in any of the four corners, but do need to be placed in the same corner.

The figure below shows the architecture for the six-controller design with shared resets, assuming that the clock I/O and PLL are placed in the SW corner.



3048342-06-10272015

Figure 23: Six-Controller Design with Shared Resets

For the PHY/controller reset chains, the PHY reset needs to be shared across all controllers. Each interface still has its own controller reset with multiple pipeline stages. Without these details, closing timing in the fabric at the higher data rates becomes a significant challenge.

For this implementation, there are only two core-generated I/O resets since there is only one reset domain. The sharing of the PHY resets across all six controllers and the read-leveling state machine reset sharing scheme are necessary to ensure a single reset domain.

Table 17: Resource Metrics for the Six-Controller Design with Shared Resets

Metrics	Values
DDR3 interfaces	6
Reference clock I/O	1
Clocking	Global

Metrics	Values
Core-generated I/O resets	2
PHY/controller reset chains	Independent: PHY – 1 chain; controller – 6 chains

DLL and Write/Read Leveling

DLL Specs and Operation

The DLL IP block in the Speedster22i HD1000 is a wide-range DLL with one master DLL (MDLL) and 12 slave DLLs (SDLLs). The table below provides the DLL IP specifications and Figure 10 provides a high-level block diagram of the DLL architecture.

Table 18: DLL IP Specifications

Performance Parameters	Data
Frequency range	311 MHz – 1066 MHz
Maximum peak-to-peak period jitter at 2133 MHz with noise frequency = 200 Mhz and ± 15 mV sinusoidal noise	< 2% of cycle time
Minimum high-low slave pulse width	25% reference cycle
DLL lock time	< 500 reference clock cycles
SDLL step size	360/64° nominal
Output phase accuracy	$\pm 4\%$ reference clock cycle
Output phase resolution	6 bits
Slave delay adjustment	0% to 100% of reference cycle
Number of outputs per lane	1
Number of lanes per master	12
Reference Input Duty Cycle	40% – 60%

The MDLL uses a regulated supply generated by a high-performance, on-board regulator to achieve the best possible performance in terms of jitter. It receives a clock as its reference to generate the desired delay in its delay cells. The delay cells used in its voltage-controlled delay line (VCDL) is based on a current-starved technique to provide the delay to generate the feedback signal. The phase of the feedback signal is then compared with the reference signal. This phase difference is translated to voltage (PBIAS and NBIAS) by the phase detector and charge pump, which is given back to the VCDL block to generate the required delay by either pushing out or pulling in the feedback clock to reduce the phase error between the reference clock and the feedback clock.

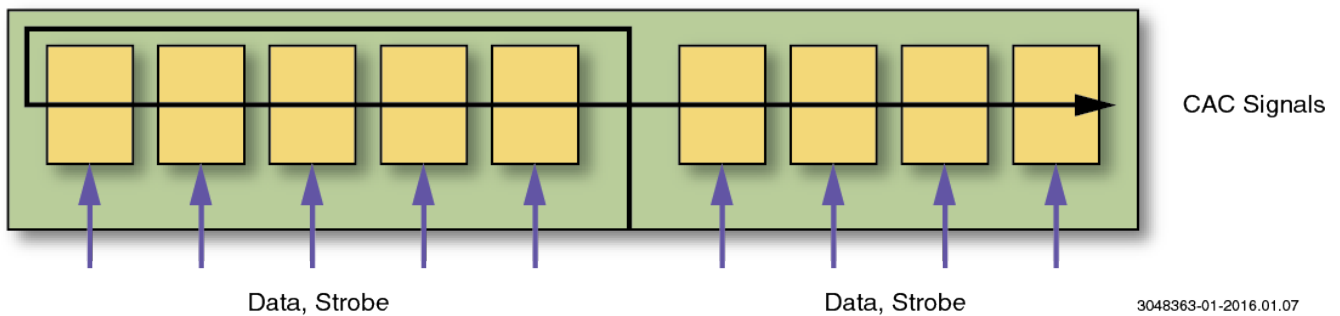
The SDLL and the phase interpolator are used to adjust the delay of the strobe signal and data signals in the data module so that they are aligned. The phase interpolator gets 17 clocks with a phase separation of 22.5° from the SDLL, and then performs fine tuning by mixing various phases as determined by the programmable configuration bit settings. This phase interpolator has two stages of clock mixing.

Soft Write/Read Leveling

Write and read leveling are needed in DDR3 interfaces to align the DQS with the data. Both signals are routed as point-to-point connections from the controller to the DIMM, along with the control-address-command (CAC) signals, which are routed in a fly-by network topology on the DIMM for performance and signal integrity reasons. With the total delay across component memories on a ×72 dual-rank DIMM being in excess of 1.5 ns, it is important for the controller to have the full 1.5T delay range at its disposal to provide for appropriate leveling delays to all component memories. At 1600 Mbps, with an 800 MHz clock (1.25 ns), a 1.5T delay range provides for 1.875 ns of total delay.

The figure below is a high-level illustration showing the delay through the component memories for the CAC line in a fly-by topology. In dual-rank DIMMs with component memories on both sides of the DIMM, the delay through the components is much larger.

Figure 24: High-Level View of CAC Signals on a DIMM



The following figure highlights the difference between using a hard and soft write-leveling controller.

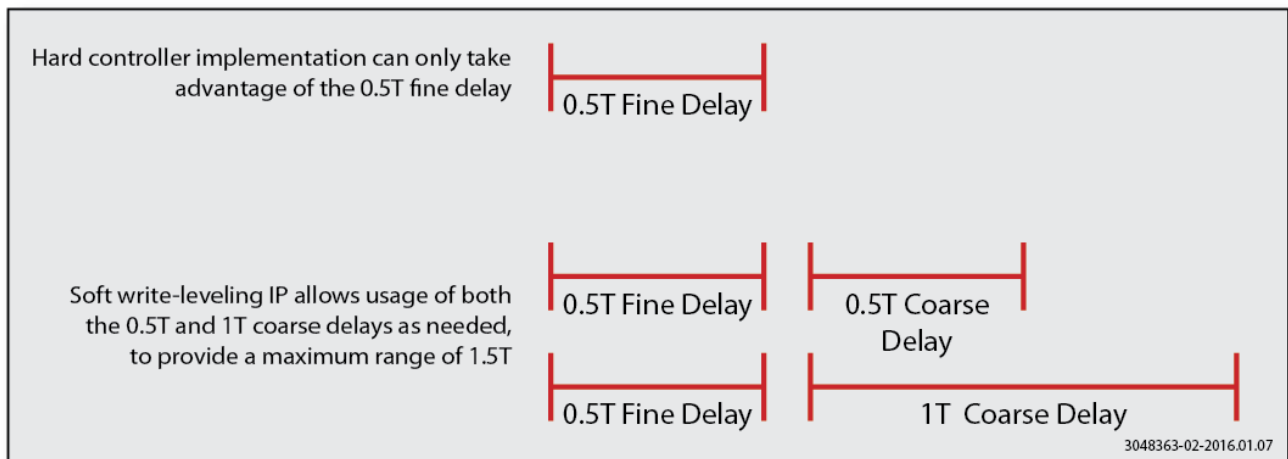



Figure 25: Hard Versus Soft Controller Comparison

Starting with ACE 6.0, all DDR3 interfaces are required to use a soft-logic implementation in the FPGA fabric to implement the write and read-leveling algorithms. This soft-logic implementation provides users the ability to take

advantage of the full 1.5T delay range for write leveling plus additional features to ensure a more robust and production quality implementation across data rates of up to 1600 Mbps.

Note

-  The soft-logic implementation of the write and read-leveling logic requires HD1000 silicon rev A3 or later. Contact Achronix support if using an earlier revision of the HD1000 silicon.

The soft write/read-leveling logic in the fabric consumes about 2,500 registers, 2,300 LUTs and 15 LRAM memories. Even with all six DDR3 controllers in use, the fabric resources consumed would amount to 15,000 registers (2.14%), 15,000 LUTs (2.14%) and 90 LRAMs (1.46%). With a total resource usage of about 2%, the soft write/read-leveling logic consumes a very small fraction of the total available resources, leaving the rest of the fabric for the customer design and implementation.

Read-Leveling Values

Read-leveling tap values are represented by a 9-bit value. Every setting is supported, supplying a continuous array of values from 0 to 511 (although the upper end of the range is rarely used). The formula for the amount of delay is:

Full-rate clock time period at DDR3 data rate / 64 = tap delay

For example, at 1600 Mbps with a clock rate of 800 MHz or a period of 1.25 ns, the delay per tap is:

$1.25 \text{ ns} / 64 = 19.5 \text{ ps}$

Write-Leveling Values

Write-leveling tap values are represented by per byte-lane 8-bit values. Each 8-bit value is composed of two parts:

- The lower 6 bits indicate the DLL tap value, with legal values range between 6'd17 and 6'd45 (or 6'h11 and 6'h2D), representing 29 possible tap settings.
- The upper 2 bits indicate a coarse delay to be added to the lower 6-bit delay value if required. Possible settings are:

00 – 0T delay (no additional delay)


01 – 0.5T delay (one-half clock delay is added)

10 – 1T delay (one full clock cycle delay is added)

11 – Reserved (not legal)

During write leveling, the controller sweeps through each of the 6-bit fine-delay values. If a suitable setting is not found, the controller increments the 2-bit coarse delay value, and then sweeps through the fine-delay values again to find a match. If no match is found, the process is repeated.

Note

-  The JEDEC specification allows for a tolerance of $\pm 0.25T$ (± 8 taps) for the delay value.

Chapter - 8: Memory Interface Latency

Depending on the data rate and the mode (width) used, the write and read latency for the DDR3 controller will vary. Table 4 below provides detailed clock cycle counts for write and read latencies for each of these cases.

Table 19: Latency Information for Different Memory Controller Modes

Mode	Data-Rate	Write-Latency (Clock Cycles)	Read-Latency (Clock Cycles)
2X	Up to 1600 Mbps	6	9
Wide-Bus	Up to 1600 Mbps	8	11

Chapter - 9: SPD Parameters and Configuration

In order for the DDR3 core to properly interface with off-chip memory modules, the serial presence detect (SPD) parameters must be programmed into the core. Currently, the only supported method for determining the SPD parameters is manually calculating them via an Achronix-supplied spreadsheet.

Manual Setting with Parameter Calculator

Generating the need DDR3 macro configuration is a straightforward process.

Step 1. Collect Data

After selecting the desired memory module, determine the exact memory devices used in the module and locate the needed timing parameters for the three supported data rates: 1066 Mbps, 1333 Mbps and 1600 Mbps.

Step 2. Enter Data into the Spreadsheet

Transfer the needed timing parameters to the `DDR3_Parameter_Worksheet.xlsx` spreadsheet. The spreadsheet performs the translation from nanosecond-based specifications to data-rate-specific clock requirement, expressed as a hex value as needed for the DDR3 macro.

Although there may be min-max range in the memory vendor's datasheet, typically, only the min value is needed. There are two exceptions: the CAS write latency (t_{CWL}) and the maximum average periodic refresh (t_{REFI}), where the maximum value is needed.

Note



For those items in the spreadsheet where no timing parameter is shown, the hard-coded values should be used. These timings are specific to the macro and should not be altered.

Table 20: Sample DDR3_Parameter_Worksheet.xlsx Spreadsheet Showing the Timing Parameters for Micron Part Number MT18KSF1G72HZ

Speed (Mbps)		1066				1333				1600			
Clk Period (ns)		1.88				1.50				1.25			
Description	Timing Parameter	min (ns)	max (ns)	min (CK)	max (CK)	min (ns)	max (ns)	min (CK)	max (CK)	min (ns)	max (ns)	min (CK)	max (CK)
ACTIVATE-to-PRECHARGE command period	tRAS	37.5	702.00	'h14	'h9229	36	702.00	'h18	'hB6C5	35	702.00	'h1C	'hDB60
ACTIVATE to internal READ or WRITE delay time	tRCD	13.125	-	'h7	-	13.5	-	'h9	-	13.75	-	'hB	-
PRECHARGE command period	tRP	13.125	-	'h7	-	13.5	-	'h9	-	13.75	-	'hB	-

Speed (Mbps)		1066				1333				1600			
ACTIVATE-to-ACTIVATE or REFRESH command period	tRC	50.625	-	'h1B	-	49.5	-	'h21	-	48.75	-	'h27	-
ACTIVATE-to-ACTIVATE minimum command period	tRRD	7.5	-	'h4	-	6	-	'h4	-	6	-	'h5	-
Four ACTIVATE windows	tFAW	37.5	-	'h14	-	30	-	'h14	-	30	-	'h18	-
REFRESH-to-ACTIVATE or REFRESH command period	tRFC	260	70200	'h8B	'h9229	260	70200	'hAE	'hB6C5	260	70200	'hD0	'hDB60
READ-to-PRECHARGE time	tRTP	7.5	-	'h4	-	7.5	-	'h5	-	7.5	-	'h6	-
Write recovery time	tWR	15	-	'h8	-	15	-	'hA	-	15	-	'hC	-
Delay from start of internal WRITE transaction to internal READ command	tWTR	7.5	-	'h4	-	7.5	-	'h5	-	7.5	-	'h6	-
Delay from start of internal READ transaction to internal WRITE command	tRTW	-	-	'h5	-	-	-	'h5	-	-	-	'h5	-
Delay from start of WRITE transaction to WRITE command when accessing different banks		-	-	'h3	-	-	-	'h3	-	-	-	'h4	-
Delay from start of READ transaction to READ command when accessing different banks		-	-	'h3	-	-	-	'h3	-	-	-	'h4	-
Delay from start of WRITE transaction to READ command when accessing different banks		-	-	'h1	-	-	-	'h1	-	-	-	'h1	-
Additive DDR3 Latency		-	-	'h0	-	-	-	'h0	-	-	-	'h0	-
CAS Read Latency	tCL	13.125	-	'h7	-	13.125	-	'h9	-	13.75	-	'hB	-
CAS Write Latency	tCWL	-	-	'h5	'h6	-	-	'h5	'h7	-	-	'h5	'h8
Delay from read enable to valid, per byte lane		-	-	'h3	-	-	-	'h3	-	-	-	'h4	-
MODE REGISTER SET command cycle time	tMRD	-	-	'h4	-	-	-	'h4	-	-	-	'h4	-
MODE REGISTER SET command update delay	tMOD	-	-	'hC	-	-	-	'hC	-	-	-	'hC	-
Exit self refresh to commands not requiring a locked DLL	tXS	270	-	'h90	-	270	-	'hB4	-	270	-	'hD8	-
Exit self refresh to commands requiring a locked DLL		512	-	'hF	'hF	512	-	'hF	'hF	512	-	'hF	'hF
Exit reset from CKE HIGH to a valid command	tXPR	270	-	'h90	-	270	-	'hB4	-	270	-	'hD8	-
Maximum average periodic refresh	tREFI	3900	7800	'h81E	'h103D	3900	7800	'hA27	'h144E	3900	7800	'hC30	'h1860

Step 3. Transfer the Calculated Values to the Macro

The values marked in green in the `DDR3_Parameter_Worksheet.xlsx` spreadsheet should be transferred to the Memory Timing Page for the DDR3 macro within ACE and the macro code regenerated. Alternately, enter the timing values in macro wrapper file.

Step 4. Regenerate the Bitstream for the Revised Design

In order to write the updated timing parameters to the devices configuration register, an update bitstream is needed. Run the design the entire design flow: synthesis to ACE.

Automated I2C Read and Update

This option is currently not available. For more information, please contact Achronix support.

Chapter - 10: ACE Macro Generation and Integration


Note



The following content has been excerpted from the ACE User Guide (UG001 and UG070).

Overview

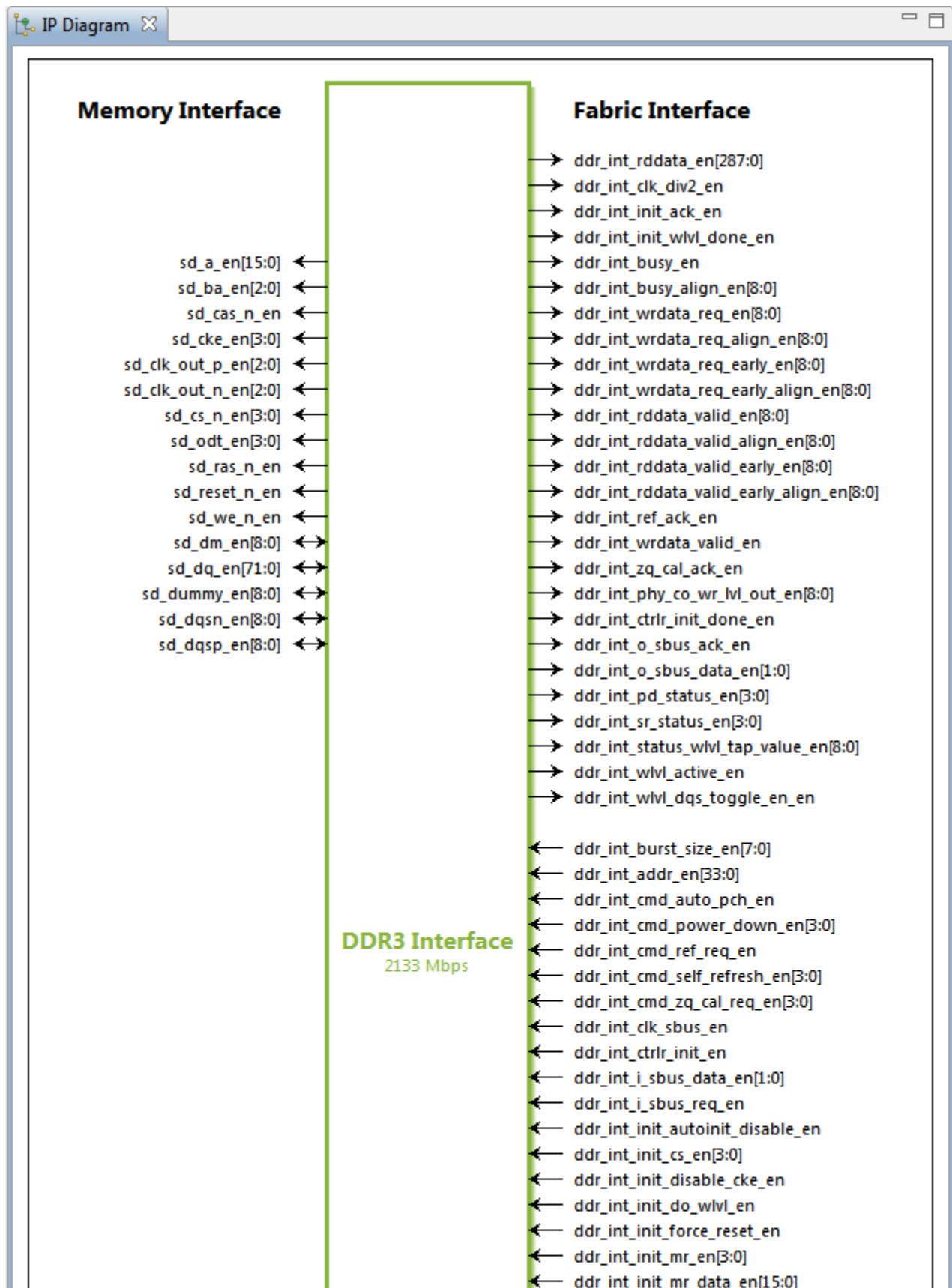
The DDR3 Configuration Editor provides a graphical wizard for creating a DDR3 Interface IP configuration file (.acxip). This editor allows the user to generate the required configuration files for designs requiring the embedded DDR3 controllers. See [Creating an IP Configuration](#).

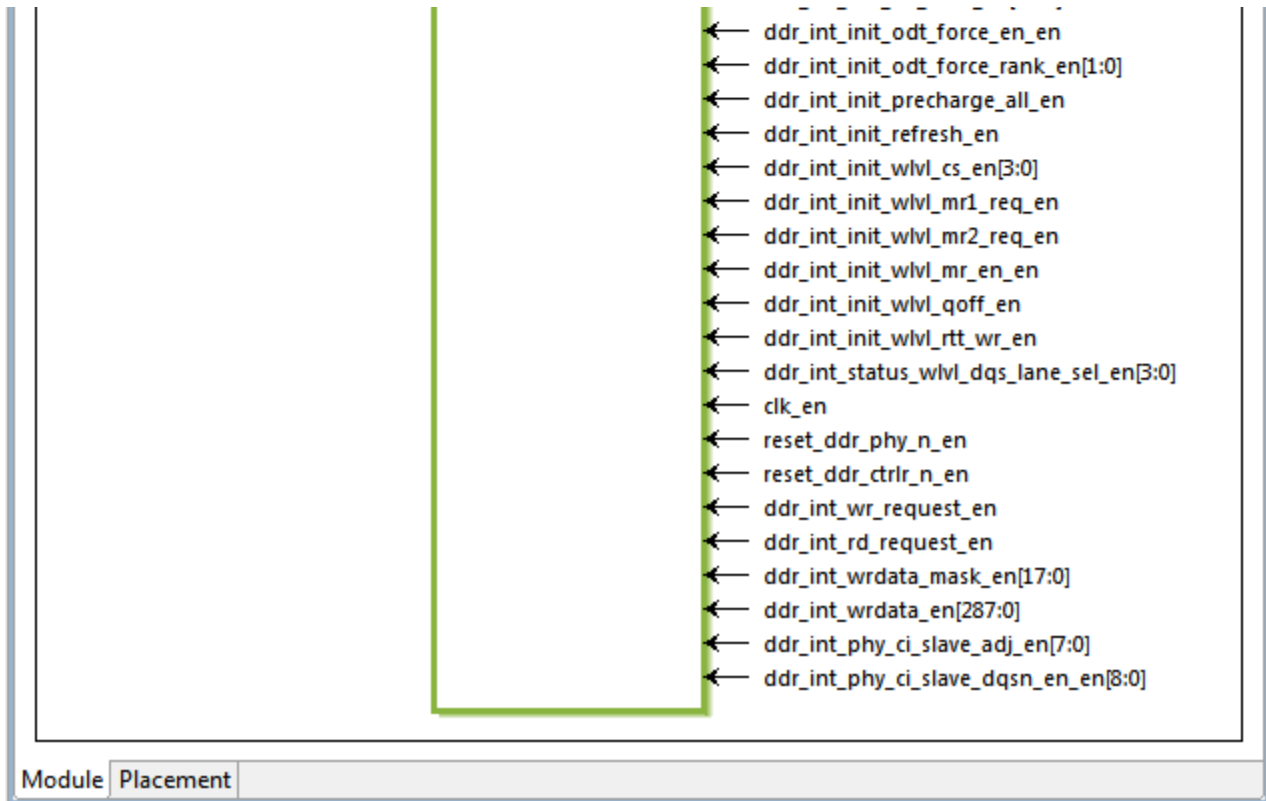
By default, the DDR3 Configuration Editor is included in the  IP Configuration perspective (**Window -> Open Perspective -> IP Configuration**). The DDR3 Interface configuration information is broken up into several pages, organized by concept.

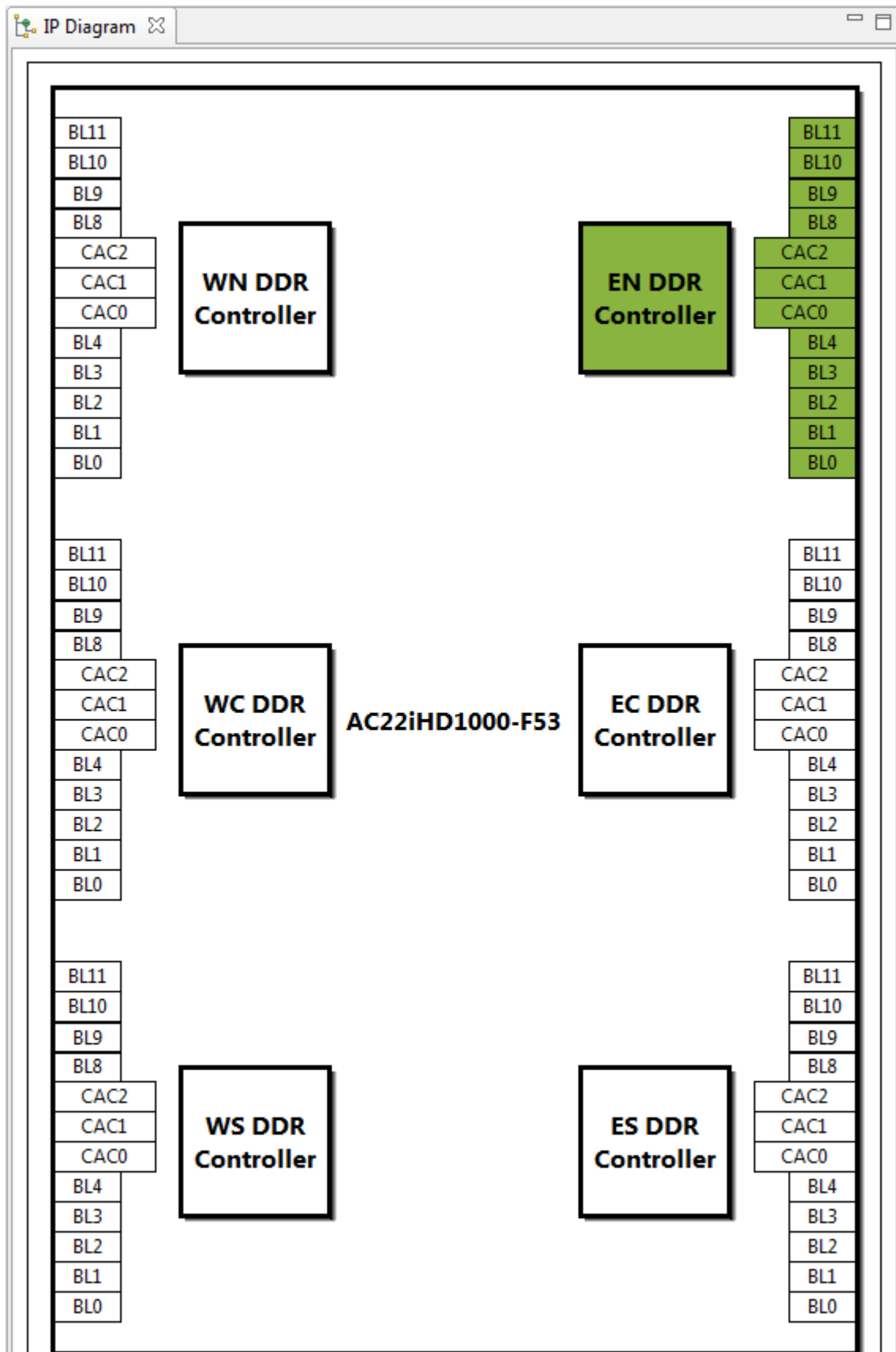
Once the user has configured the DDR Interface to meet their requirements, and the DDR3 Configuration Editor has determined that there are no errors in the configuration, the user may choose to generate their IP design files (see [Generating the IP Design Files](#)).

DDR3 IP Diagrams

Note that the DDR3 Configuration Editors support two subtabs in the IP Diagram View: a **Module** tab showing the detailed DDR3 Interface based upon the current configuration, and a **Placement** tab showing the placement of the DDR3 Controller within the chosen **Target Device**, as selected on the Overview Page. Both of these diagram tabs will update on-the-fly to match the current DDR3 configuration in the editor pages.









Sections

- [DDR3 Configuration Editor Overview Page \(see page 73\)](#)
- [DDR3 Configuration Editor Memory Timing Page \(see page 77\)](#)
- [DDR3 Configuration Editor DLL Timing Page \(see page 80\)](#)
- [DDR3 Configuration Editor DQS Preamble Page \(see page 82\)](#)
- [DDR3 Controller Placement Constraints \(see page 85\)](#)

DDR3 Configuration Editor Overview Page

The Overview page of the Speedster22i DDR3 Configuration Editor contains the top-level, global properties that govern the structure and base configuration of the DDR3 Interface.

Speedster22i DDR3

Overview

This page contains the top-level, global properties that govern the structure and base configuration of the DDR Interfa...

✓	Target Device	AC22iHD1000-F53
✓	Placement	WN
✓	<input checked="" type="checkbox"/> Add Placement Suffix to Port Names	
✓	Clock Pin Name	clk
✓	DIMM Type	Custom
✓	Data Rate (Mbps)	1600
✓	Fabric Interface Width	2x
✓	Data Width	64
✓	Number of CLKOUTs	2
✓	Number of Ranks	2
✓	FPGA IO Buffer Termination (Ohms)	50
✓	Termination at the DIMM	Dynamic 60 Ohms
✓	<input type="checkbox"/> Enable Address Mirroring	
✓	<input checked="" type="checkbox"/> Enable Wide Bus Interface	
✓	<input checked="" type="checkbox"/> Enable Data Bit Swizzle	
✓	<input checked="" type="checkbox"/> Enable Soft Read and Write Leveling	

DIMM Settings


✓	Number Of Column Bits	10
✓	Number Of Row Bits	15
✓	Number Of Bank Bits	3
✓	DQ Per DQS	8
✓	<input type="checkbox"/> Registered DIMM	
✓	<input type="checkbox"/> Enable Single Rank Command Issuing	

?
Generate
<< Back
Next >>

Configuration	File Preview	
---------------	--------------	--

Table 21: DDR3 Editor Overview Page Options

Option	Description
Target Device	Allows the user to select from the Achronix devices that support this IP.
Placement	Select the location on the chip where this DDR interface should be placed. (Alternately, left-click on the desired Controller in the Placement Diagram of the IP Diagram View.)
Add Placement Suffix to Port Names	Enabling this option causes the generated RTL wrapper and constraints to attach a suffix to all port names based on placement. This helps uniquify the top-level IOs in the design. For example, if you place your DDR interface on the East North site, "_en" would get added to the end of all the port names.
Clock Pin Name	Enter the reference clock pin name. Will be used to generate clock constraints. May be a top level design pin, a PLL clock output pin, etc.
DIMM Type	Select from a predefined library of standard DIMMs, or select Custom for full customization options.
Data Rate (Mbps)	May be configured to use a standard rate, or may be customized. The max data rate for 1x Fabric Interface Width is 1066Mbps. The max data rate for a Fabric Interface Width of 2x is 1600Mbps.
Fabric Interface Width	The fabric interface width can be set to 2x to enable a wider Core interface that runs at half speed. This is needed for higher data rates, such as 1600 Mbps.
Data Width	Local side data width.
Number of CLKOUTs	Number of DDR3 DIMM clocks.
Number of Ranks	Number of chip selects used.
FPGA IO Buffer Termination (Ohms)	The FPGA IO buffer termination is specified to ensure that the trace impedance is matched and that reflections are minimized during read operations.
Termination at the DIMM	Termination settings at the DIMM configured through MR registers. The options are to enable or disable dynamic ODT, and the termination value to be used when performing write operations.
Enable Address Mirroring	Enable Address Mirroring in the DDR Controller.
Enable Wide Bus Interface	When enabled, this doubles the width of the data bus, and helps meet timing at higher frequencies.

Option	Description
Enable Data Bit Swizzle	<p>Swizzle enables data bits from the fabric to the IO ring to be re-arranged to optimize for routing. This is especially important at high data rates, as it will help significantly with timing closure.</p> <div data-bbox="342 380 1463 443" style="border: 1px solid #add8e6; padding: 5px;">  Please note that when swizzle is enabled, data masking will not be available. </div>
Enable Soft Read and Write Leveling	<p>Enabling soft read and write leveling ensures that the algorithms to perform the read and write leveling functions are built in the programmable logic fabric and seamlessly integrated into the ddr3 macro. This option allows the PHY to take advantage of a larger delay range, and prevents the need for manual tuning of the PHY clk/cac delay values. This option prevents the need for manual tuning of the read delay parameters for every bit in the data byte lanes.</p>
DIMM Settings	
Number of Column Bits	Number of bits for Column Address.
Number of Row Bits	Number of bits for Row Address.
Number of Bank Bits	Number of bits for Bank Address.
DQ Per DQS	Number of DQ bits per DQS line.
Registered DIMM	Whether or not the DDR3 DIMM is registered.
Enable Single Rank Command Issuing	<p>When set, controller only issues commands to one rank at a time (i.e. only one chip select is ever asserted). This needs to be set for RDIMMs, and can optionally be set for UDIMMs for staggered refresh commands to reduce instantaneous current.</p>

DDR3 Configuration Editor Memory Timing Page

The Memory Timing Page allows the user to configure the memory timings for the DDR3 Interface.

Speedster22i DDR3

Memory Timing

This page allows the user to configure the Memory Timing

AL/CL/CWL

- ✓ Additive latency value (AL) in Clocks (decimal)
- ✓ CAS Latency (tCL) in Clocks (decimal)
- ✓ CAS Write Latency (tCWL) in Clocks (decimal)

Auto-Refresh

- ✓ Controller Refresh Enable
- ✓ Refresh Period (tREFI) in ns (decimal)

Command-to-Command Delays

- ✓ Activate To Activate Diff Bank (tRRD) in Clocks (decimal)
- ✓ Activate To Activate Same Bank (tRC) in ns (decimal)
- ✓ Activate To Precharge (tRAS) in ns (decimal)
- ✓ Activate To R/W (tRCD) in ns (decimal)
- ✓ Auto-refresh To Activate Same Bank (tRFC) in ns (decimal)
- ✓ Bank Activate Period (tFAW) in ns (decimal)
- ✓ Loadmode To Activate (tMRD) in Clocks (decimal)
- ✓ Loadmode To Any (tMOD) in Clocks (decimal)
- ✓ Precharge To Activate (tRP) in ns (decimal)
- ✓ Read To Precharge (tRTP) in Clocks (decimal)
- ✓ Reset High to Clock High (tXPR) in ns (decimal)
- ✓ Self-refresh to Non-DLL command (tXS) in ns (decimal)
- ✓ Self-refresh to Non-Read command in Clocks (decimal)
- ✓ Write To Precharge (tWR) in ns (decimal)
- ✓ Write To Read (tWTR) in Clocks (decimal)
- ✓ Read To Read Diff Bank in Clocks (decimal)
- ✓ Read to Write in Clocks (decimal)
- ✓ Write To Read Diff Bank in Clocks (decimal)

Write To Write Diff Bank in Clocks (decimal)

Write To Write Diff Bank in Clocks (decimal)

Configuration File Preview

Table 22: DDR3 Editor Memory Timing Page Options

Option	Description
AL/CL/CWL	
Additive latency value (AL) in Clocks (decimal)	AL in clock cycles
CAS Latency (tCL) in Clocks (decimal)	CAS Latency (tCL) in clock cycles
CAS Write Latency (tCWL) in Clocks (decimal)	CAS Write Latency (tCWL) in clock cycles
Auto-Refresh	
Controller Refresh Enable	When enabled, the DDR Controller will handle DDR Memory Refreshes
Refresh Period (tREFI) in ns (decimal)	Duration between Refresh commands (tREFI) in nanoseconds
Command-to-Command Delays	
Activate To Activate Diff Bank (tRRD) in Clocks (decimal)	Minimum clock cycles between ACTIVATE to ACTIVATE in different banks (tRRD)
Activate To Activate Same Bank (tRC) in ns (decimal)	Minimum duration from Activate to Activate/Auto-refresh in same bank (tRC) in nanoseconds
Activate To Precharge (tRAS) in ns (decimal)	Minimum duration from Activate to Precharge (tRAS) in nanoseconds
Activate To R/W (tRCD) in ns (decimal)	Minimum duration between Activate and Read/Write (tRCD) in nanoseconds
Auto-refresh To Activate Same Bank (tRFC) in ns (decimal)	Minimum duration in nanoseconds from Auto Refresh to Activate /Auto Refresh in the same Bank
Bank Activate Period (tFAW) in ns (decimal)	Four bank activate period (tFAW) in nanoseconds
Loadmode To Activate (tMRD) in Clocks (decimal)	Minimum clock cycles from Loadmode to Activate command (tMRD)

Option	Description
Loadmode To Any (tMOD) in Clocks (decimal)	Minimum clock cycles from Loadmode to Any command (tMOD)
Precharge To Activate (tRP) in ns (decimal)	Minimum duration between Precharge to Activate in nanoseconds
Read To Precharge (tRTP) in Clocks (decimal)	Minimum clock cycles from Read to Precharge (tRTP)
Reset High to Clock High (tXPR) in ns (decimal)	Minimum duration from memory reset high to cke high (tXPR) in nanoseconds
Self-refresh to Non-DLL command (tXS) in ns (decimal)	Minimum duration from Self-refresh to Non-DLL command in nanoseconds
Self-refresh to Non-Read command in Clocks (decimal)	Minimum clock cycles from Self-refresh to Non-Read command
Write To Precharge (tWR) in ns (decimal)	Minimum duration from write to Precharge (tWR) in nanoseconds
Write To Read (tWTR) in Clocks (decimal)	Minimum clock cycles from Write to Read (tWTR)
Read To Read Diff Bank in Clocks (decimal)	Minimum clock cycles from Read to Read (different banks)
Read to Write in Clocks (decimal)	Read to Write delay in clock cycles
Write To Read Diff Bank in Clocks (decimal)	Minimum clock cycles from Write to Read (different banks)
Write To Write Diff Bank in Clocks (decimal)	Minimum clock cycles from Write to Write (different banks)

DDR3 Configuration Editor DLL Timing Page

The DLL Timing Page of the Speedster22i DDR3 Configuration Editor allows the user to configure the DLL timing parameters for the DDR3 Interface.

The number of data entry fields on this page will vary according to the number of Byte Lanes used by the memory interface. Each used byte lane will have a DQ Slave Adjust value for each of the 8 bits, along with a single DQS Slave Adjust value.

For simplicity, the following screenshot will show a single Byte Lane configuration, which is only valid for a (custom) data width of 8.

*new_ddr3_1.acxip

Speedster22i DDR3

DLL Timing

This page allows the user to configure the DLL Timing

✓ Board Material FAST (eg. Megtron6, TSM)

Byte Lane 0

✓ DQ Slave Adjust Bit 0 (hex) 0f

✓ DQ Slave Adjust Bit 1 (hex) 0c

✓ DQ Slave Adjust Bit 2 (hex) 12

✓ DQ Slave Adjust Bit 3 (hex) 12

✓ DQ Slave Adjust Bit 4 (hex) 10

✓ DQ Slave Adjust Bit 5 (hex) 11

✓ DQ Slave Adjust Bit 6 (hex) 13

✓ DQ Slave Adjust Bit 7 (hex) 13

✓ DQS Slave Adjust (hex) 1f

✓ Use Dynamic PHY Delay Settings

Static PHY Delay Settings

✓ DP Slave adjust for CAC0 Byte Lane Clock (hex) 38

✓ DP Slave adjust for CAC0 Byte Lane CAC (hex) 0c

✓ DP Slave adjust for CAC1 Byte Lane Clock (hex) 38

✓ DP Slave adjust for CAC1 Byte Lane CAC (hex) 0c

✓ DP Slave adjust for CAC2 Byte Lane Clock (hex) 38

✓ DP Slave adjust for CAC2 Byte Lane CAC (hex) 0c

✓ DP Slave adjust for CAC3 Byte Lane Clock (hex) 38

✓ DP Slave adjust for CAC3 Byte Lane CAC (hex) 0c

? Generate << Back Next >>

Configuration File Preview

Table 23: DDR3 Editor DLL Timing Page Options

Option	Description
Board Material	Board material and associated properties are used for delay calculations.
Byte Lane N	
DQ Slave Adjust Bit 0 (hex)	DLL delay adjust value for DQ bit 0
...	...
DQ Slave Adjust Bit 7 (hex)	DLL delay adjust value for DQ bit 7
DQS Slave Adjust (hex)	DLL delay adjust value for all DQS lines in byte_laneN
Use Dynamic PHY Delay Settings	Static PHY delay setting enables a bitstream specific set of Clk and Cac delay values to be provided to the controller as specified below. These are used to ensure that write-leveling values for each bytelane fall within a desired range. Dynamic PHY delay setting provides user the ability to change these in user mode by feeding the appropriate values through a fabric interface.
Static PHY Delay Settings	
DP Slave adjust for CAC0 Byte Lane Clock (hex)	DP Slave adjust for CAC Byte Lane Clock
DP Slave adjust for CAC0 Byte Lane CAC (hex)	DP Slave adjust for CAC Byte Lane CAC
...	...
DP Slave adjust for CAC3 Byte Lane Clock (hex)	DP Slave adjust for CAC Byte Lane Clock
DP Slave adjust for CAC3 Byte Lane CAC (hex)	DP Slave adjust for CAC Byte Lane CAC

DDR3 Configuration Editor DQS Preamble Page

The DQS Preamble page of the Speedster22i DDR3 Configuration Editor allows the user to configure the DQS Preamble settings of the interface.

Speedster22i DDR3
DQS Preamble
 This page allows the user to configure the DQS Preamble

DQS Preamble Enable On Lane 0 (hex)
 DQS Preamble Enable Off Lane 0 (hex)
 DQS Preamble Enable On Lane 1 (hex)
 DQS Preamble Enable Off Lane 1 (hex)
 DQS Preamble Enable On Lane 2 (hex)
 DQS Preamble Enable Off Lane 2 (hex)
 DQS Preamble Enable On Lane 3 (hex)
 DQS Preamble Enable Off Lane 3 (hex)
 DQS Preamble Enable On Lane 8 (hex)
 DQS Preamble Enable Off Lane 8 (hex)
 DQS Preamble Enable On Lane 9 (hex)
 DQS Preamble Enable Off Lane 9 (hex)
 DQS Preamble Enable On Lane 10 (hex)
 DQS Preamble Enable Off Lane 10 (hex)
 DQS Preamble Enable On Lane 11 (hex)
 DQS Preamble Enable Off Lane 11 (hex)
 DQS Preamble Enable On Lane 12 (hex)
 DQS Preamble Enable Off Lane 12 (hex)

Option	Description
DQS Preamble Enable On Lane <i>N</i> (hex)	DQS Preamble Enable On Lane <i>N</i>
DQS Preamble Enable Off Lane <i>N</i> (hex)	DQS Preamble Enable Off Lane <i>N</i>

DDR3 Controller Placement Constraints

The sample code below contains ACE pre-placement directives of one DDR controller instance as well as the I/O placements onto the 22i device for NW quadrant. The same placement information can be used for all other quadrants (WN, WC, WN, ES, EC, and EN) by replacing WS with respective letters. When generating a DDR Macro using the ACE GUI, these constraints are generated automatically.

```

set_placement -fixed -batch {p:clk_p} {d:pad0_clk_bank_ws}
set_placement -fixed -batch {p:clk_n} {d:pad1_clk_bank_ws}
set_placement -fixed -batch {p:reset_n} {d:pad2_clk_bank_ws}
set_placement -fixed -batch {p:sd_dq[0]} {d:pad_ws_byteio0_dq0}
set_placement -fixed -batch {p:sd_dq[1]} {d:pad_ws_byteio0_dq1}
set_placement -fixed -batch {p:sd_dq[2]} {d:pad_ws_byteio0_dq2}
set_placement -fixed -batch {p:sd_dq[3]} {d:pad_ws_byteio0_dq3}
set_placement -fixed -batch {p:sd_dq[4]} {d:pad_ws_byteio0_dq4}
set_placement -fixed -batch {p:sd_dq[5]} {d:pad_ws_byteio0_dq5}
set_placement -fixed -batch {p:sd_dq[6]} {d:pad_ws_byteio0_dq6}
set_placement -fixed -batch {p:sd_dq[7]} {d:pad_ws_byteio0_dq7}
set_placement -fixed -batch {p:sd_dqsp[0]} {d:pad_ws_byteio0_dqs}
set_placement -fixed -batch {p:sd_dqsn[0]} {d:pad_ws_byteio0_dqsn}
set_placement -fixed -batch {p:sd_dm[0]} {d:pad_ws_byteio0_dq8}
set_placement -fixed -batch {p:sd_dummy[0]} {d:pad_ws_byteio0_dq9}
set_placement -fixed -batch {p:sd_dq[8]} {d:pad_ws_byteio1_dq0}
set_placement -fixed -batch {p:sd_dq[9]} {d:pad_ws_byteio1_dq1}
set_placement -fixed -batch {p:sd_dq[10]} {d:pad_ws_byteio1_dq2}
set_placement -fixed -batch {p:sd_dq[11]} {d:pad_ws_byteio1_dq3}
set_placement -fixed -batch {p:sd_dq[12]} {d:pad_ws_byteio1_dq4}
set_placement -fixed -batch {p:sd_dq[13]} {d:pad_ws_byteio1_dq5}
set_placement -fixed -batch {p:sd_dq[14]} {d:pad_ws_byteio1_dq6}
set_placement -fixed -batch {p:sd_dq[15]} {d:pad_ws_byteio1_dq7}
set_placement -fixed -batch {p:sd_dqsp[1]} {d:pad_ws_byteio1_dqs}
set_placement -fixed -batch {p:sd_dqsn[1]} {d:pad_ws_byteio1_dqsn}
set_placement -fixed -batch {p:sd_dm[1]} {d:pad_ws_byteio1_dq8}
set_placement -fixed -batch {p:sd_dummy[1]} {d:pad_ws_byteio1_dq9}
set_placement -fixed -batch {p:sd_dq[16]} {d:pad_ws_byteio2_dq0}
set_placement -fixed -batch {p:sd_dq[17]} {d:pad_ws_byteio2_dq1}
set_placement -fixed -batch {p:sd_dq[18]} {d:pad_ws_byteio2_dq2}
set_placement -fixed -batch {p:sd_dq[19]} {d:pad_ws_byteio2_dq3}
set_placement -fixed -batch {p:sd_dq[20]} {d:pad_ws_byteio2_dq4}
set_placement -fixed -batch {p:sd_dq[21]} {d:pad_ws_byteio2_dq5}
set_placement -fixed -batch {p:sd_dq[22]} {d:pad_ws_byteio2_dq6}
set_placement -fixed -batch {p:sd_dq[23]} {d:pad_ws_byteio2_dq7}
set_placement -fixed -batch {p:sd_dqsp[2]} {d:pad_ws_byteio2_dqs}
set_placement -fixed -batch {p:sd_dqsn[2]} {d:pad_ws_byteio2_dqsn}
set_placement -fixed -batch {p:sd_dm[2]} {d:pad_ws_byteio2_dq8}
set_placement -fixed -batch {p:sd_dummy[2]} {d:pad_ws_byteio2_dq9}
set_placement -fixed -batch {p:sd_dq[24]} {d:pad_ws_byteio3_dq0}
set_placement -fixed -batch {p:sd_dq[25]} {d:pad_ws_byteio3_dq1}
set_placement -fixed -batch {p:sd_dq[26]} {d:pad_ws_byteio3_dq2}
set_placement -fixed -batch {p:sd_dq[27]} {d:pad_ws_byteio3_dq3}
set_placement -fixed -batch {p:sd_dq[28]} {d:pad_ws_byteio3_dq4}
set_placement -fixed -batch {p:sd_dq[29]} {d:pad_ws_byteio3_dq5}
set_placement -fixed -batch {p:sd_dq[30]} {d:pad_ws_byteio3_dq6}
set_placement -fixed -batch {p:sd_dq[31]} {d:pad_ws_byteio3_dq7}
set_placement -fixed -batch {p:sd_dqsp[3]} {d:pad_ws_byteio3_dqs}
set_placement -fixed -batch {p:sd_dqsn[3]} {d:pad_ws_byteio3_dqsn}

```

```

set_placement -fixed -batch {p:sd_dm[3]} {d:pad_ws_byteio3_dq8}
set_placement -fixed -batch {p:sd_dummy[3]} {d:pad_ws_byteio3_dq9}

set_placement -fixed -batch {p:sd_dq[32]} {d:pad_ws_byteio8_dq0}
set_placement -fixed -batch {p:sd_dq[33]} {d:pad_ws_byteio8_dq1}
set_placement -fixed -batch {p:sd_dq[34]} {d:pad_ws_byteio8_dq2}
set_placement -fixed -batch {p:sd_dq[35]} {d:pad_ws_byteio8_dq3}
set_placement -fixed -batch {p:sd_dq[36]} {d:pad_ws_byteio8_dq4}
set_placement -fixed -batch {p:sd_dq[37]} {d:pad_ws_byteio8_dq5}
set_placement -fixed -batch {p:sd_dq[38]} {d:pad_ws_byteio8_dq6}
set_placement -fixed -batch {p:sd_dq[39]} {d:pad_ws_byteio8_dq7}
set_placement -fixed -batch {p:sd_dqsp[4]} {d:pad_ws_byteio8_dqs}
set_placement -fixed -batch {p:sd_dqsn[4]} {d:pad_ws_byteio8_dqsn}
set_placement -fixed -batch {p:sd_dm[4]} {d:pad_ws_byteio8_dq8}
set_placement -fixed -batch {p:sd_dummy[4]} {d:pad_ws_byteio8_dq9}
set_placement -fixed -batch {p:sd_dq[40]} {d:pad_ws_byteio9_dq0}
set_placement -fixed -batch {p:sd_dq[41]} {d:pad_ws_byteio9_dq1}
set_placement -fixed -batch {p:sd_dq[42]} {d:pad_ws_byteio9_dq2}
set_placement -fixed -batch {p:sd_dq[43]} {d:pad_ws_byteio9_dq3}
set_placement -fixed -batch {p:sd_dq[44]} {d:pad_ws_byteio9_dq4}
set_placement -fixed -batch {p:sd_dq[45]} {d:pad_ws_byteio9_dq5}
set_placement -fixed -batch {p:sd_dq[46]} {d:pad_ws_byteio9_dq6}
set_placement -fixed -batch {p:sd_dq[47]} {d:pad_ws_byteio9_dq7}
set_placement -fixed -batch {p:sd_dqsp[5]} {d:pad_ws_byteio9_dqs}
set_placement -fixed -batch {p:sd_dqsn[5]} {d:pad_ws_byteio9_dqsn}
set_placement -fixed -batch {p:sd_dm[5]} {d:pad_ws_byteio9_dq8}
set_placement -fixed -batch {p:sd_dummy[5]} {d:pad_ws_byteio9_dq9}
set_placement -fixed -batch {p:sd_dq[48]} {d:pad_ws_byteio10_dq0}
set_placement -fixed -batch {p:sd_dq[49]} {d:pad_ws_byteio10_dq1}
set_placement -fixed -batch {p:sd_dq[50]} {d:pad_ws_byteio10_dq2}
set_placement -fixed -batch {p:sd_dq[51]} {d:pad_ws_byteio10_dq3}
set_placement -fixed -batch {p:sd_dq[52]} {d:pad_ws_byteio10_dq4}
set_placement -fixed -batch {p:sd_dq[53]} {d:pad_ws_byteio10_dq5}
set_placement -fixed -batch {p:sd_dq[54]} {d:pad_ws_byteio10_dq6}
set_placement -fixed -batch {p:sd_dq[55]} {d:pad_ws_byteio10_dq7}
set_placement -fixed -batch {p:sd_dqsp[6]} {d:pad_ws_byteio10_dqs}
set_placement -fixed -batch {p:sd_dqsn[6]} {d:pad_ws_byteio10_dqsn}
set_placement -fixed -batch {p:sd_dm[6]} {d:pad_ws_byteio10_dq8}
set_placement -fixed -batch {p:sd_dummy[6]} {d:pad_ws_byteio10_dq9}
set_placement -fixed -batch {p:sd_dq[56]} {d:pad_ws_byteio11_dq0}
set_placement -fixed -batch {p:sd_dq[57]} {d:pad_ws_byteio11_dq1}
set_placement -fixed -batch {p:sd_dq[58]} {d:pad_ws_byteio11_dq2}
set_placement -fixed -batch {p:sd_dq[59]} {d:pad_ws_byteio11_dq3}
set_placement -fixed -batch {p:sd_dq[60]} {d:pad_ws_byteio11_dq4}
set_placement -fixed -batch {p:sd_dq[61]} {d:pad_ws_byteio11_dq5}
set_placement -fixed -batch {p:sd_dq[62]} {d:pad_ws_byteio11_dq6}
set_placement -fixed -batch {p:sd_dq[63]} {d:pad_ws_byteio11_dq7}
set_placement -fixed -batch {p:sd_dqsp[7]} {d:pad_ws_byteio11_dqs}
set_placement -fixed -batch {p:sd_dqsn[7]} {d:pad_ws_byteio11_dqsn}
set_placement -fixed -batch {p:sd_dm[7]} {d:pad_ws_byteio11_dq8}
set_placement -fixed -batch {p:sd_dummy[7]} {d:pad_ws_byteio11_dq9}
set_placement -fixed -batch {p:sd_dq[64]} {d:pad_ws_byteio12_dq0}
set_placement -fixed -batch {p:sd_dq[65]} {d:pad_ws_byteio12_dq1}
set_placement -fixed -batch {p:sd_dq[66]} {d:pad_ws_byteio12_dq2}
set_placement -fixed -batch {p:sd_dq[67]} {d:pad_ws_byteio12_dq3}
set_placement -fixed -batch {p:sd_dq[68]} {d:pad_ws_byteio12_dq4}
set_placement -fixed -batch {p:sd_dq[69]} {d:pad_ws_byteio12_dq5}
set_placement -fixed -batch {p:sd_dq[70]} {d:pad_ws_byteio12_dq6}

```

```

set_placement -fixed -batch {p:sd_dq[71]} {d:pad_ws_byteio12_dq7}
set_placement -fixed -batch {p:sd_dqsp[8]} {d:pad_ws_byteio12_dqs}
set_placement -fixed -batch {p:sd_dqsn[8]} {d:pad_ws_byteio12_dqsn}
set_placement -fixed -batch {p:sd_dm[8]} {d:pad_ws_byteio12_dq8}
set_placement -fixed -batch {p:sd_dummy[8]} {d:pad_ws_byteio12_dq9}
set_placement -fixed -batch {p:sd_cs_n[2]} {d:pad_ws_byteio4_dq0}
set_placement -fixed -batch {p:sd_cs_n[3]} {d:pad_ws_byteio4_dq1}
set_placement -fixed -batch {p:sd_cke[2]} {d:pad_ws_byteio4_dq2}
set_placement -fixed -batch {p:sd_cke[3]} {d:pad_ws_byteio4_dq3}
set_placement -fixed -batch {p:sd_odt[2]} {d:pad_ws_byteio4_dq4}
set_placement -fixed -batch {p:sd_odt[3]} {d:pad_ws_byteio4_dq5}
set_placement -fixed -batch {p:sd_clk_p[0]} {d:pad_ws_byteio4_dqs}
set_placement -fixed -batch {p:sd_clk_n[0]} {d:pad_ws_byteio4_dqsn}
set_placement -fixed -batch {p:sd_cs_n[0]} {d:pad_ws_byteio5_dq0}
set_placement -fixed -batch {p:sd_cs_n[1]} {d:pad_ws_byteio5_dq1}
set_placement -fixed -batch {p:sd_cke[0]} {d:pad_ws_byteio5_dq2}
set_placement -fixed -batch {p:sd_cke[1]} {d:pad_ws_byteio5_dq3}
set_placement -fixed -batch {p:sd_odt[0]} {d:pad_ws_byteio5_dq4}
set_placement -fixed -batch {p:sd_odt[1]} {d:pad_ws_byteio5_dq5}
set_placement -fixed -batch {p:sd_reset_n} {d:pad_ws_byteio5_dq6}
set_placement -fixed -batch {p:sd_a[14]} {d:pad_ws_byteio5_dq8}
set_placement -fixed -batch {p:sd_a[15]} {d:pad_ws_byteio5_dq9}
set_placement -fixed -batch {p:sd_clk_p[1]} {d:pad_ws_byteio5_dqs}
set_placement -fixed -batch {p:sd_clk_n[1]} {d:pad_ws_byteio5_dqsn}
set_placement -fixed -batch {p:sd_a[0]} {d:pad_ws_byteio6_dq0}
set_placement -fixed -batch {p:sd_a[1]} {d:pad_ws_byteio6_dq1}
set_placement -fixed -batch {p:sd_a[2]} {d:pad_ws_byteio6_dq2}
set_placement -fixed -batch {p:sd_a[3]} {d:pad_ws_byteio6_dq3}
set_placement -fixed -batch {p:sd_a[4]} {d:pad_ws_byteio6_dq4}
set_placement -fixed -batch {p:sd_a[5]} {d:pad_ws_byteio6_dq5}
set_placement -fixed -batch {p:sd_a[6]} {d:pad_ws_byteio6_dq6}
set_placement -fixed -batch {p:sd_a[7]} {d:pad_ws_byteio6_dq7}
set_placement -fixed -batch {p:sd_a[8]} {d:pad_ws_byteio6_dq8}
set_placement -fixed -batch {p:sd_a[9]} {d:pad_ws_byteio6_dq9}
set_placement -fixed -batch {p:sd_clk_p[2]} {d:pad_ws_byteio6_dqs}
set_placement -fixed -batch {p:sd_clk_n[2]} {d:pad_ws_byteio6_dqsn}
set_placement -fixed -batch {p:sd_a[10]} {d:pad_ws_byteio7_dq0}
set_placement -fixed -batch {p:sd_a[11]} {d:pad_ws_byteio7_dq1}
set_placement -fixed -batch {p:sd_a[12]} {d:pad_ws_byteio7_dq2}
set_placement -fixed -batch {p:sd_a[13]} {d:pad_ws_byteio7_dq3}
set_placement -fixed -batch {p:sd_ba[0]} {d:pad_ws_byteio7_dq4}
set_placement -fixed -batch {p:sd_ba[1]} {d:pad_ws_byteio7_dq5}
set_placement -fixed -batch {p:sd_ba[2]} {d:pad_ws_byteio7_dq6}
set_placement -fixed -batch {p:sd_we_n} {d:pad_ws_byteio7_dq7}
set_placement -fixed -batch {p:sd_cas_n} {d:pad_ws_byteio7_dq8}
set_placement -fixed -batch {p:sd_ras_n} {d:pad_ws_byteio7_dq9}
set_placement -fixed -batch {p:sd_clk_p[3]} {d:pad_ws_byteio7_dqs}
set_placement -fixed -batch {p:sd_clk_n[3]} {d:pad_ws_byteio7_dqsn}

set_placement -fixed -batch {i:i_ddr3xN_phy_w_ctrl_core.i_ddr3xN_phy_w_controller.
i_ACX_DDRCONTROLLER} {s:x_iobank_ws.sdram_ddr123_lb_w_sif_wrapper.x_sdram_ddr123_lb_w_sif}

```

Chapter - 11: Design Simulation

Testbenches and Memory Models

Achronix provides two methods for simulating the DDR controller:

- A cycle-accurate model of the DDR3 controller, connected to industry-supplied, cycle-accurate models of DDR3 memory devices.
- A fast non-cycle-accurate behavioral model that replaces the DDR3 controller and DDR3 memory in circuit.

DDR3 Controller Model

A cycle-accurate model of the DDR controller is included within the ACE simulation libraries. To include this model, ensure the following file and library path is included in the simulation file.



In the file lists below, ACE_INSTALL_DIR refers to the ACE installation directory (where the ace executable is found).

```
+incdir+$ACE_INSTALL_DIR/libraries  
  
$ACE_INSTALL_DIR/libraries/device_models/22i_simmodels.v
```

DDR3 Memory Model

The cycle-accurate DDR3 memory model used in the reference designs is owned and copyrighted by Micron Technology. It is included in the reference design; however, it is also free to download from the Micron website at www.micron.com. All trademarks and the copyright of this work are duly acknowledged.

The memory model consists of six files;

- `ddr3.v` – The cycle-accurate model of a DDR3 device.
- `xxxxMb_ddr3_parameters.vh` – Parameterization files for each size of supported device.
- `ddr3_dimm.v` – Multiple DDR3 memory modules, linked to form the model of DIMM.

The DDR3 model is able to be parameterized to support from 512-Mb to 4-Gb devices. The parameterization is controlled by a number of Verilog defines, which can be specified either in the compiler commands or directly in the code. For the supplied reference designs, these defines are specified in the code at the top of the file `ddr3.v`. In all Achronix-supplied reference designs, the DDR3 model is set to be a 8-bit wide, 2-Gb device, running at 1600 MT/s (-125 speed grade).

The DDR3 model uses an associative array to model the memory to allow for a certain number of transactions to be stored. The size of this array is controlled by the MEM_BITS parameter in the appropriate file `xxxxMb_ddr3_parameters.vh`. Increasing the size of this array allows for more transactions to be modeled at the cost of increased simulation memory requirements. If the full size of the memory is required, then the Verilog define ``MAX_MEM` has to be defined.

Fast Behavioral Model

A fast behavioral model is provided in the PCIe Accelerator Card 6D reference design. This design is available from the Achronix secure FTP site, under public/reference designs. Once the design is downloaded, the model is located in `/src/tb/models/tb_ACX_DDR3_INTERFACE.v`.

The model works by replacing the entity `ACX_DDR3_INTERFACE` in the simulation library with the new behavioral entity. This model handles all transactions internally and does not require the DDR3 memory model. All the DDR3 interface signals are tied to fixed values to reduce X's in the simulation waveforms.

The amount of memory used by this behavioral model is controlled by an internal parameter `ADDR_SIZE_BITS`. This parameter can be modified to allow for a greater range of memory addresses to be modeled; however, at the cost of increased simulation memory requirements. In order to use this fast model, the file needs to be included AFTER the cycle-accurate DDR3 controller model as shown below:

```
# Include cycle-accurate DDR3 controller, (and other) models
+incdir+$ACE_INSTALL_DIR/libraries
$ACE_INSTALL_DIR/libraries/device_models/22i_simmodels.v

# Overwrite cycle-accurate DDR3 controller with behavioural model
../src/tb/models/tb_ACX_DDR3_INTERFACE.v

# No requirement to include DDR3 model in file list
# ../src/tb/models/ddr3_dimm.v
# ../src/tb/models/micron_ddr3.v
```

PAC 6D Card Example

The PCIe Accelerator Card 6D reference design includes the DDR3 fast behavioral model. When this model is used, it is also possible to remove the instantiation of the DDR3 memory model and DIMM from the top-level testbench in order to further speed up the simulation.

In `/src/tb/tb_pac_board.v`, set the following:

```
// Set USE_DDR_INT_MODEL if we have included tb_ACX_DDR3_INTERFACE in the compilation
// This then uses a simplified model in situ.
`define USE_DDR_INT_MODEL 1
```

Simulation Development Cycle

In a development cycle, the two models can compliment each other. For initial development to ensure that all signals are connected correctly, and that start-up reset and leveling sequences operate as expected, the cycle-accurate DDR controller and memory model should be used. This model provides a cycle-accurate representation of the DDR controller and DDR memory operation and should always be used as the final guide to the circuit operation. Due to being cycle-accurate, this model requires that the simulation complete the read and write leveling processes before the DDR controller is ready to accept transactions. Completing this process ensures that the DDR signal interfaces are all connected correctly.

In a simulation environment, the time point to achieve DDR controller initialization is approximately 250 μ s. With multiple DDR controllers, and possibly other circuitry such as SerDes models present, reaching this time point in a simulation can take between several minutes and possibly up to one hour (the actual time is dependent upon many factors: the number of DDR controllers, the speed of the simulation processor, the simulator used, the complexity of the rest of the circuit, and the simulation optimizations set). For this phase, if practical, a reduced number of DDR controllers can be used.

Once initial operation including the DDR controller initialization cycle and the first few transactions is proven, then the faster non-cycle-accurate model can be used. Changing to this model allows for more rapid development of DDR-based clients as simulation times are greatly reduced. It is important to realize that the DDR controller signals from the behavioral model have fixed approximate timing with regard to delays between transactions and the initialization sequence. However, within a single transaction, their sequencing and relationship to each other is accurate. This model, used in conjunction with the detailed waveforms for read and write transactions, allows for quick development of any user block that interfaces to the DDR controller.

Table 24: Suggested Development Cycle for the DDR Controller

Phase	Simulation Environment	Comments	Simulation Speed	Included in Reference Design		
				RD005	RD010	PAC Card
Initial single DDR	Single DDR controller. DDR memory model	Prove correct connection of clocks, and DDR memory interface signals. PLL frequencies. DDR controller initialization cycle should complete (<code>ddr_int_ready</code> be asserted).	Medium	Y	Y	Y
Single first transactions		Prove user code can perform simple write and read operations to single DDR. Alternatively use provided DDR tester to prove correct DDR client side operation.	Medium	Y	Y	Y
Multiple DDR controllers	Multiple DDR controllers, each connected to DDR memory models	Prove correct read and write leveling state machines and shared reset schemes. All DDR controllers should assert <code>ddr_int_ready</code> . User logic should be gated until all <code>ddr_int_ready</code> signals asserted.	Slow	N	Y	Y
Multiple first transactions		Prove user code can perform simple write and read operations to all DDR controllers. Alternatively use provided DDR tester to prove correct DDR client side operation for all DDR controllers	Slow	N	Y	Y
DDR client development	Multiple behavioral models	Switch simulation to multiple behavioral models. Develop DDR client-side applications and dataflow with rest of user code. Fast code turnaround. Develop verification test suite.	Fast	N	N	Y
Sign-off verification	Multiple DDR controllers, each connected to DDR memory models	Run verification suite using cycle-accurate models to ensure that DDR client-side applications meet all requirements.	Slowest	N	N	N

Chapter - 12: Debug Using the DDR3 Bring-up Design

DDR3 Bring-Up Designs

Achronix provides a set of bring-up designs to help users during development. The Speedster22i DDR3 bring-up designs are structured such that they can be tailored to be used on any board, including the the Achronix development board. The user can choose to test a single instance on any of the six DDR3 locations, or alternatively use one of the variants of the four or six-controller designs to test multiple instances in parallel with the chosen architecture.

There are a total of two packages with five design variants. Each package has its own set of source code, one or more testbenches, and appropriate timing and placement constraints.

Package 1

Package one contains a single-controller design which can be placed on any one of the six DDR3 locations. Included with the release are placement constraints to test the $\times 64$ WS SO-DIMM interface and the $\times 16$ WC discrete device interfaces on the Achronix development board. Additional $\times 72$ UDIMM and $\times 64$ SO-DIMM constraints for the Achronix characterization board have also been provided for reference.

Package 2

Package two contains four multi-controller designs:

- Four-controller design for the north and center DDR3 interfaces optimized to use low-jitter boundary clocks.
- Four-controller design for the south and center DDR3 interfaces optimized to use low-jitter boundary clocks.
- Six-controller design using all the DDR3 interfaces optimized to use low-jitter boundary clocks.
- Six-controller design using all the DDR3 interfaces using global boundary clocks and with a minimum number of core-generated I/O resets.

The first three all have their own individual testbenches, and each one of the variants comes with its own set of placement and timing constraints. Besides the top-level module which defines the architecture, all of the remaining source code is shared between the variants. The designs provide for the capability to test all supported DDR3 data widths and data rates and additionally has options to set various parameters, drive on-board LEDs or take advantage of the Snapshot in-system debugger.

Bring-Up Methodology

Signal Integrity Analysis

The right place to start with any DDR3 bring-up is to look at the signal integrity on the board. The single-controller design should be generated to observe the signaling and behavior for one of the four interfaces at the lowest supported data rate of 1066 Mbps. Furthermore, even if dual-rank (and/or dual-slot) DIMMs are intended to be used, starting the bring-up with single-rank DIMMs makes the most sense.

The SI analysis can be best done using DIMM and component memory interposers, which are passive elements that enable observations of the signal levels, including timing, on a scope (contact Achronix for specific

equipment that can be used for this purpose). Many of the commercially available solutions allow for extensive DDR3 electrical and protocol compliance analysis. One thing to keep in mind is that these solutions almost always allow for observation at the DIMM, which is very useful for write-side analysis, but read-side analysis on the FPGA side is equally useful if it can be planned for during board development with test points and the like.

The signal integrity analysis, which can start with the baseline case of 1066 Mbps with single-rank DIMMs, can progressively be done at higher data rates and multi-rank memories. Based on the results from the data collection, the following DDR3 parameters can be investigated and tuned as needed to ensure desired swing, termination, and clean operation:

- FPGA write drive strength (drive) – Currently auto-set to a value between 16 mA and 32 mA depending on the number of ranks.
- FPGA on-die termination (termination) – Can be set to a wide range of values between 40Ω and 120Ω. Most commonly used settings are 50Ω and 75Ω.
- DIMM ODT (odt_dyn_term) – This can be set to various combinations of static vs dynamic termination at 40/60/120Ω. The current default settings are to use static with 40Ω for single-rank and dynamic with 60Ω for multi-rank DIMMs.
- DIMM drive strength (mem_drive_strength – available starting in ACE 6.0) – Allows for toggling the setting between 34Ω (default) and 40Ω.

Functional Debug and Tuning

Assuming that at the completion of the signal integrity analysis, the compliance checks and signaling are all aligned to JEDEC specs, it is possible to move forward with functional debug.

The same single-controller design can be used to kick-start the functional debug at 1066 Mbps using single-rank DIMMs. The two mechanisms used for observability and success of the DDR3 BIST design are:

- LED outputs – Refer to the LEDs section above to ascertain the LED mapping and expected behavior after bitstream programming or reset.
- Snapshot monitor signals – The monitor bus can be modified to provide indicators of the desired signals. The three classes of signals that need to be verified are the following:
 - State machine control signals and indicators – some of these should include controller_init_done, test_done, pass and state machine states. Some of these are already routed to LEDs, but it is still a good idea to observe them here.
 - Leveling values – Ideally the full array of write-leveling and read-leveling values for all byte lanes should be on display. It is important to confirm that these values lie within legal ranges, that none of them are 0s or saturated, and that the values make sense for each of the component memories given the delays on the DIMM. Please refer to [DLL and Write/Read Leveling \(see page 62\)](#) to better understand expected legal ranges for the write-leveling and read-leveling values.
 - Expected vs read data – Comparisons between the compare data and actual data from the memory can provide indications of data integrity as well as system latency issues. These comparisons are especially useful when counting data is used for debug (in the place of pseudo-random LFSR data).

For stability and regression testing, it is recommended that the test be run:

- For extended periods of time (overnight) with VT stress conditions across parts to test for stability.
- Across hundreds or thousands of reset or programming cycles to test for robustness and consistency of the leveling algorithms and controller initialization.

However, the first-level debug can be done by simply running the test a few times, and only for a few seconds each time. Gross errors can easily be caught using this methodology.

It is very possible that CLK and CAC settings will need to be tuned in order to get each of the individual DDR3 interfaces to have clock, command-address-control and dq/dqs to align properly. This tuning needs to be done by shifting the CLK_DELAY and CAC_DELAY settings appropriately to ensure that the write-leveling and read-leveling values are legal and fall within expected ranges.

If after tuning, all of the above observations indicate that the test is working perfectly, it would make sense to proceed with the next steps, but it is likely that one or more issues may be encountered or still exist when running this test. The sections below highlight some of the more commonly seen problems and where to direct the debug effort in such cases.

Single-controller Design Proliferation

Once the single-controller design works as expected at 1066 Mbps for the first location under test with single-rank DIMMs, the next steps are to:

- Bridge the data rate gap to verify correct operation up to 1600 Mbps.
- Ensure that all locations can be tuned or work as expected at the desired data rates.
- Tailor design to use the required memory, configuration and clocking modes.

Multi-controller Design Implementation and Expansion

The final step is to port all the known settings, changes and learnings from the single-controller design to the four-controller NC design and build it at the different data rates and configurations for the board.

Even if the single-controller design implementation and proliferation was successful across all data rates and configurations, it is still worthwhile, at the very least, to start out at 1066 Mbps and move up to 1600 Mbps. Dual-rank and local/global boundary clock usage settings can be preserved and should not require re-verifying with simpler baselines.

Chapter - 13: Reference Designs

DDR3 Bring-Up Design

Achronix provides a set of bring-up designs to help users during development. The Speedster22i DDR3 bring-up designs are structured such that they can be tailored to be used on any board, including the the Achronix development board. The user can choose to test a single instance on any of the six DDR3 locations, or alternatively use one of the variants of the four or six-controller designs to test multiple instances in parallel with the chosen architecture.

There are a total of two packages with five design variants. Each package has its own set of source code, one or more testbenches, and appropriate timing and placement constraints.

Refer to *Speedster22i DDR3 Bring-Up Design User Guide* (RD010) for more details.

PCIe Accelerator 6D Card Multi-IP framework

The Multi-IP framework reference design, targeted to the PCIe Accelerator 6D card, (PAC-6D), includes six DDR controllers operating at 1600MT/s. The design demonstrates correct configuration of clocks and resets for the full set of DDR controllers available on the HD1000. The design supports read and write leveling during initialization.

Refer to *PCIe Accelerator-6D Card Multi-IP Reference Design User Guide* (RD014) for more details.

Revision History

Version	Date	Description
3.0	November 19, 2018	<ul style="list-style-type: none">• Complete review and revision.