CoreQDR v3.3

Handbook

Microsemi
a Microchip company

Table of Contents

Introduction 3
About QDR 3
QDR Supported Families and Devices 3
Core Overview 4
Key Features 5
Core Version 5
Supported Families 5
Utilization and Performance 5

Interface Description 7
Parameters/Generics 7
I/O Signals 7
I/Os 9

Operations 10
Clock Generation and Data Registering 10
Burst-of-2 Operation 11
Burst-of-4 Operation 12
User Interface Timing 13

Tool Flows 15
Licensing 15
SmartDesign 15
Simulation Flows 15
Synthesis in Libero SoC 15
Place-and-Route in Libero SoC 15

Implementation Hints 16
CCC Configuration 16
Constraints 18

List of Changes 25

Introduction

About QDR

Quad data rate (QDR) synchronous dynamic random access memories (SDRAMs) are a family of SRAMs with separate read and write channels, each operating at double data rate (DDR), optimized for high-performance communication applications such as the data plane memory of routers. QDR SDRAM performs functions such as packet buffering, statistics counting, and flow rate control. The specifications for QDR are created and maintained by the QDR consortium, which comprises several companies, including Cypress Semiconductor and Renesas Electronics.

QDR Supported Families and Devices

Table 1 outlines the various QDR flavors.

Table 1 ·QDR Families (source)

QDR Family		**QDR I**	**QDR II**	**QDR II+**	**QDR II+ Xtreme**
Frequency	2-word burst	166 MHz	333 MHz	333 MHz	400 MHz
	4-word burst	200 MHz	333 MHz	500 MHz	633 MHz
Latency		1 Cycle	1.5 Cycles	2 Cycles	2.5 Cycles
Clocks		No echo CLKs	Echo CLKs	Echo CLKs	Echo CLKs
Density		9Mb/18Mb	18/36/72 Mb	18/36/72/144Mb	36/72 Mb

Core Overview

CoreQDR provides a soft IP controller for interfacing with the QDR, QDR II, or QDR II+ SRAMs. Figure 1 shows the basic use of CoreQDR in a system.

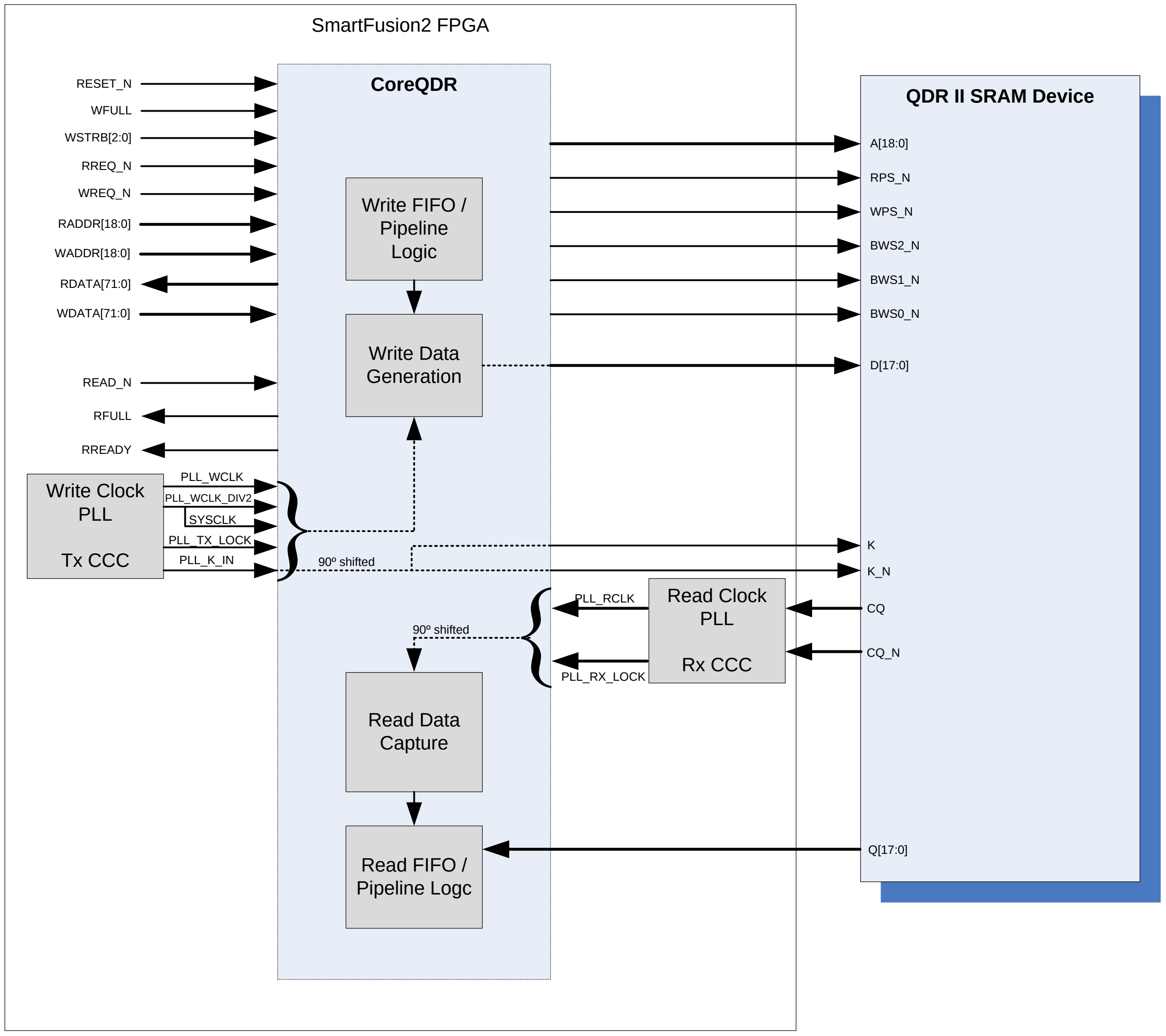


Figure 1 CoreQDR System

Note:

- In Figure 1, burst-of-2 Operation with an SRAM width of 18 is Configured
- For optimal performance and to easy Place-and-Route/timing closure, it is recommended to select an SRAM device with QVLD support, that is, QDR II+ devices and set USE_QVALID to 1. For more information, refer to the Parameters/Generics section below.
- The actual Read and Write Clock PLL's are implemented in the device but outside of CoreQDR. For more information, refer to the relevant application note on Meeting Timing for CoreQDR in SmartFusion2. For examples on how to instantiate these blocks, please refer to the top-level testbench file *coreqdr_tb.v*.
- The examples given in this document show a system clock (PLL_WCLK_DIV2) of 166 MHz and QDR clocks (RCLK, WCLK, K/K#, and CQ/CQ#) of 333 MHz. For certain designs (depending on the overall system), these frequencies may not be achievable, in which case the clock rate may always be scaled down.

Key Features

CoreQDR is a configurable memory controller for QDR static random access memory (SRAM) devices and has the following features:

- Supports QDR II Interface
  - Up to 666 MHz double data rate (333 MHz clock)
  - Separate read and write channels D (input) and Q (output), supporting concurrent transactions
  - Single address channel A
  - Burst of 2 and burst of 4 support (configurable)
- Configurable clock cycle latency using configuration input ports
  - Clock cycle (coarse latency)
  - Clock edge (fine latency)
- Configurable QDR data width D/Q (8, 9, 18, 36 bits)
- Two phase-locked loops (PLLs), instantiated outside of CoreQDR; top-level PLL inputs in CoreQDR
  - One to generate true write clock (K and K_n)
  - One to generate 90 degree phase shifted clock from Echo (read) clock
- Two-Port FIFO's for clock synchronization/pipelining
  - Write Data Path
  - Read Data Path
  - Write Address Path
  - Read Address Path
- Configurable use of QVLD signal
  - When enabled, QVLD signal is used and there is no write-read clock crossing
  - When disabled (for QDR I / QDR II devices not supporting QVLD), CoreQDR generates an internal data valid signal

Core Version

This handbook applies to CoreQDR v3.3.

Supported Families

- SmartFusion®2
- RTG4™

Utilization and Performance

Table 2 and Table 3 show CoreQDR utilization and performance for two configurations (burst-of-2 and burst-of-4) at 18-bit data width (maximum configurable).

Table 2 ·Burst-of-2 Utilization and Performance

Family	Logic Elements					RAM Usage	Speed
	Sequential	Combinatorial	Total	Device	Total		
SmartFusion2	893	558	1,451	M2S050T	2%	12 RAM1K18 blocks	167 MHz
RTG4	1,291	1,586	2,877	RT4G150	2%	12 RAM1K18 blocks	100 MHz

Note: In burst-of-2 configuration with SRAM_WIDTH = 18

Table 3 ·Burst-of-4 Utilization and Performance

Family	Logic Elements					RAM Usage	Speed
	Sequential	Combinatorial	Total	Device	Total		
SmartFusion2	871	500	1,371	M2S050T	2%	12 RAM1K18 blocks	167 MHz
RTG4	1,244	1,581	2,825	RT4G150	2%	12 RAM1K18 blocks	100 MHz

Note: In burst-of-4 configuration with SRAM_WIDTH = 18

Interface Description

CoreQDR consists of a QDR II interface on one side for controlling a QDR memory chip as well as capturing data from it and generating data to it. On the other side, there is a generic user interface, which consists of basic write/read selects, separate read and write data buses, and an input clock PLL_WCLK_DIV2. For more information on top-level ports, refer to the I/O Signals section.

Parameters/Generics

All the parameters (Verilog) and generics (VHDL) for CoreQDR are described in Table 4. All parameters and generics are positive integer types.

Table 4 ·CoreQDR Parameter/Generic Descriptions

Name	Valid Range	Default	Description
FAMILY	19, 25	19	Must be set to the required field programmable gate array (FPGA) family: 19: SmartFusion2 25: RTG4
SRAM_DWIDTH	8, 9, 18, 36	8	SRAM data width; specifies the width, in bits, of data buses D and Q Note: This IP does not support x36 mode for SmartFusion2.
SRAM_AWIDTH	19, 20, 21	19	SRAM address width; specifies the width, in bits, of address bus A
SRAM_BURST	0 or 1	0	Burst size 0: Burst-of-2 operation is used 1: Burst-of-4 operation is used For more information, refer to the Burst-of-2 Operation, Burst-of-4 Operation, and User Interface Timing sections.
USE_QVALID	0 or 1	0	QVLD enable When 0: CoreQDR does not use QVLD input When 1: CoreQDR uses USE_QVALID as an enable to register data from the QDR memory. This is enabled only for QDRII+ and higher devices.

I/O Signals

The input/output (I/O) ports are listed in Table 5.

Table 5 ·CoreQDR Signals and Descriptions

Name	Width	Type	Description
Configuration Ports			
LAT_SEL	3	Input	Latency select Note: 1. This port is only used when Q_VALID is 0. When Q_VALID is 1, there is no need to generate an internal read valid signal. 2. This value should be tweaked post-layout, as the clock cycle that the data appears valid to CoreQDR is dependent on Place and Route results. It is vital that the user run post-layout simulations.

Name	Width	Type	Description
			This port determines how many clock cycles after RPS_N is asserted data becomes valid. 0: 1 cycle 1: 2 cycles 2: 3 cycles … 7: 8 cycles
Q_EDGE	1	Input	Edge select A finer-grained tuning of expected data arrival, this port selects whether the first data coming from the QDR SRAM is expected on the rising edge or the falling edge. 0: Rising Edge 1: Falling Edge Note: Q_EDGE is used only when USE_QVALID is 0.
Clocks and Reset			
Note: All signals prefaced with "PLL_" should be outputs of a CCC that the user must instantiate in the top-level of their design.			
PLL_WCLK_DIV2	1	Input	System clock, drives the write data path in the 166 MHz domain.
PLL_TX_LOCK	1	Input	Lock output of CCC used for generating write clock from system clock.
PLL_RX_LOCK	1	Input	Lock output of CCC used for generating read clock from CQ/CQ#.
PLL_RCLK	1	Input	90-degree phase-shifted output of a CCC for which CQ is the reference clock.
PLL_WCLK	1	Input	A 333 MHz clock must be synchronous with PLL_WCLK_DIV2 and 2x the frequency of PLL_WCLK_DIV2.
PLL_K_IN	1	Input	A 333 MHz clock must be 90 phase-shifted with regards to PLL_WCLK.
RESET_N	1	Input	System reset, active low
User Interface Signals			
RREQ_N	1	Input	Read request signal, sampled on the rising edge of PLL_WCLK_DIV2. Requests a read operation by loading the RADDR into the address FIFO. Can be asserted at the same time as WREQ_N. Active low.
RREADY	1	Output	Indicates that there is data waiting in the READ FIFO.
RFULL	1	Output	Indicates that the read FIFO is full. Subsequent read transactions will be lost.
READ_N	1	Input	When RREADY is asserted, this signal can be asserted to pull data from the READ FIFO; data appears on the read data bus two cycles after READ_N is asserted.
WREQ_N	1	Input	Write request signal, sampled on the rising edge of PLL_WCLK_DIV2. Write data must be presented on this same cock cycle at the data bus WDATA. Write data then gets written to the write FIFO and subsequently the QDR memory, following the standard QDR II timing. Can be asserted at the same time as RREQ_N. Active low.
WFULL	1	Output	Indicates that the write FIFO is full. This should not typically happen, since data should be cleared to the FIFO as quickly as it is written onto the QDR interface.
WADDR	SRAM_AWIDTH	Input	Write address, sampled on the rising edge of PLL_WCLK_DIV2.
RADDR	SRAM_AWIDTH	Input	Read address, sampled on the rising edge of PLL_WCLK_DIV2.
RDATA	SRAM_DWIDTH*4	Output	Read data, output at rising edge of PLL_WCLK_DIV2.

Name	Width	Type	Description
WDATA	SRAM_D WIDTH*4	Input	Write data, sampled on the rising edge of PLL_WCLK_DIV2.
WSTRB	3	Input	Reserved, currently unused.
Interrupts			
CLK_READY	1	Output	Indicates that the RX and TX clocks are locked and ready. Data should not be written or read until this output is asserted. RESET_N should be held low until CLK_READY is asserted.
QDR SRAM Signals			
Control			
RPS_N	1	Output	Read port select, active low
WPS_N	1	Output	Write port select, active low
BWS2_N	1	Output	Byte write select 2, active low; used for 18-bit memory only
BWS1_N	1	Output	Byte write select 1, active low; used for 9-bit and 18-bit memories
BWS0_N	1	Output	Byte write select 0, active low; used for 8-, 9-, and 18-bit memories
NWS1_N	1	Output	Nibble write select 1, active low; used only for 8-bit memories
NWS0_N	1	Output	Nibble write select 0, active low; used only for 8-bit memories
A	SRAM_A WIDTH	Output	QDR address bus
Write Channel			
K	1	Output	Write clock
K_N	1	Output	Inverted write clock
D	SRAM_D WIDTH	Output	Write data bus
Read Channel			
Q	SRAM_D WIDTH	Input	Read data bus
Q_VALID	1	Input	Data valid signal

Notes:

1. CQ and CQ_N are only provided for QDR II/QDR II+ mode, and are used to derive the PLL_RCLK signal.
2. All signals are active high unless otherwise indicated.

I/Os

QDR II requires the HSTL I/O standard using a VREF of 0.7 V and a VDDIO of 1.5 V, which is supported by the SmartFusion2 DDRIO. A termination voltage of VDDIO/2 is required, in this case 0.75 V, which is also supported by SmartFusion2 DDRIO.

For more information on supported DDRIO modes, refer to the *SmartFusion2 Datasheet*

For more information on the HSTL I/O standard, visit www.jedec.org.

Operations

Clock Generation and Data Registering

The QDR interface consists of two write clocks, K and K_N, and two read clocks, CQ and CQ_N. For optimal data capture, CoreQDR internally shifts both the write clocks and the read clocks by 90 degrees and uses these shifted clocks to capture data. This ensures that timing is met and that the clock edges are roughly in the middle of the data capture window.

To that end (meeting timing), the nominal clock domain of 333 MHz (666 MHz DDR) must be slowed to a 167 MHz single data rate (SDR) using multiplexing/demultiplexing logic. Essentially, the data rate is traded for bus width – that is, lowering the local bus data rate by increasing its width. This presents unique challenges (and requires much more heavy pipelining), but allows the main CoreQDR controller logic run at a more manageable 167 MHz. This applies to both the read datapath and the write datapath. The address path need not be widened.

Figure 2 and Figure 3 on page 11 show the clock generation scheme and data registering/generation scheme used in CoreQDR.

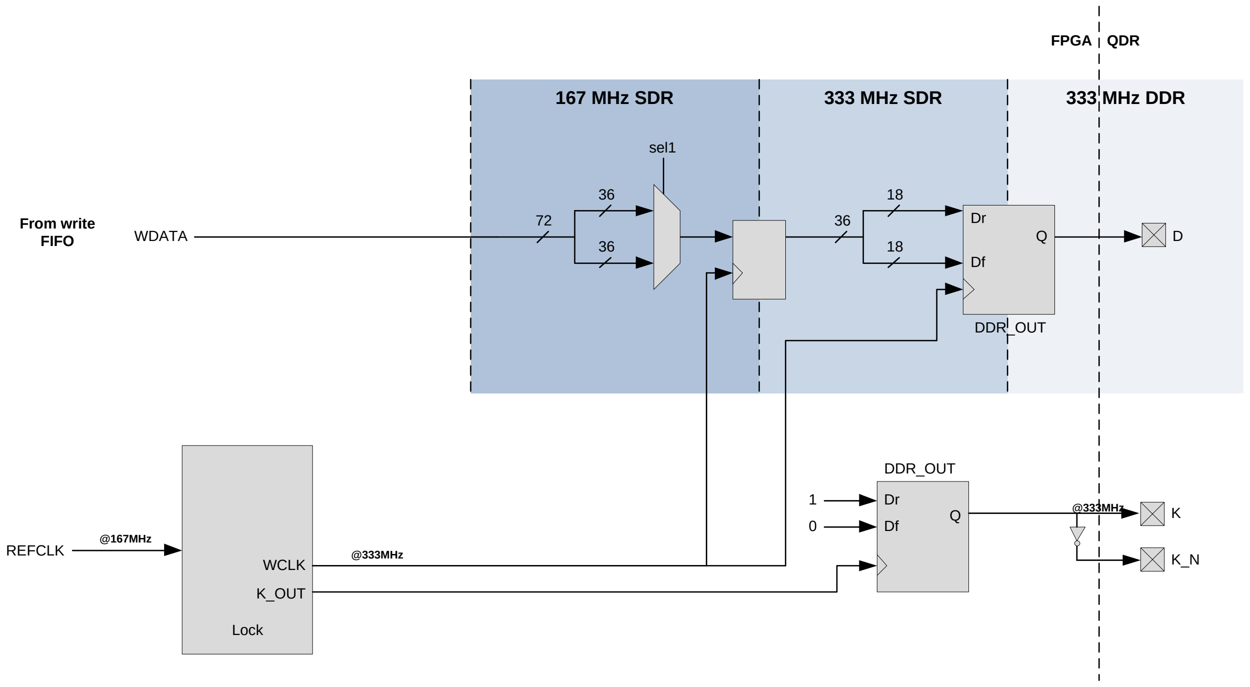


Figure 2 ·QDR Write Datapath and Clocks for SRAM_WIDTH = 18

In Figure 2, the data bus is first narrowed, and the data rate increased, by a factor of 4. For the case of burst-of-2 operation, this is done by a factor of 2. The final step in the datapath to the QDR II memory is register the data using the DDR_OUT output register, which acts as a sort of multiplexer (MUX), selecting Dr on the positive edge and Df on the negative edge.

The output K and its complement K_N are generated using a 270 degree output clock operating at 2 times the system clock frequency (333 MHz) and an invert located in the output I/O, which comes free in the DDRIO block.

Note: The TX PLL as shown above must be instantiated in the top-level of the design. You must generate and route the clock signals from the CCC to the appropriate inputs of CoreQDR (signals proceeding with PLL_WCLK* as described in the Port List).

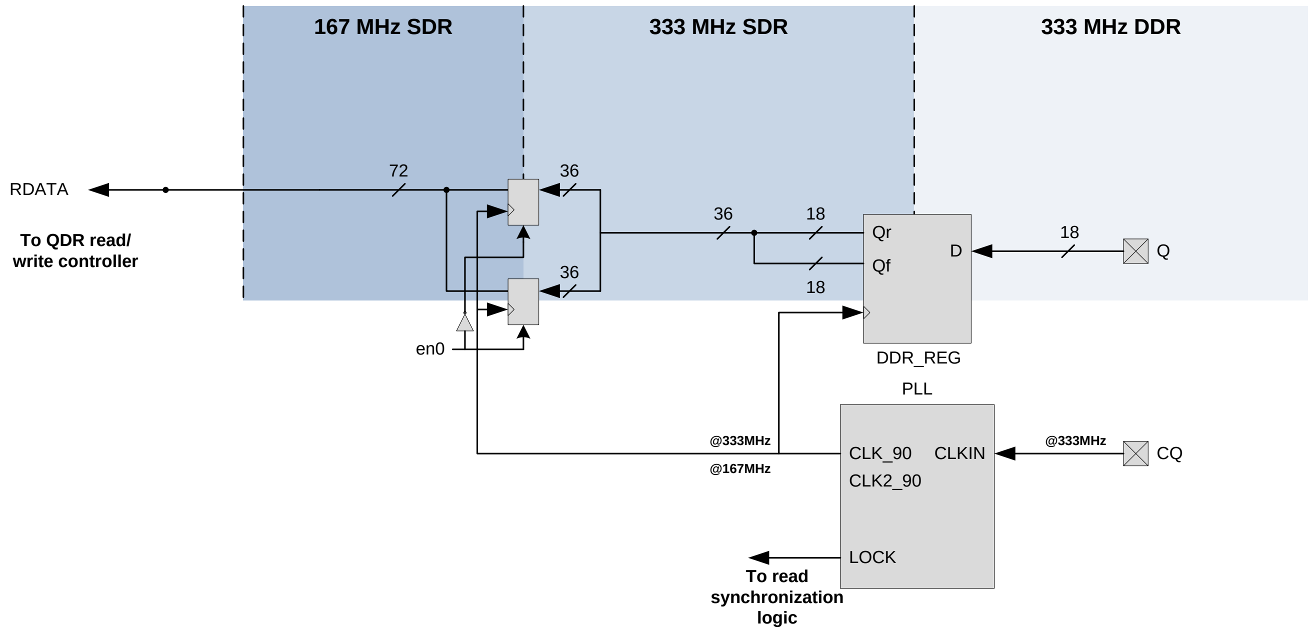


Figure 3 ·QDR Read Datapath and Clocks for SRAM_WIDTH = 18

In Figure 3, a simplified version of read data is registered from the QDR SRAM using the echo clock; the data is edge-aligned to the echo clock CQ. CQ_N is not used because the DDR_REG block only uses a single clock and registers data at both falling and rising edges. In order to meet timing, the read clock is also shifted by 90 degrees; this ensures that data is centered around the read clock, CQ. The data is then widened using 2 sets of registers and the register enables that alternate high and low on consecutive clock cycles.

Note:

- The RX PLL as shown above must be instantiated in the top-level of the design. You must generate and route the clock signals from the CCC to the appropriate inputs of CoreQDR (signals proceeding with PLL_R* as described in the Port List).
- To ensure that the clock routing delay does not erode timing margins, and to ensure that there is indeed a 90º phase-shift between PLL_RCLK and the data (Q). You instantiate a CCC with a dedicated Hardwired Input as the reference clock. They should also generate a 0º clock as one of the outputs of the CCC and connect it to a Fabric Input feedback port of that same CCC. This will mitigate clock routing delay between the clock PAD and the fabric.

Burst-of-2 Operation

The latencies shown in Figure 4 are contingent on timing being met, and are not guaranteed in the final design. During a burst-of-2 read operation, a read address must be presented on the *RADDR* bus of the local user interface. This is latched using the rising edge of the input clock *PLL_WCLK_DIV2*, along with the *RREQ_N* control signal, which indicates that a read is requested. After several clock cycles (the precise number depending on the status of the write and read FIFOs), the address appears on the address bus *A* at the rising edge of *K* clock signal. After a clock latency defined by the *Q_EDGE* and *LAT_SEL* inputs, the data appears on the QDR data bus, at which point it is latched at the rising edge and then falling edge (DDR) of the 90-degree shifted CQ echo clock from the QDR SRAM device. This latched data is stored in FIFO and the *RREADY* signal (active high) is asserted. It will be possible to read from the FIFO by asserting the *READ_N* input.

During a burst-of-2 write operation, a write address must be presented on the *WADDR* bus of the local user interface, as well as valid data on the *WDATA* line. These are latched using the rising edge of the input clock *PLL_WCLK_DIV2*, along with the *WREQ_N* control signal, which indicates that a write is requested using the current address and data. After some time, CoreQDR presents the least significant half of the data (for example, *WDATA[7:0]* for an 8-bit memory) on the rising edge of *K*. On the subsequent falling edge of *K*, the most significant half of the data (*WDATA[15:8],* for example), is presented at the same time as the latched address is presented on *A*.

Since QDR uses separate read and write channels and since the shared address bus uses rising edge *K* capture for reads and falling edge *K* capture for writes, it is possible to conduct concurrent read and write operations with no change in timing from read-only or write-only.

Figure 4 shows the waveform for both read and write transactions at maximum throughput (back-to-back transactions). This waveform only describes the deterministic QDR II timing, not the local interface.

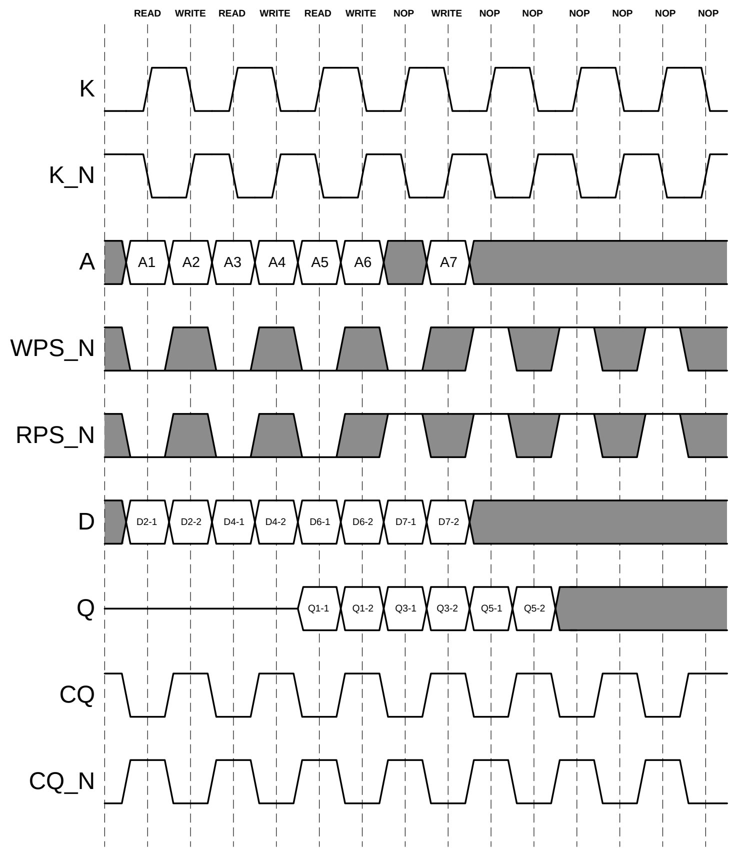


Figure 4 ·Burst-of-2 QDR Operation

Burst-of-4 Operation

Burst-of-4 operation is similar to burst-of-2, except that the user interface data width is twice as wide (that is, four times data width overall), and each transaction on that side produces a burst-of-4. As a result, consecutive transactions of the same type (read or write), cannot be performed on consecutive clock cycles, since it takes 2 full clock cycles for a burst-of-4 to occur (that is, 2 rising edges and 2 falling edges).

Figure 5 shows the waveform for both read and write transactions at maximum throughput (back-to-back transactions). This waveform only describes the deterministic QDR II timing, not the local interface.

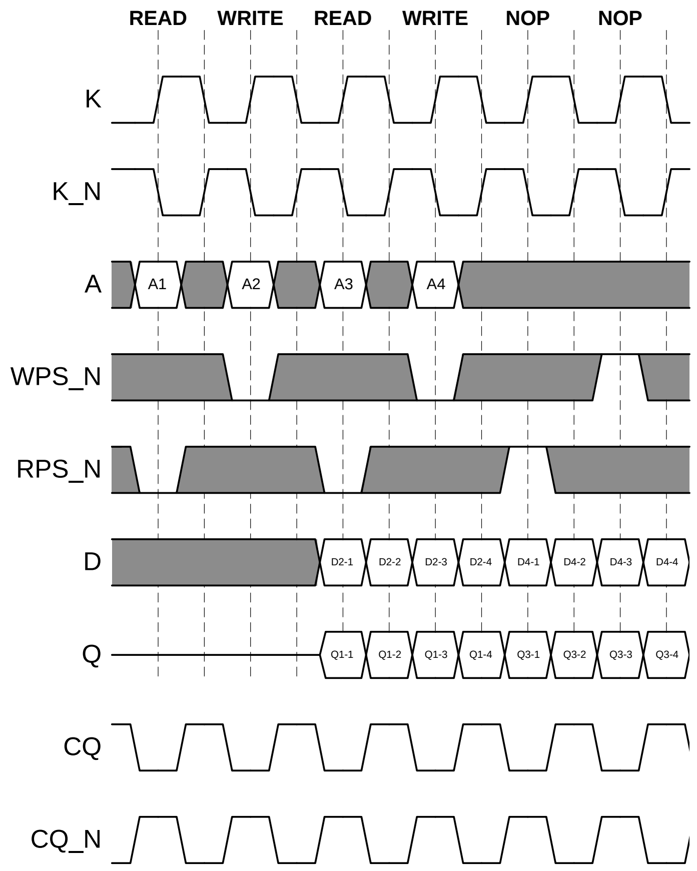


Figure 5 ·Burst-of-4 QDR Operation

User Interface Timing

All user interface signals, both control and data, are synchronous to the rising edge of PLL_WCLK_DIV2. On any given clock cycle (rising edge of PLL_WCLK_DIV2), the user may perform a write operation, a read operation, or both at the same time, as long as the WFULL is not asserted (for writes) and RFULL is not asserted (for reads).

Figure 6 shows a sample of reads and writes on the user interface. Control signals, interrupts, and status outputs are omitted for simplicity. Write transactions are straightforward; as long as the WFULL output (not shown) is not asserted, it is possible to perform a write by pulling the WREQ_N signal low. Write address (WADDR) and write data (WDATA) must be presented on the same clock edge as the WREQ_N control.

For read transactions, a read can be requested on any clock cycle by pulling the RREQ_N signal low, as long as the RFULL output (not shown) is not asserted. The read address (RADDR) must be presented on the same clock cycle as the RREQ. After several clock cycles[1], CoreQDR asserts the RREADY signal, indicating that there is data present in the read FIFO. When RREADY is asserted, the available data can be read by pulling the READ_N signal low. Read data is presented on the RDATA bus two clock cycles following the assertion of READ_N.

[1] *The exact number depends on the latency configuration of the core and whether it is operating in Burst-of-2 mode or Burst-of-4 mode.*

Note: For burst-of-2 operation, consecutive reads or consecutive writes should be performed at increments of addresses of 2, because each user-side transaction corresponds to two QDR bursts on the QDR interface. For burst-of-4 operation, consecutive operations should be performed on sequential addresses, because each user-side operation corresponds to a single QDR burst.

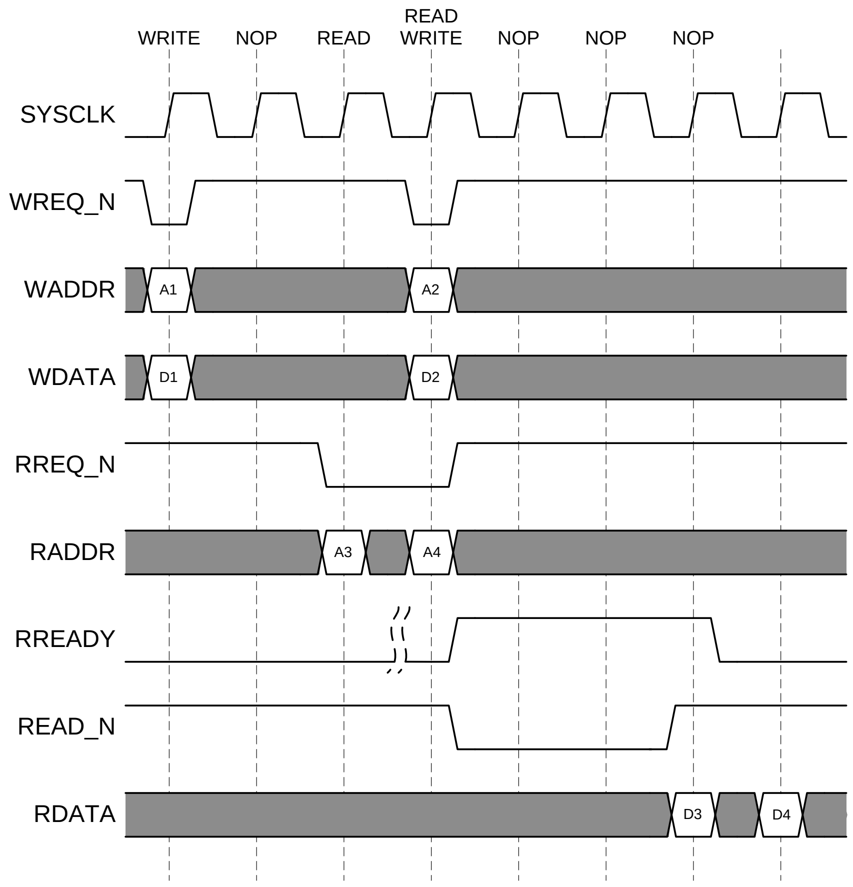


Figure 6 ·User Interface Timing (Burst-of-2 and Burst-of-4)

Note: In Burst-of-2 mode, each (4x data width) user interface read/write gets translated to 2 burst-of-2 read/writes on the QDR interface. In Burst-of-4 mode, each (4x data width) user interface read/write gets translated to a single burst-of-4 read/write on the QDR interface.

Tool Flows

Licensing

CoreQDR is available as clear RTL in Verilog and VHDL versions.

SmartDesign

CoreQDR is available for download to the Libero® System-on-Chip (SoC) IP Catalog through the web repository. Once it is listed on the catalog, the core can be instantiated using the SmartDesign flow. The core can be configured using the configuration GUI within SmartDesign.

For information on using SmartDesign to instantiate and generate cores, refer to the *Using DirectCore in Libero SoC User Guide*.

Simulation Flows

A testbench is included with the CoreQDR release.

To run simulations, select **Testbench: User** inside the CoreQDR configurator in SmartDesign. Click **Save and Generate** on the Generate pane.

When SmartDesign generates the Libero SoC project, it installs the User Testbench files.

User Testbench

The User Testbench includes the CoreQDR with a sample memory model. The memory model should be replaced with the appropriate model being used for the target application.

Synthesis in Libero SoC

Right-click on **Synthesize** in the **Design Flow** pane. Select **Configure Options**. Ensure that the **Verilog 2001** standard option is selected. To run Synthesis, right-click on **Synthesize** and select **Run**.

Place-and-Route in Libero SoC

*Note: To run place and route, right click on **Place and Route** and select **Run**. CoreQDR requires no special place-and-route settings.*

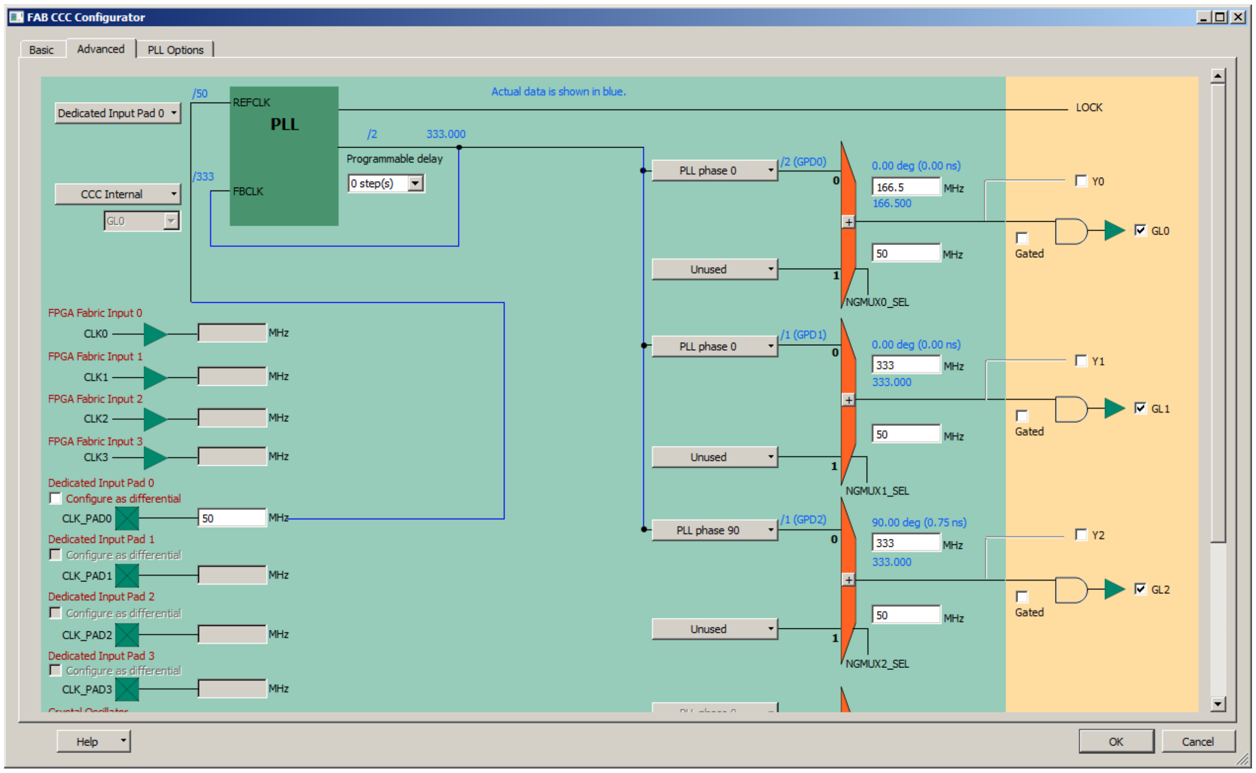
Implementation Hints

CCC Configuration

In order to operate CoreQDR at the required speeds, it is important to configure the two Clock Condition Circuits (CCC's) appropriately. These consist of a transmit clock CCC and receive clock CCC.

The transmit or write path CCC's input can be configured as per the system requirements and the outputs must be configured with a 333 MHz 0-degree write path launch clock, 333 MHz 90-degree write clock, and 166.5 MHz 0-degree system or user interface clock. Figure 7 and Figure 8 show an example of TX clock configuration in the CCC configurator in Libero SoC (v11.2) and its connection to CoreQDR.

Note: The GL0 is the 0-degree system clock (connecting to PLL_WCLK_DIV2), GL1 is the 333 MHz QDR launch clock (connecting to PLL_WCLK), and GL2 is a 90-degree phase-shifted version of GL1 used to pass through to K via DDR block to ease timing (connecting to PLL_K_IN).



Reference clock: Dedicated Input Pad 0, /50; CLK_PAD0: 50 MHz
Feedback: CCC Internal, GL0, /333
PLL output: /2, 333.000
Programmable delay: 0 step(s)

GL0: PLL phase 0, /2 (GPD0), 0.00 deg (0.00 ns), 166.5 MHz (166.500); Unused, 50 MHz; ✓ GL0
GL1: PLL phase 0, /1 (GPD1), 0.00 deg (0.00 ns), 333 MHz (333.000); Unused, 50 MHz; ✓ GL1
GL2: PLL phase 90, /1 (GPD2), 90.00 deg (0.75 ns), 333 MHz (333.000); Unused, 50 MHz; ✓ GL2

Figure 7 ·TX PLL Configuration

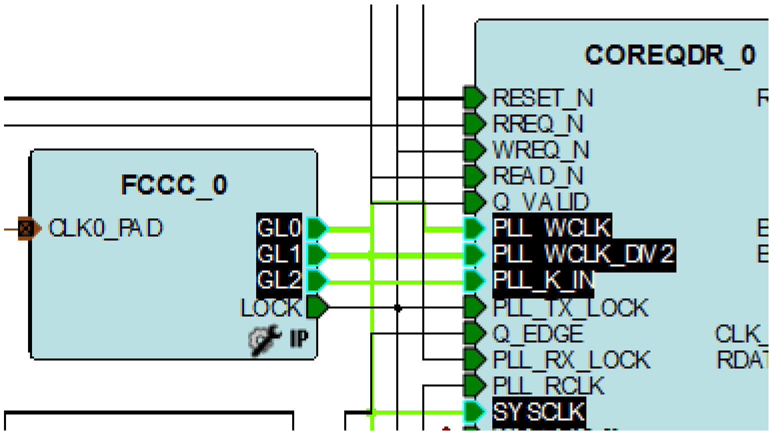
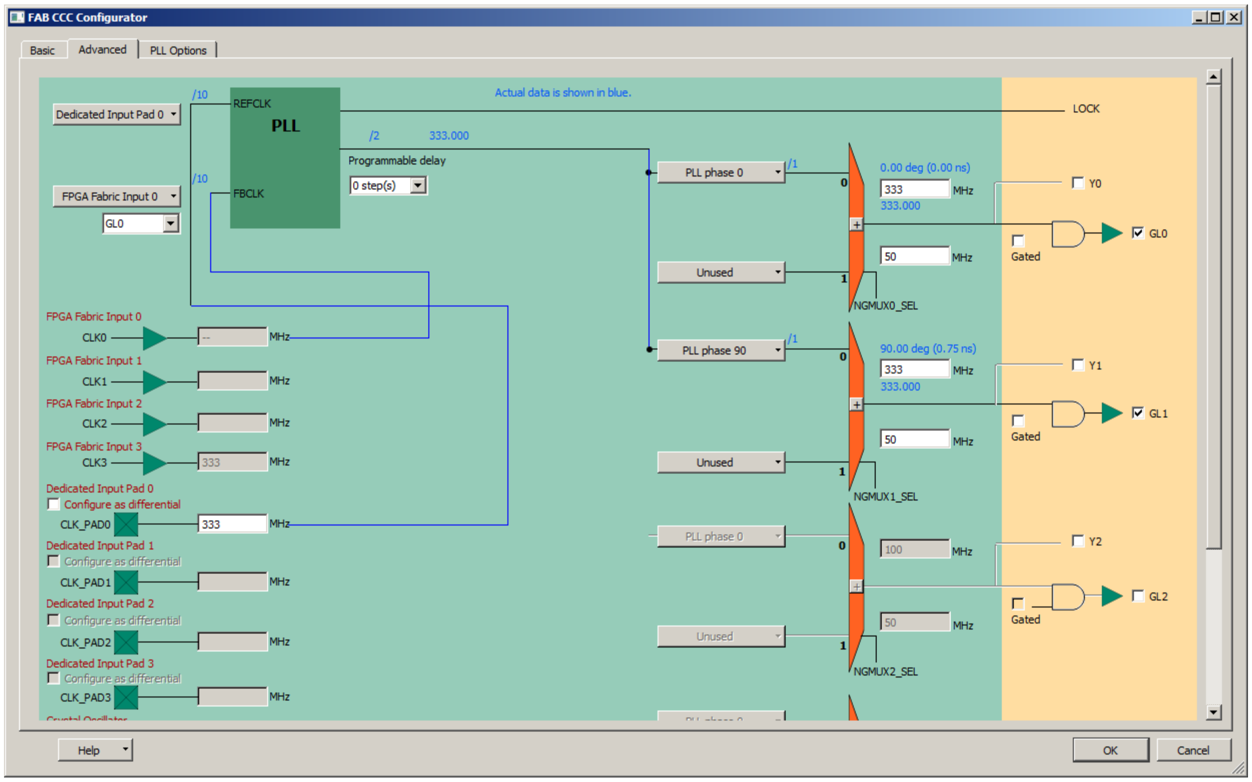


Figure 8 ·Connecting TX CCC to CoreQDR

Similarly, the receive or read path, CCC must be configured appropriately. In this case, you must configure the reference clock to correspond with the echo clock coming back from the QDR SRAM device. Microsemi® recommends using a dedicated (hardwired) CCC input pad for this clock. This allows for deterministic timing between the CCC and CLK PAD. These pins are documented in the pinout table as CCC_***_CLKlw pins and are specific to CCC locations.

Note: The REFCLK in Figure 9 is configured as a Dedicated CCC Input Pad. In addition, to mitigate the clock PAD delay and to ensure that read data and the read clock are aligned. You must use a 0-degree feedback clock. Figure 9 shows this as GL0. GL1, on the other hand, is the read capture clock, which is a 90-degree phase shifted clock and which should be connected to the CoreQDR PLL_RCLK input port.



Actual data is shown in blue.

REFCLK: Dedicated Input Pad 0, /10

FBCLK: FPGA Fabric Input 0, GL0, /10

/2 333.000

Programmable delay: 0 step(s)

PLL phase 0, /1: 0.00 deg (0.00 ns), 333 MHz, 333.000; Unused: 50 MHz; ☑ GL0

PLL phase 90, /1: 90.00 deg (0.75 ns), 333 MHz, 333.000; Unused: 50 MHz; ☑ GL1

PLL phase 0: 100 MHz; Unused: 50 MHz; ☐ GL2

FPGA Fabric Input 0: CLK0 -- MHz

FPGA Fabric Input 1: CLK1 MHz

FPGA Fabric Input 2: CLK2 MHz

FPGA Fabric Input 3: CLK3 333 MHz

Dedicated Input Pad 0: ☐ Configure as differential; CLK_PAD0 333 MHz

Dedicated Input Pad 1: ☐ Configure as differential; CLK_PAD1 MHz

Dedicated Input Pad 2: ☐ Configure as differential; CLK_PAD2 MHz

Dedicated Input Pad 3: ☐ Configure as differential; CLK_PAD3 MHz

Figure 9 ·RX PLL Configuration

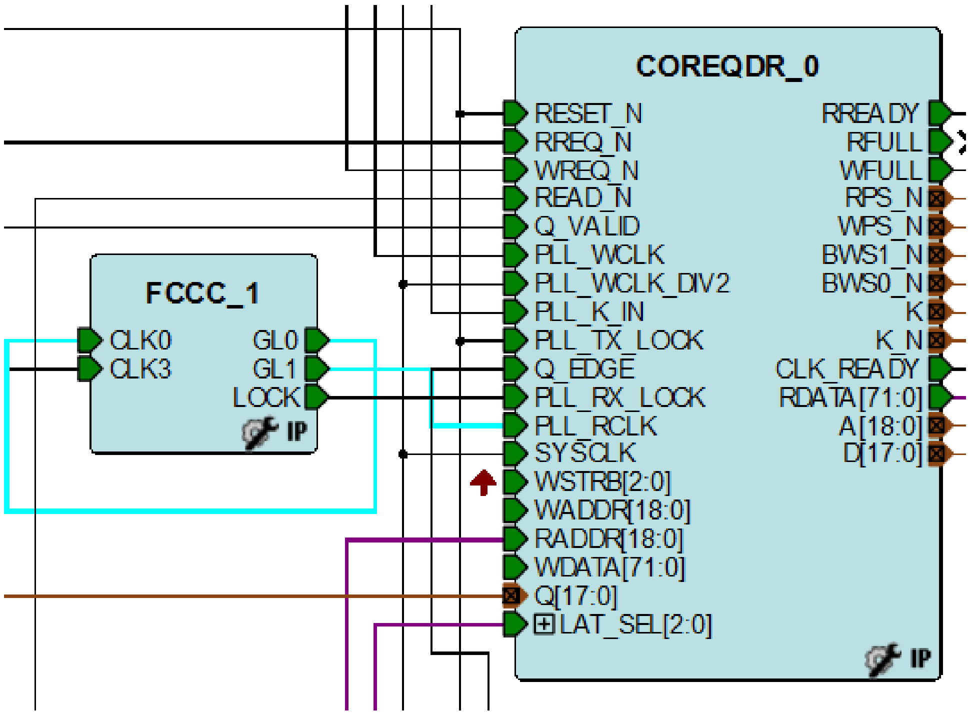


Figure 10 ·Connected RX CCC to CoreQDR

Constraints

As the CoreQDR is a high-speed controller, it is important to provide it with tight constraints in order to Place-and-Route a Libero SoC design without timing violations.

Pin Constraints

There are several pins in CoreQDR, WPS_N and RPS_N, A[*] and D[*] (write port select and read port select, address, and data, respectively) whose final stage is a register. To aid in timing, it is required that these ports be pushed to the IO registers. In the physical design constraint (PDC) file, this can be forced with a -register switch. For example:

```
# ENABLE IO REGISTER PACKING
set_io RPS_N \
-pinname E11 \
-fixed yes \
-iostd HSTLII \
-FF_IO_STATE TRISTATE \
-OUT_LOAD 5 \
-RES_PULL None \
-register yes \
-DIRECTION OUTPUT

# ENABLE IO REGISTER PACKING
set_io WPS_N \
-pinname D15 \
-fixed yes \
-iostd HSTLII \
-FF_IO_STATE TRISTATE \
-OUT_LOAD 5 \
-RES_PULL None \
-register yes \
-DIRECTION OUTPUT
set_io {A[*]} \
-pinname A14 \
-fixed yes \
-REGISTER Yes \
-OUT_REG Yes \
-DIRECTION OUTPUT
```

Note: To help with placement and routing, it is advised to assign buses contiguously. That is, for the address bus A, A[0] must be next to A[1], followed by A[2], and so on. The same principal applies to the data buses D and Q.

Timing Constraints

To create input and output delay constraints, refer to the datasheet for the particular QDR device with which they are interfacing.

Input Constraints

For input delays, create two sets of delays, a maximum and a minimum for the Q ports (with respect to CQ) for both rising and falling edges of the clock. This will yield a set of four constraints, which are derived from the memory’s output clock-to-Q and hold time (for max and min, respectively). An example of these for a particular device operating at 333 MHz is given below:

```
# Input delay (max) = trace delay + Tcqd = 0.257 + 0.250 (@333Mhz) = 0.507ns
# Input delay (min) = trace delay + Tcqdoh = 0.257 - 0.250 (@333Mhz) = 0.007ns

set max_input_delay 0.507
set min_input_delay 0.007

set_input_delay $max_input_delay \
-clock { CQ } \
-max [get_ports { Q }]

set_input_delay $min_input_delay \
-clock { CQ } \
-min [get_ports { Q }]

set_input_delay -clock_fall $max_input_delay \
-clock { CQ } \
-max [get_ports { Q }]

set_input_delay -clock_fall $min_input_delay \
-clock { CQ } \
-min [get_ports { Q }]
```

Output Constraints

The same principles apply to Output delays, with the exception that SmartTime must be aware that launch and capture are happening on the same edge. This is achieved by setting a max delay of 0 (setup) and one clock cycle (hold) min delay for control, data, and address signals. An example of output delay constraints is shown below.

```
################################################################
# OUTPUT DELAYS
# LAUNCH & CAPTURE ARE HAPPENING ON SAME EDGE
# Output delay (max) = trace delay + Tsa = 0.257 + 0.300 = 0.557ns
# Output delay (min) = trace delay + Tha = 0.257 - 0.300 = -0.043ns
################################################################

set max_output_delay 0.557
set min_output_delay -0.043
set clock_period 3.000
set half_period 1.500

# RPS_N, WPS_N_A are SDR
set_output_delay $max_output_delay \
-clock { K } \
-max [get_ports { RPS_N WPS_N }]

set_output_delay $min_output_delay \
-clock { K } \
-min [get_ports { RPS_N WPS_N }]

#### SETUP = 0, HOLD = -CLOCK PERIOD ####
set_max_delay 0\
-to [get_ports { RPS_N WPS_N }]
set_min_delay -$clock_period\
-to [get_ports { RPS_N WPS_N }]

# A, D are DDR
set_output_delay $max_output_delay \
-clock { K } \
-max { D A }

set_output_delay -clock_fall $max_output_delay \
-clock { K } \
-max { D A }

set_output_delay $min_output_delay \
-clock { K } \
-min { D A }

set_output_delay -clock_fall $min_output_delay \
-clock { K } \
-min { D A }

#### SETUP = 0, HOLD = -HALF PERIOD ####
set_max_delay 0\
-to [get_ports { D A }]

set_min_delay -$half_period\
-to [get_ports { D A }]
```

Synthesis Constraints

The input and output constraints defined above must also be included as constraints for Synthesis. The input and output constraints are shown below:

```
###==== INPUT / OUTPUT DELAYS
set_input_delay {p:Q[17:0]} -clock {c:CQ} {0.507}
set_input_delay {p:Q[17:0]} -clock {c:CQ} -clock_fall {0.507}

set_output_delay {p:RPS_N} -clock {c:K} {0.557}
set_output_delay {p:WPS_N} -clock {c:K} {0.557}

set_output_delay {p:A[18:0]} -clock {c:K} {0.557}
set_output_delay {p:A[18:0]} -clock {c:K} -clock_fall {0.557}

set_output_delay {p:D[17:0]} -clock {c:K} {0.557}
set_output_delay {p:D[17:0]} -clock {c:K} -clock_fall {0.557}
```

It is also important to ensure that all clocks are correctly defined in the Synplify SDC file. For example, using the clock configuration above, using a 50 MHz external input clock:

```
###==== CLOCKS
create_clock -name {CLK0_PAD} {p:CLK0_PAD} -period {20}
create_clock -name {CQ} {p:CQ} -period {3}

###==== GENERATED CLOCKS
create_generated_clock -name {FCCC_1_GL0} -source {p:CQ} {n:FCCC_1.GL0_net} -multiply_by
{1}
create_generated_clock -name {FCCC_1_GL1} -source {p:CQ} {n:FCCC_1.GL1_net} -multiply_by
{1}


create_generated_clock -name {FCCC_0_GL0} -source {p:CLK0_PAD} {n:FCCC_0.GL0_net} -
multiply_by {333} -divide_by {100}
create_generated_clock -name {FCCC_0_GL1} -source {p:CLK0_PAD} {n:FCCC_0.GL1_net} -
multiply_by {333} -divide_by {50}
create_generated_clock -name {FCCC_0_GL2} -source {p:CLK0_PAD} {n:FCCC_0.GL2_net} -
multiply_by {333} -divide_by {50}
```

Note: Depending on the version of Synplify being used, it may also be necessary to shorten net paths – which can be observed in the SmartTime timing analyzer – by ensuring that the fanout of certain signals is 1. This can be achieved using the **syn_maxfan** attribute. The following lines in the SDC file are greatly improved timing:

```
###==== ATTRIBUTES
define_attribute {i:COREQDR_0.genblk1\.waddr_select} {syn_maxfan} {1}
define_attribute {i:COREQDR_0.genblk1\.raddr_select} {syn_maxfan} {1}
```

Clock Source Latencies

As mentioned in CCC Configuration section, the read clock CCC contains a feedback loop to mitigate the clock generation and clock routing delay from CQ to the read clock. SmartTime must be made aware of this path by adding a clock latency in the timing constraints. This latency can be calculated as follows in SmartTime, and must be done for both the maximum and minimum case:

$$\text{Latency} = t_{\text{REFCLK to GL0}} + t_{\text{GL0 to CLK0}}$$

Where GL0 is the 0-degree CCC output, REFCLK is the dedicated input Pad, and CLK0 is the feedback input. This can be calculated by adding User Sets in SmartTime and observing the values reported. For more information on this flow, refer to the Libero SoC User Guide. As calculated this information, add a clock latency with the following:

```
#### RX PLL GL0 ####
set_clock_latency -source \
-early \
-rise \
$max_pll_clock_latency \
[get_clocks FCCC_1/CCC_INST/INST_CCC_IP:GL0]

set_clock_latency -source \
-early \
-fall \
$max_pll_clock_latency \
[get_clocks FCCC_1/CCC_INST/INST_CCC_IP:GL0]

set_clock_latency -source \
-late \
-rise \
$max_pll_clock_latency \
[get_clocks FCCC_1/CCC_INST/INST_CCC_IP:GL0]

set_clock_latency -source \
-late \
-fall \
$max_pll_clock_latency \
[get_clocks FCCC_1/CCC_INST/INST_CCC_IP:GL0]
```

Note: This path must be calculated and checked against the design for both minimum (hold time) and maximum (setup time).

False Paths and Special Cases

It is possible that SmartTime will report paths as having timing violations when those paths are invalid or not in use for particular configurations. In this case, it is advised to set those paths as false paths so that designer is able to ignore those paths and concentrate on placement relevant to meeting timing.

In particular, there are paths meant to be asynchronous in the read data FIFO that are false paths, and should be set as such if they are being reported as having timing violations in SmartTime, using the following SDC commands:

```
#### FALSE PATH FOR ASYNC FIFO PATHS ####
set_false_path -from {
COREQDR_0/genr_18.fifo_rdata/COREFIFO_0/genblk1.U_fifocore_async/genblk1.rptr[*]:CLK } \
-to {
COREQDR_0/genr_18.fifo_rdata/COREFIFO_0/genblk1.U_fifocore_async/Rd_doubleSync/sync_int[*
]:D }

set_false_path -from {
COREQDR_0/genr_18.fifo_rdata/COREFIFO_0/genblk1.U_fifocore_async/genblk1.wptr[*]:CLK } \
-to {
COREQDR_0/genr_18.fifo_rdata/COREFIFO_0/genblk1.U_fifocore_async/Wr_doubleSync/sync_int[*
]:D }
```

In addition, there may be cases where the user must over-constrain certain paths that should be theoretically solvable by placement – that is, if there is a significant Net Delay to Cell Delay ratio. This situation indicates more routing delay compared to logical delay through the cells. By setting a max_delay constraint the place and route can be forced to tighten the routing segments and improving the entire path timing. This can be done by setting a maximum to less than one clock period. For example, for a 333 MHz clock, the user might set a register-to-register max delay to 2.5 ns, as such:

```
### ADD MAX DELAY TO IMPROVE TIMING ####
set_max_delay 2.500\
 -from { COREQDR_0/genblk1.fifo_raddr_0[*]:CLK } \
 -to { COREQDR_0/genblk1.raddr_0[*]:D }

set_max_delay 2.500\
 -from { COREQDR_0/genblk1.fifo_waddr_0[*]:CLK } \
 -to { COREQDR_0/genblk1.waddr_reg[*]:D }
```

Post-layout Simulation and Clock Adjustment

Microsemi recommends connecting CoreQDR to a QDR SRAM device with a QVLD signal. If that is the case, the steps outlined in this section are not required, as there are no asynchronous clock domain crossings.

If CoreQDR is used without QVLD enabled – that is, if the USE_QVALID parameter is set to 0 – this complicates timing significantly. If QVLD is not enabled, CoreQDR attempts to predict when data will become valid on the read data bus (Q) by counting clock cycles, selecting a fixed edge using the LAT_SEL and Q_EDGE configuration ports. It then asserts a **read_data_valid** signal internally, which gets passed to the SHIM layer of the core and is used to register the data from the DDR block. The data is then written into the read data FIFO.

The challenge with this approach is that there is a clock domain crossing between the read clock and the write clock that is not necessarily deterministic. It is entirely dependent on the propagation of the output K clock from the I/O pad on the board, to the QDR SRAM device, through the internal SRAM PLL, back out through the echo clock of the SRAM, and back into the IO pad of the SmartFusion2/IGLOO2 device through the CQ input, feeding into a second CCC (refer to CCC Configuration section).

As this phase relationship is *variable* depending on chip layout and board layout and I/O placement, ensure that, at the register(s) at which the clock crossing occurs, there is sufficient phase shift (nominally 180-degrees) between the clocks to not violate either setup or hold time in either of the clock domains. In the case of CoreQDR, this register is **rd_l2_en**, clocked on the read data path but driven by signals from the write data path. A safe constraint to set (for 333 MHz operation) is 1 ns for both minimum and maximum delays, as follows:

```
# NOT REQUIRED FOR QDR VERSION WITH VALID CONTROL
set_min_delay -1.000\
 -from { COREQDR_0/genblk1.RPS_N_*:CLK } \
 -to { COREQDR_0/shim_0/rd_l2_en:D }

set_max_delay 1.000\
 -from { COREQDR_0/genblk1.RPS_N_*:CLK } \
 -to { COREQDR_0/shim_0/rd_l2_en:D }
```

Note that, while SmartTime will *report* if there is a setup or hold time violation on this path, Libero cannot adjust the phase relationship between the two clocks. To do this, you must run **post-layout simulation** and observe the relationship between the SLE CLK input of RPS_N_* and the SLE CLK input of rd_l2_en. They should be as close to 180-degrees as possible. If they are not, it can be adjusted by adding input delay to **both** Q and CQ signals by the same amount (this ensures that the 90-degree relationship is maintained between read clock and read data).

Note that to accurately model the WCLK to RCLK relationship, a latency constraint is required from K to CQ that is an estimation of the loopback path from the K clock (with generation delay) back to the CQ clock, as follows:

```
##########################################################################################
#############################################
#
# ADD CLOCK SOURCE LATENCY FOR CQ
# CQ LATENCY = (FCCC_0:CLK0_PAD --> K (clock generation from pad)) + (Trace Delay K to
QDR chip) + (K to CQ delay inside QDR chip (Tccqo, Tcqoh)) + (Trace delay QDR chip to CQ)
#
# CALCULATED AT 200MHZ
# CQ LATENCY(max) = 10.548 + 0.257 + 0.450 + 0.257 = 11.842ns [Simplified to 11.512 -
5.000 = 6.512ns]
# CQ LATENCY(min) = 6.291 + 0.257 - 0.450 + 0.257 = 6.355ns [Simplified to 6.355 - 5.000
= 1.355ns]
#
```

```
##########################################################################################
#############################################
set max_clock_latency 6.512
#set min_clock_latency 1.355

#### CQ ####
set_clock_latency -source \
-early \
-rise \
$max_clock_latency \
[get_clocks CQ]

set_clock_latency -source \
-early \
-fall \
$max_clock_latency \
[get_clocks CQ]

set_clock_latency -source \
-late \
-rise \
$max_clock_latency \
[get_clocks CQ]

set_clock_latency -source \
-late \
-fall \
$max_clock_latency \
[get_clocks CQ]
```

Note: This path too must be calculated for both minimum delay (hold requirement) and maximum delay (setup requirement).

List of Changes

The following table shows important changes made in this document for each revision.

Date	Change
March 2019	Updated changes related to CoreQDR v3.3.
March 2017	License Lock feature included.
April 2015	Added support for RTG4.
October 2014	Added a note to Pin Constraints section.
	Updated User Interface Timing section.
April 2014	Updated Figure 6.
February 2014	Added Implementation Hints section.
December 2013	CoreQDR v3.0 release.
August 2013	CoreQDR v2.1 release.
March 2013	CoreQDR v2.0 release.

Microsemi Headquarters
One Enterprise, Aliso Viejo, CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
email: sales.support@microsemi.com
www.microsemi.com

50200431-7/3.19