Virtex-4 FPGA User Guide

UG070 (v2.6) December 1, 2008

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/02/04	1.0	Initial Xilinx release. Printed Handbook version.
09/10/04	1.1	In Chapter 1, "Clock Resources": Removed Table 1-6: "BUFGMUX_VIRTEX4 Attributes". Updated Table 1-1, Table 1-2, Table 1-5, the new Table 1-6. Revised Figure 1-2, Figure 1-5, Figure 1-6, Figure 1-7, Figure 1-9, Figure 1-10, Figure 1-13, Figure 1-14, and Figure 1-16. Associated text around these tables and figures were revised. In Chapter 2, "Digital Clock Managers (DCMs)", changes to "FACTORY_JF Attribute" and in Table 2-6. In Chapter 9, "System Monitor": Changed in Figure 9-4, Figure 9-5, Figure 9-7, Figure 9-8, Figure 9-9, Figure 9-10, Figure 9-21, Figure 9-25, Figure 9-26, and Figure 9-27. Changes to the equation in the Temperature Sensor section. The following tables had changes: Table 9-3, Table 9-5, Table 9-6, Table 9-9, Table 9-11, Table 9-12, Table 9-14, and Table 9-15. Changes to the entire System Monitor Calibration, System Monitor VHDL and Verilog Design Example sections.
02/01/05	1.2	In Chapter 1, "Clock Resources", revised "Global Clock Buffers", "Clock Regions", and "Clock Capable I/O" sections. In Chapter 4, "Block RAM," revised "Reset," page 151 description and Table 4-13. In Chapter 6, "SelectIO Resources," removed the device configuration section. The *Virtex-4 Configuration Guide* describes this information in detail. Edited "SSTL (Stub-Series Terminated Logic)," page 281. Replaced LVDS_25_DCI with LVDCI_25 in "Compatible example:," page 302. Added rule "7" to "DCI in Virtex-4 FPGA Hardware," page 241. Added "Simultaneous Switching Output Limits," page 306. Removed Chapter 9: System Monitor.

Date	Version	Revision
04/11/05	1.3	Chapter 1: Revised Table 1-1, page 26, Figure 1-14, and "BUFR Attributes and Modes" section including Figure 1-21, page 43. Chapter 2: Revised FACTORY_JF value in Table 2-6, page 65. Added "Phase-Shift Overflow" section. Clarified global clock discussion in "Global Clock Buffers", "Clock Regions", and "Clock Capable I/O". Chapter 4: Added "Built-in Block RAM Error Correction Code" section. Revised Figure 4-6 and Figure 4-8, page 123. Chapter 5: Revised Table 5-1 and Table 5-2, page 184. Chapter 6: Revised Table 6-29, page 290. Chapter 7: Revised "REFCLK - Reference Clock" and added Table 7-10, page 326. Chapter 8: Added "ISERDES Latencies," page 379 and "OSERDES Latencies," page 394. Revised "Guidelines for Using the Bitslip Submodule" section.
09/12/05	1.4	Chapter 2: Revised FACTORY_JF value in Table 2-6, page 65. The LOCKED signal description is updated in Figure 2-20 and Figure 2-21. Chapter 6: Revised the "Simultaneous Switching Output Limits" section. Chapter 8: Added more information to "Clock Enable Inputs – CE1 and CE2," page 369.
03/21/06	1.5	Chapter 1: Updated description under Table 1-1. Updated Figure 1-21, page 43. Chapter 4: Changed Table 4-8, page 144 and added a note. Updated the discussions in NO_CHANGE Mode and Cascadable Block RAM sections. Removed synchronous FIFO application example. Chapter 5: Revised slice label in Figure 5-30, page 224. Chapter 6: Added to the "Xilinx DCI" section. Added IBUF to the "PULLUP/PULLDOWN/KEEPER for IBUF, OBUFT, and IOBUF" discussion. Added V_{CCO} numbers in the +1.5V column in Table 6-5, page 258. Corrected Figure 6-70, page 292. Added notes 4 and 5 to Table 6-38, page 299. Updated 3.3V I/O Design Guidelines "Summary," page 306. Added "HSLVDCI (High-Speed Low Voltage Digitally Controlled Impedance)," page 259 section. Added 1.2V to Table 6-40, page 308, and added link to SSO calculator to text above table. Added HSLVDCI to Table 6-42, page 310. Revised Virtex-4 (SX Family) FF668 in Table 6-43. Chapter 8: Revised "Clock Enable Inputs – CE1 and CE2". Chapter 9, "Temperature Sensing Diode": Added the Virtex-4 temperature-sensing diode.
10/06/06	1.6	Chapter 7, "SelectIO Logic Resources": Modified text in section "REFCLK - Reference Clock" and deleted former Table 7-10.

Date	Version	Revision
01/04/07	2.0	• Chapter 1, "Clock Resources": ♦ "I/O Clock Buffer - BUFIO": Added "in the same region" to BUFIO ability to drive BUFRs. ♦ "BUFG VHDL and Verilog Templates": Corrected typo in VHDL template. ♦ "Regional Clocks and I/O Clocks": Added reference to the PACE tool for identifying clock regions. • Chapter 2, "Digital Clock Managers (DCMs)": ♦ "Status Flags": Corrected descriptions for Clock Events 2, 3, and 4. ♦ "Input Clock Requirements": Clarified when DCM output clocks are deskewed. ♦ "Reset Input — RST": Updated RST hold time to 200 ms after clock stabilization. ♦ "Frequency Synthesizer Characteristics": Added reference and link to a macro for monitoring LOCKED. • Chapter 4, "Block RAM": ♦ "Data Flow": Added paragraph clarifying ADDR setup/hold requirements. ♦ Table 4-11: Corrected typo to ALMOST FULL. ♦ "RAMB16 Port Mapping Design Rules": Corrected logic level tie for unused ADDR[A|B] pins to High. ♦ "Synchronous Clocking": Clarified synchronous write/read timing. ♦ Deleted `SIM_COLLISION_CHECK` statements from all templates. • Chapter 6, "SelectIO Resources": ♦ Figure 6-53: Corrected internal termination resistor designation. ♦ Table 6-1: Updated LVTTL DC voltage specifications. ♦ Table 6-31 and following: Globally corrected OBUFGDS to OBUFTDS. ♦ "Differential Termination Attribute": Corrected paragraph describing use of DIFF_TERM attribute. ♦ "Xilinx DCI": Added reference to section "Driver with Termination to VCCO / 2 (Split Termination)." ♦ Figure 6-64: Corrected I/O standard name to DIFF_SSTL2_II. ♦ Table 6-38: Corrected I/O standard name to DIFF_HSTL_II_18_DCI. • Chapter 7, "SelectIO Logic Resources": ♦ "IDELAYCTRL Locations": Reworded description of IDELAYCTRL locations in clock regions. ♦ Table 7-6: Added "when in Variable mode" to function descriptions of C, INC, and CE ports. ♦ Table 7-9: Added Note (1) to $T_{IDELAYRESOLUTION}$. ♦ Added requirement to wait 8 clock cycles after increment or decrement before sampling IDELAY. ♦ Figure 7-12: Modified to show 8 clock cycle wait time. ♦ Modified timing description to match new Figure 7-12. ♦ "IDELAY VHDL and Verilog Instantiation Template": Changed port map for C, CE, INC, and RST from open to zero (both Verilog and VHDL). ♦ Deleted `synthesis translate_off/synthesis translate_on` statements from all IDELAY instantiation templates.

Date	Version	Revision
01/04/07 *(cont'd)*	2.0 *(cont'd)*	• Chapter 8, "Advanced SelectIO Logic Resources": ♦ Table 8-1: REV: Added instruction to connect to GND. ♦ Table 8-2: Corrected BITSLIP_ENABLE value from "String" to "Boolean". ♦ "Registered Outputs – Q1 to Q6": Added clarification on bit in/out sequence. ♦ "High-Speed Clock for Strobe-Based Memory Interfaces – OCLK": Added instruction to ground OCLK when INTERFACE_TYPE is NETWORKING. ♦ "BITSLIP_ENABLE Attribute": Specified setting according to setting of INTERFACE_TYPE. ♦ "INTERFACE_TYPE Attribute": Added recommendation to use MIG when ISERDES is in Memory Mode. Added Figure 8-6 to illustrate ISERDES internal connections in Memory Mode. ♦ Added section "ISERDES Clocking Methods." ♦ "ISERDES Width Expansion": Added explanatory paragraph regarding master/slave ISERDES use with differential/single-ended inputs. ♦ "Guidelines for Expanding the Serial-to-Parallel Converter Bit Width": Corrected a number of master/slave and input/output reversals. ♦ "Verilog Instantiation Template to use Width Expansion Feature": Corrected a number of errors in the template. ♦ "ISERDES Latencies": Deleted former Table 8-4 and most of the text in this section and replaced with statement relating latency to INTERFACE_TYPE. ♦ Deleted `synthesis translate_off/synthesis translate_on` statements from all ISERDES instantiation templates. ♦ "Data Parallel-to-Serial Converter": Added recommendation to apply a reset to OSERDES prior to use. ♦ "OSERDES Width Expansion": Added explanatory paragraph regarding master/slave OSERDES use with differential/single-ended outputs. ♦ "OSERDES VHDL Template" in Chapter 8: Removed erroneous semicolon following TRISTATE_WIDTH.
03/15/07	2.1	• "ILOGIC Resources": Added sentence clarifying SR and REV sharing between ILOGIC/ISERDES and OLOGIC/OSERDES. • Figure 7-1: Removed OFB/TFB inputs and associated MUXes. • Figure 8-2: Removed OFB/TFB inputs. • "DIFF_SSTL2_II_DCI, DIFF_SSTL18_II_DCI Usage": Removed incorrect bidirectional link requirements and reference to on-chip *differential* termination. • "DCI in Virtex-4 FPGA Hardware": Modified point 3 detailing when VRP/VRN reference resistors are not required. • "PULLUP/PULLDOWN/KEEPER for IBUF, OBUFT, and IOBUF": Added a paragraph recommending against using these circuits to drive a logic level on a board-level trace. • "Frequency Synthesizer Characteristics": Updated information regarding the setting of AUTOCALIBRATE and CONFIG STEPPING. • Added new section "FIFO16 Error Condition and Work-Arounds" in Chapter 4, including VHDL/Verilog source files in `UG070.zip`. • Table 6-41: Added SSO data for FF676 device/package combinations.

Date	Version	Revision
04/10/07	2.2	• Added section "Cascading DCMs" in Chapter 2. • Table 7-9: Deleted Note (1). • Figure 7-12: Added assumption that IOBDELAY_VALUE = 0 to text. • Section "IDELAY Timing": Revised descriptions of Clock Events 1, 2, and 3 in Figure 7-12. • Added new section "Note on Instability after an Increment/Decrement Operation". • Table 7-12: Revised description of CE port. • Chapter 8, "Advanced SelectIO Logic Resources": ISERDES and OSERDES sections extensively revised and expanded with many new figures and tables.
08/10/07	2.3	• Figure 2-5 and associated text: Updated. • Figure 2-20: Corrected reset requirement from 3 periods to 200 ns. • Figure 2-22, associated text: Corrected number of clock cycles in Clock Event 4. • "Frequency Synthesizer Characteristics" in Chapter 2: Added note to indicate no need for the LOCKED monitoring macro on recent step devices. • "SelectIO Resources Introduction" in Chapter 6: Added note that differential and V_{REF}-dependent inputs are powered by V_{CCAUX}. • "DCI in Virtex-4 FPGA Hardware" in Chapter 6: Removed erroneous reference to SSTL3 standard. • "Lower Capacitance I/O Attributes" in Chapter 6: Added RSDS_25 to list of standards that do not have differential driver circuits. • Added Note (1) to Table 6-40. • Table 6-43: Included FX family devices and added note (3) for Banks 9 and 10. • "Temperature Sensor Examples" in Chapter 9: Added information on Texas Instruments temperature sensor.
04/10/08	2.4	• Table 2-6, page 65: Added CLK_FEEDBACK and DCM_AUTOCALIBRATION attribute rows. Added descriptions to CLKFX_DIVIDE and CLKFX_MULTIPLY rows. • "DCM_AUTOCALIBRATION Attribute," page 68: New section. • Figure 2-9, page 84 and Figure 2-11, page 85: Removed element from Q output. • Under Figure 3-5, page 104: Clarified bullet regarding RST must be Low before REL has an effect. • Figure 4-11, page 142: Removed REGCEN. • Table 6-40, page 308: Added LVCMOS15_16_fast, LVDCI_DV2_18, and LVTTL24_fast. • "REFCLK - Reference Clock," page 342: Changed IDELAYCTRL_REF_PRECISION units to MHz. • Figure 7-21, page 355: Corrected OFFDDRB labeling.
06/17/08	2.5	• Figure 2-4, page 73: Revised the contents of the DCM block. • "System-Synchronous Setting (Default)," page 73: Added text to the end of the section describing cases when the DESKEW_ADJUST parameter has no effect.
12/01/08	2.6	• "Asynchronous Clocking," page 119: Added the results of performing a read and write operation. • Figure 6-6, page 238: Moved VREF to be inside the FPGA. • "DCI in Virtex-4 FPGA Hardware," page 241: Added SSTL18_I_DCI to the list of DCI outputs that do not require reference resistors on VRP/VRN. • Figure 7-10, page 330: Updated figure title.

Table of Contents

Chapter 2: Digital Clock Managers (DCMs)

Chapter 3: Phase-Matched Clock Dividers (PMCDs)

Chapter 4: Block RAM

Chapter 5: Configurable Logic Blocks (CLBs)

Chapter 6: SelectIO Resources

Chapter 7: SelectIO Logic Resources

Chapter 8: Advanced SelectIO Logic Resources

Chapter 9: Temperature Sensing Diode

About This Guide

This document describes the Virtex®-4 FPGA architecture. Complete and up-to-date documentation of the Virtex-4 family of FPGAs is available on the Xilinx® website at http://www.xilinx.com/virtex4.

Guide Contents

Additional Documentation

The following documents are also available for download at http://www.xilinx.com/virtex4.

- DS112, *Virtex-4 Family Overview*

  The features and product selection of the Virtex-4 family are outlined in this overview.

- DS302, *Virtex-4 Data Sheet: DC and Switching Characteristics*

  This data sheet contains the DC and Switching Characteristic specifications for the Virtex-4 family.

- UG073, *XtremeDSP for Virtex-4 FPGAs User Guide*

  This guide describes the XtremeDSP™ slice and includes reference designs for using DSP48 math functions and various FIR filters.

- UG071, *Virtex-4 Configuration Guide*

  This all-encompassing configuration guide includes chapters on configuration interfaces (serial and SelectMAP), bitstream encryption, Boundary-Scan and JTAG configuration, reconfiguration techniques, and readback through the SelectMAP and JTAG interfaces.

- UG072, *Virtex-4 PCB Designer's Guide*

  This guide describes PCB guidelines for the Virtex-4 family. It covers SelectIO™ signaling, RocketIO™ signaling, power distribution systems, PCB breakout, and parts placement.

- UG075, *Virtex-4 Packaging and Pinout Specification*

  This specification includes the tables for device/package combinations and maximum I/Os, pin definitions, pinout tables, pinout diagrams, mechanical drawings, and thermal specifications.

- UG076, *Virtex-4 RocketIO Multi-Gigabit Transceiver User Guide*

  This guide describes the RocketIO Multi-Gigabit Transceivers available in the Virtex-4 FX family.

- UG074, *Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC User Guide*

  This guide describes the Tri-mode Ethernet Media Access Controller available in the Virtex-4 FX family.

- UG018, *PowerPC 405 Processor Block Reference Guide*

  This guide describes the IBM PowerPC® 405 processor block available in the Virtex-4 FX family.

Additional Support Resources

To search the database of silicon and software questions and answers, or to create a technical support case in WebCase, see the Xilinx website at: http://www.xilinx.com/support.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**

Convention	Meaning or Use	Example
Italic font	Variables in a syntax statement for which you must supply values	**ngdbuild** *design_name*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **bus[7:0]**, they are required.	**ngdbuild** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**lowpwr =**{**on**|**off**}
Vertical bar |	Separates items in a list of choices	**lowpwr =**{**on**|**off**}
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Repetitive material that has been omitted	**allow block** *block_name loc1 loc2 ... locn;*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 5 in the *Virtex-II Platform FPGA User Guide.*
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Chapter 1

Clock Resources

Global and Regional Clocks

For clocking purposes, each Virtex®-4 device is divided into regions. The number of regions varies with device size, eight regions in the smallest device to 24 regions in the largest one.

Global Clocks

Each Virtex-4 device has 32 matched-skew global clock lines that can clock all sequential resources on the whole device (CLB, block RAM, DCMs, and I/O), and also drive logic signals. Any eight of these 32 global clock lines can be used in any region. Global clock lines are only driven by a global clock buffer, and can also be used as a clock enable circuit or a glitch-free multiplexer. It can select between two clock sources, and can also switch away from a failed clock source, a new feature in the Virtex-4 architecture.

A global clock buffer is often driven by a Digital Clock Manager (DCM) to eliminate the clock distribution delay, or to adjust its delay relative to another clock. There are more global clocks than DCMs, but a DCM often drives more than one global clock.

Regional Clocks and I/O Clocks

Each region has two "clock capable" regional clock inputs. Each input can differentially or single-endedly drive regional clocks and I/O clocks in the same region, and also in the region above or below (i.e., in up to three adjacent regions).

The regional clock buffer can be programmed to divide the incoming clock rate by any integer number from 1 to 8. This feature, in conjunction with the programmable serializer/deserializer in the IOB (see Chapter 8, "Advanced SelectIO Logic Resources") allows source-synchronous systems to cross clock domains without using additional logic resources.

A third type of clocking resource, I/O clocks, are very fast and serve localized I/O serializer/deserializer circuits (see Chapter 8, "Advanced SelectIO Logic Resources").

For more detail on how to identify clock regions and the associated components, please use the PACE tool.

Global Clocking Resources

Global clocks are a dedicated network of interconnect specifically designed to reach all clock inputs to the various resources in an FPGA. These networks are designed to have low skew and low duty cycle distortion, low power, and increased jitter tolerance. They are also designed to support very high frequency signals.

Understanding the signal path for a global clock expands the understanding of the various global clock resources. The global clocking resources and network consist of the following paths and components:

- Global Clock Inputs
- Global Clock Buffers
- Clock Tree and Nets - GCLK
- Clock Regions

Global Clock Inputs

Virtex-4 FPGAs contain specialized global clock input locations for use as regular user I/Os if not used as clock inputs. The number of clock inputs varies with the device size. Smaller devices contain 16 clock inputs, while larger devices have 32 clock inputs. Table 1-1 summarizes the number of clock inputs available for different Virtex-4 devices.

Table 1-1: **Number of Clock I/O Inputs by Device**

Device	Number of Clock I/O Inputs
XC4VLX15, XC4VLX25 XC4VSX25, XC4VSX35 XC4VFX12, XC4VFX20, XC4VFX40, XC4VFX60	16
XC4VLX40(1), XC4VLX60(1), XC4VLX80, XC4VLX100, XC4VLX160, XC4VLX200 XC4VSX55 XC4VFX100(2), XC4VFX140	32

Notes:

1. The XC4VLX40 and XC4VLX60 in the FF668 package only have 16 clock input pins.
2. The XC4VFX100 in the FF1152 package only has 16 clock input pins.

Clock inputs can be configured for any I/O standard, including differential I/O standards. Each clock input can be either single-ended or differential. All 16 or 32 clock inputs can be differential if desired. When used as outputs, global clock input pins can be configured for any output standard except LVDS and HT output differential standards. Each global clock input pin supports any single-ended output standard or any CSE output differential standard.

Global Clock Input Buffer Primitives

The primitives in Table 1-2 are different configurations of the input clock I/O input buffer.

Table 1-2: **Clock Buffer Primitives**

Primitive	Input	Output	Description
IBUFG	I	O	Input clock buffer for single-ended I/O
IBUFGDS	I, IB	O	Input clock buffer for differential I/O

These two primitives work in conjunction with the Virtex-4 FPGA I/O resource by setting the IOSTANDARD attribute to the desired standard. Refer to Chapter 6, "I/O Compatibility" Table 6-38 for a complete list of possible I/O standards.

Power Savings by Disabling Global Clock Buffer

The Virtex-4 FPGA clock architecture provides a straightforward means of implementing clock gating for the purposes of powering down portions of a design.

Most designs contain several unused BUFGMUX resources. A clock can drive multiple BUFGMUX inputs, and the BUFGMUX outputs, which will be synchronous with each other, can be used to drive distinct regions of logic. For example, if all the logic required to be always operating can be constrained to a few clocking regions, then one of the BUFGMUX outputs can be used to drive those regions. Toggling the enable of the other BUFGMUX then provides a simple means of stopping all dynamic power consumption in those regions of logic available for power savings.

The XPower tool can be used to estimate the power savings from such an approach. The difference can be calculated either by toggling the BUFGMUX enable or by setting the frequency on the corresponding clock net to 0 MHz.

Global Clock Buffers

There are 32 global clock buffers in every Virtex-4 device. Each half of the die (top/bottom) contains 16 global clock buffers. A global clock input can directly connect from the P-side of the differential input pin pair to any global clock buffer input in the same half, either top or bottom, of the device. Each differential global clock pin pair can connect to either a differential or single-ended clock on the PCB. If using a single-ended clock, then the P-side of the pin pair must be used because a direct connection only exists on this pin. For pin naming conventions, refer to the *Virtex-4 Packaging and Pinout Specification*. A single-ended clock connected to the N-side of the differential pair results in a local route and creates additional delay. If a single-ended clock is connected to a differential pin pair then the other side (N-side typically) can not be used as another single-ended clock pin. However, it can be used as a user I/O. A device with 16 global clock pins can be connected to 16 differential or 16 single-ended board clocks. A device with 32 global clock pins can be connected to 32 clocks under these same conditions.

Global clock buffers allow various clock/signal sources to access the global clock trees and nets. The possible sources for input to the global clock buffers include:

- Global clock inputs
- Digital Clock Manager (DCM) outputs
- Phase-Matched Clock Divider (PMCD) outputs
- Rocket IO Multi-Gigabit Transceivers
- Other global clock buffer outputs
- General interconnect

The global clock buffers can only be driven by sources in the same half of the die (top/bottom).

All global clock buffers can drive all clock regions in Virtex-4 devices. The primary/secondary rules from Virtex-II and Virtex-II Pro FPGAs do not apply. However, only eight different clocks can be driven in a single clock region. A clock region (16 CLBs) is a branch of the clock tree consisting of eight CLB rows up and eight CLB rows down. A clock region only spans halfway across the device.

The clock buffers are designed to be configured as a synchronous or asynchronous “glitch free” 2:1 multiplexer with two clock inputs. Virtex-4 devices have more control pins to provide a wider range of functionality and more robust input switching. The following

subsections detail the various configurations, primitives, and use models of the Virtex-4 FPGA clock buffers.

Global Clock Buffer Primitives

The primitives in Table 1-3 are different configurations of the global clock buffers.

Table 1-3: **Global Clock Buffer Primitives**

Primitive	Input	Output	Control
BUFGCTRL	I0, I1	O	CE0, CE1, IGNORE0, IGNORE1, S0, S1
BUFG	I	O	–
BUFGCE	I	O	CE
BUFGCE_1	I	O	CE
BUFGMUX	I0, I1	O	S
BUFGMUX_1	I0, I1	O	S
BUFGMUX_VIRTEX4	I0, I1	O	S

Notes:

1. All primitives are derived from a software preset of BUFGCTRL.

BUFGCTRL

The BUFGCTRL primitive shown in Figure 1-1, can switch between two asynchronous clocks. All other global clock buffer primitives are derived from certain configurations of BUFGCTRL. The ISE® software tools manage the configuration of all these primitives.

BUFGCTRL has four select lines, S0, S1, CE0, and CE1. It also has two additional control lines, IGNORE0 and IGNORE1. These six control lines are used to control the input I0 and I1.

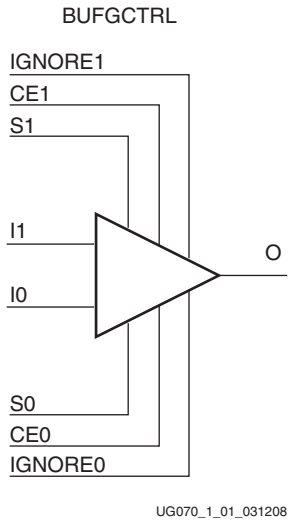


Figure 1-1: **BUFGCTRL Primitive**

BUFGCTRL is designed to switch between two clock inputs without the possibility of a glitch. When the presently selected clock transitions from High to Low after S0 and S1

change, the output is kept Low until the other ("to-be-selected") clock has transitioned from High to Low. Then the new clock starts driving the output.The default configuration for BUFGCTRL is falling edge sensitive and held at Low prior to the input switching. BUFGCTRL can also be rising edge sensitive and held at High prior to the input switching.

In some applications the conditions previously described are not desirable. Asserting the IGNORE pins bypasses the BUFGCTRL from detecting the conditions for switching between two clock inputs. In other words, asserting IGNORE causes the mux to switch the inputs at the instant the select pin changes. IGNORE0 causes the output to switch away from the I0 input immediately when the select pin changes, while IGNORE1 causes the output to switch away from the I1 input immediately when the select pin changes.

Selection of an input clock requires a "select" pair (S0 and CE0, or S1 and CE1) to be asserted High. If either S or CE is not asserted High, the desired input is not selected. In normal operation, both S and CE pairs (all four select lines) are not expected to be asserted High simultaneously. Typically only one pin of a "select" pair is used as a select line, while the other pin is tied High. The truth table is shown in Table 1-4.

Table 1-4: **Truth Table for Clock Resources**

CE0	S0	CE1	S1	O
1	1	0	X	I0
1	1	X	0	I0
0	X	1	1	I1
X	0	1	1	I1
1	1	1	1	Old Input (1)

Notes:

1. Old input refers to the valid input clock before this state is achieved.
2. For all other states, the output becomes the value of INIT_OUT and does not toggle.

Although both S and CE are used to select a desired output, each one of these pins behaves slightly different. When using CE to switch clocks, the change in clock selection can be faster than when using S. Violation in setup/hold times of the CE pins causes a glitch at the clock output. On the other hand, using the S pins allows the user to switch between the two clock inputs without regard to setup/hold times. It does not result in a glitch. See the discussion of "BUFGMUX_VIRTEX4". The CE pin is designed to allow backward compatibility from Virtex-II and Virtex-II Pro FPGAs.

The timing diagram in Figure 1-2 illustrates various clock switching conditions using the BUFGCTRL primitives. Exact timing numbers are best found using the speed specification.

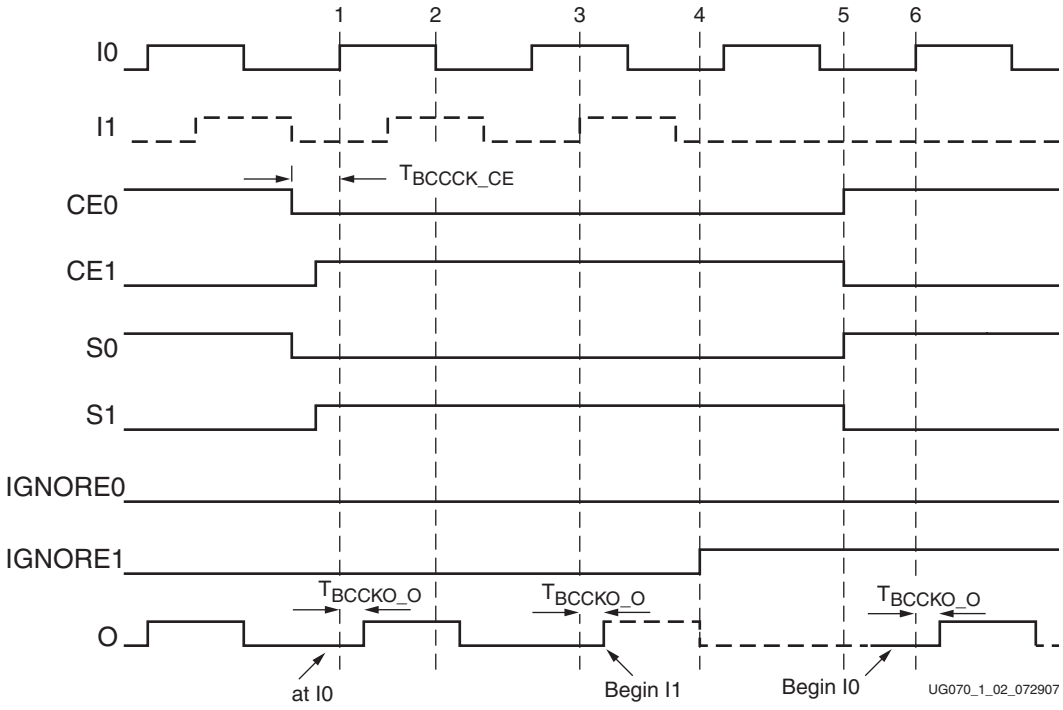


Figure 1-2: **BUFGCTRL Timing Diagram**

- Before time event 1, output O uses input I0.
- At time T_{BCCCK_CE}, before the rising edge at time event 1, both CE0 and S0 are deasserted Low. At about the same time, both CE1 and S1 are asserted High.
- At time T_{BCCKO_O}, after time event 3, output O uses input I1. This occurs after a High to Low transition of I0 (event 2) followed by a High to Low transition of I1.
- At time event 4, IGNORE1 is asserted.
- At time event 5, CE0 and S0 are asserted High while CE1 and S1 are deasserted Low. At T_{BCCKO_O}, after time event 6, output O has switched from I1 to I0 without requiring a High to Low transition of I1.

Other capabilities of BUFGCTRL are:

- Pre-selection of the I0 and I1 inputs are made after configuration but before device operation.
- The initial output after configuration can be selected as either High or Low.
- Clock selection using CE0 and CE1 only (S0 and S1 tied High) can change the clock selection without waiting for a High to Low transition on the previously selected clock.

Table 1-5 summarizes the attributes for the BUFGCTRL primitive.

Table 1-5: **BUFGCTRL Attributes**

Attribute Name	Description	Possible Values
INIT_OUT	Initializes the BUFGCTRL output to the specified value after configuration. Sets the positive or negative edge behavior. Sets the output level when changing clock selection.	0 (default), 1
PRESELECT_I0	If TRUE, the BUFGCTRL output uses the I0 input after configuration(1).	FALSE (default), TRUE
PRESELECT_I1	If TRUE, the BUFGCTRL output uses the I1 input after configuration(1).	FALSE (default), TRUE

Notes:

1. Both PRESELECT attributes cannot be TRUE at the same time.
2. The LOC constraint is available.

BUFG

BUFG is simply a clock buffer with one clock input and one clock output. This primitive is based on BUFGCTRL with some pins connected to logic High or Low. Figure 1-3 illustrates the relationship of BUFG and BUFGCTRL. A LOC constraint is available for BUFG.

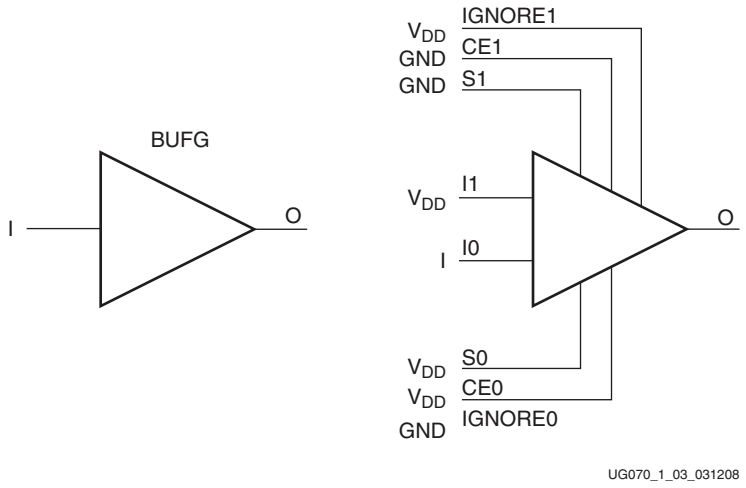


Figure 1-3: **BUFG as BUFGCTRL**

The output follows the input as shown in the timing diagram in Figure 1-4.

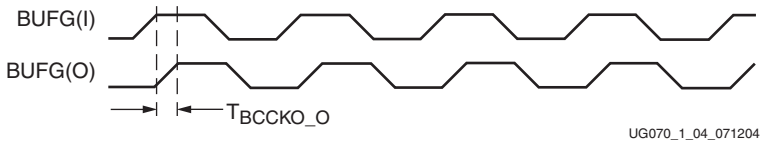


Figure 1-4: **BUFG Timing Diagram**

BUFGCE and BUFGCE_1

Unlike BUFG, BUFGCE is a clock buffer with one clock input, one clock output and a clock enable line. This primitive is based on BUFGCTRL with some pins connected to logic High

or Low. Figure 1-5 illustrates the relationship of BUFGCE and BUFGCTRL. A LOC constraint is available for BUFGCE and BUFGCE_1.

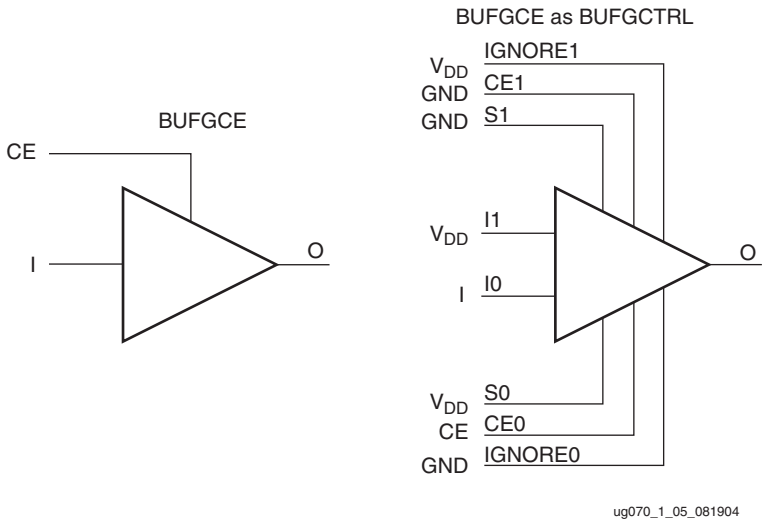


Figure 1-5: **BUFGCE as BUFGCTRL**

The switching condition for BUFGCE is similar to BUFGCTRL. If the CE input is Low prior to the incoming rising clock edge, the following clock pulse does not pass through the clock buffer, and the output stays Low. Any level change of CE during the incoming clock High pulse has no effect until the clock transitions Low. The output stays Low when the clock is disabled. However, when the clock is being disabled it completes the clock High pulse.

Since the clock enable line uses the CE pin of the BUFGCTRL, the select signal must meet the setup time requirement. Violating this setup time may result in a glitch. Figure 1-6 illustrates the timing diagram for BUFGCE.

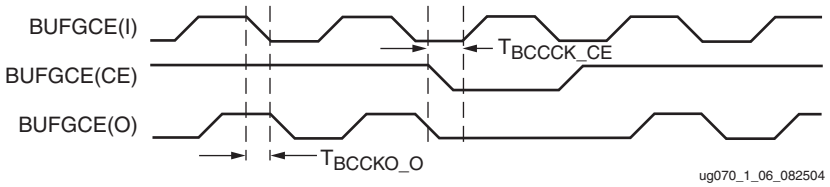


Figure 1-6: **BUFGCE Timing Diagram**

BUFGCE_1 is similar to BUFGCE, with the exception of its switching condition. If the CE input is Low prior to the incoming falling clock edge, the following clock pulse does not pass through the clock buffer, and the output stays High. Any level change of CE during the incoming clock Low pulse has no effect until the clock transitions High. The output stays High when the clock is disabled. However, when the clock is being disabled it completes the clock Low pulse.

Figure 1-7 illustrates the timing diagram for BUFGCE_1.

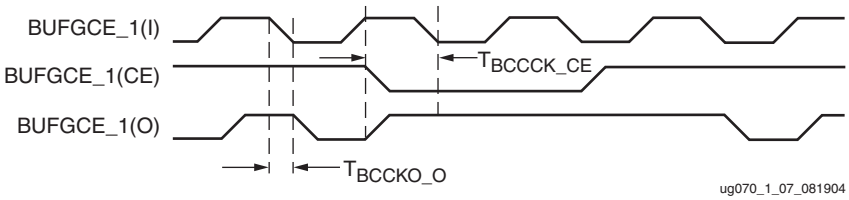


Figure 1-7: **BUFGCE_1 Timing Diagram**

BUFGMUX and BUFGMUX_1

BUFGMUX is a clock buffer with two clock inputs, one clock output, and a select line. This primitive is based on BUFGCTRL with some pins connected to logic High or Low. Figure 1-8 illustrates the relationship of BUFGMUX and BUFGCTRL. A LOC constraint is available for BUFGMUX and BUFGCTRL.

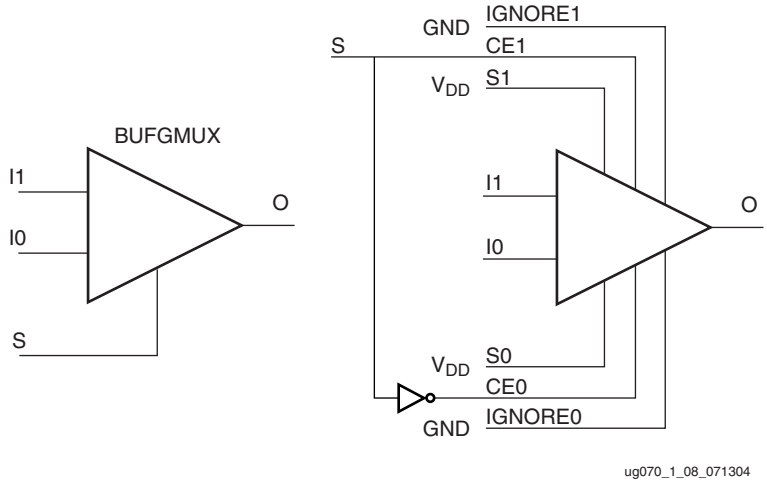


Figure 1-8: **BUFGMUX as BUFGCTRL**

Since the BUFGMUX uses the CE pins as select pins, when using the select, the setup time requirement must be met. Violating this setup time may result in a glitch.

Switching conditions for BUFGMUX are the same as the CE pins on BUFGCTRL. Figure 1-9 illustrates the timing diagram for BUFGMUX.

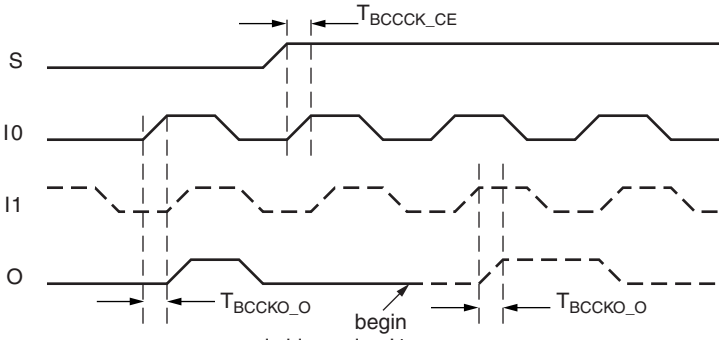


Figure 1-9: **BUFGMUX Timing Diagram**

In Figure 1-9:

- The current clock is I0.
- S is activated High.
- If I0 is currently High, the multiplexer waits for I0 to deassert Low.
- Once I0 is Low, the multiplexer output stays Low until I1 transitions High to Low.
- When I1 transitions from High to Low, the output switches to I1.
- If the setup/hold times are met, no glitches or short pulses can appear on the output.

BUFGMUX_1 is rising edge sensitive and held at High prior to input switch. Figure 1-10 illustrates the timing diagram for BUFGMUX_1. A LOC constraint is available for BUFGMUX and BUFGMUX_1.

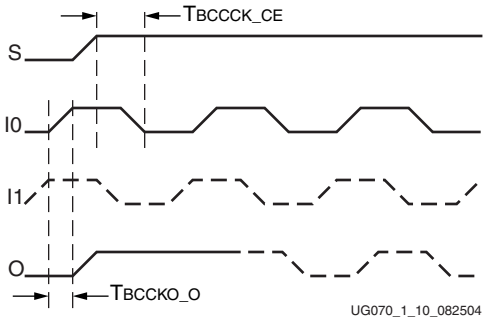


Figure 1-10: **BUFGMUX_1 Timing Diagram**

In Figure 1-10:

- The current clock is I0.
- S is activated High.
- If I0 is currently Low, the multiplexer waits for I0 to be asserted High.
- Once I0 is High, the multiplexer output stays High until I1 transitions Low to High.
- When I1 transitions from Low to High, the output switches to I1.
- If the setup/hold times are met, no glitches or short pulses can appear on the output.

BUFGMUX_VIRTEX4

BUFGMUX_VIRTEX4 is a clock buffer with two clock inputs, one clock output, and a select line. This primitive is based on BUFGCTRL with some pins connected to logic High or Low. Figure 1-11 illustrates the relationship of BUFGMUX_VIRTEX4 and BUFGCTRL.

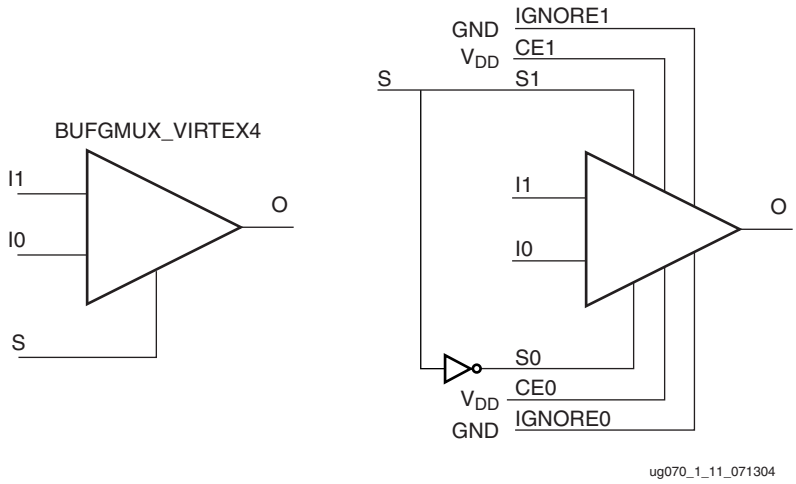


Figure 1-11: **BUFGMUX_VIRTEX4 as BUFGCTRL**

BUFGMUX_VIRTEX4 uses the S pins as select pins. S can switch anytime without causing a glitch. The setup/hold times on S determine whether the output will pass an extra pulse of the previously selected clock before switching to the new clock. If S changes as shown in Figure 1-12, prior to the setup time T_{BCCCK_S} and before I0 transitions from High to Low, then the output will not pass an extra pulse of I0. If S changes following the hold time for S, then the output will pass an extra pulse. If S violates the setup/hold requirements, the output might pass the extra pulse, but it will not glitch. In any case, the output changes to the new clock within three clock cycles of the slower clock.

The setup/hold requirements for S0 and S1 are with respect to the falling clock edge (assuming INIT_OUT = 0), not the rising edge as for CE0 and CE1.

Switching conditions for BUFGMUX_VIRTEX4 are the same as the S pin of BUFGCTRL. Figure 1-12 illustrates the timing diagram for BUFGMUX_VIRTEX4.

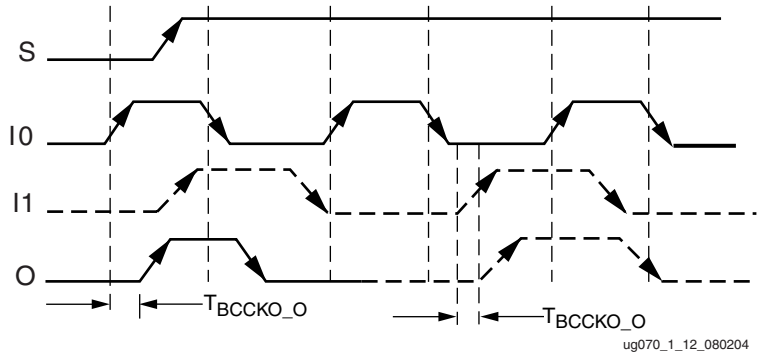


Figure 1-12: **BUFGMUX_VIRTEX4 Timing Diagram**

Other capabilities of the BUFGMUX_VIRTEX4 primitive are:

- Pre-selection of I0 and I1 input after configuration.
- Initial output can be selected as High or Low after configuration.

Additional Use Models

Asynchronous Mux Using BUFGCTRL

In some cases an application requires immediate switching between clock inputs or bypassing the edge sensitivity of BUFGCTRL. An example is when one of the clock inputs is no longer switching. If this happens, the clock output would not have the proper switching conditions because the BUFGCTRL never detected a clock edge. This case uses the asynchronous mux. Figure 1-13 illustrates an asynchronous mux with BUFGCTRL design example. Figure 1-14 shows the asynchronous mux timing diagram.

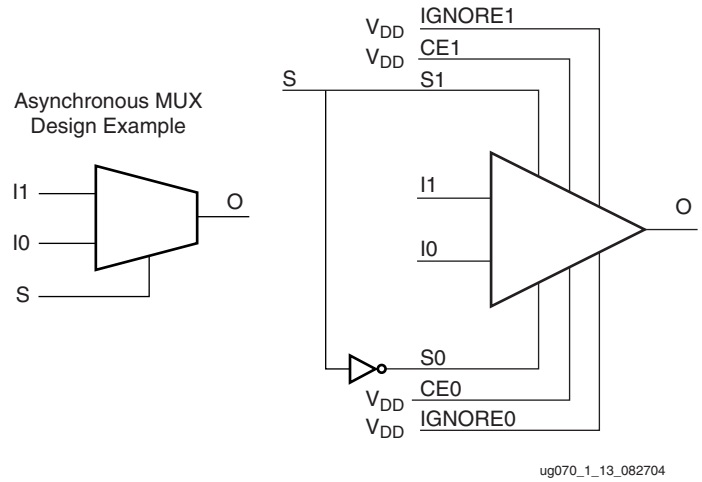


Figure 1-13: **Asynchronous Mux with BUFGCTRL Design Example**

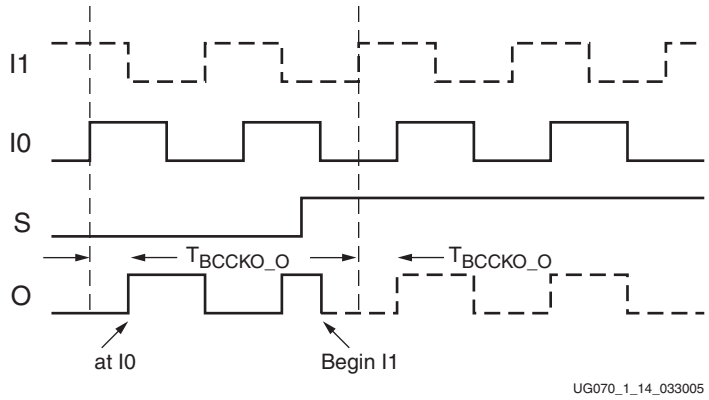


Figure 1-14: **Asynchronous Mux Timing Diagram**

In Figure 1-14:

- The current clock is from I0.
- S is activated High.
- The Clock output immediately switches to I1.
- When Ignore signals are asserted High, glitch protection is disabled.

BUFGMUX_VIRTEX4 with a Clock Enable

A BUFGMUX_VIRTEX4 with a clock enable BUFGCTRL configuration allows the user to choose between the incoming clock inputs. If needed, the clock enable is used to disable the output. Figure 1-15 illustrates the BUFGCTRL usage design example and Figure 1-16 shows the timing diagram.

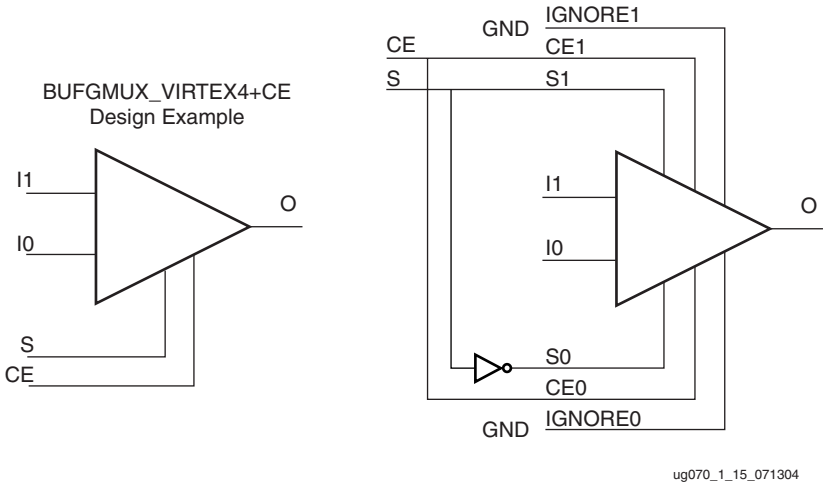


Figure 1-15: **BUFGMUX_VIRTEX4 with a CE and BUFGCTRL**

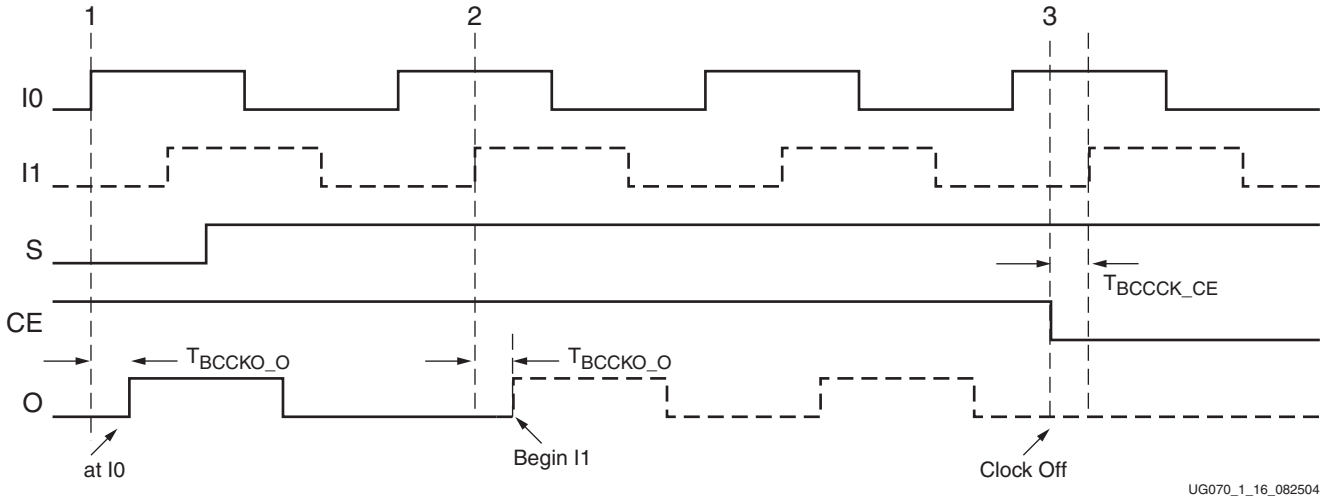


Figure 1-16: **BUFGMUX_VIRTEX4 with a CE Timing Diagram**

In Figure 1-16:

- At time event 1, output O uses input I0.
- Before time event 2, S is asserted High.
- At time T_{BCCKO_O}, after time event 2, output O uses input I1. This occurs after a High to Low transition of I0 followed by a High to Low transition of I1 is completed.
- At time T_{BCCCK_CE}, before time event 3, CE is asserted Low. The clock output is switched Low and kept at Low after a High to Low transition of I1 is completed.

Clock Tree and Nets - GCLK

Virtex-4 FPGA clock trees are designed for low-skew and low-power operation. Any unused branch is disconnected. The clock trees also manage the load/fanout when all the logic resources are used.

All global clock lines and buffers are implemented differentially. This facilitates much better duty cycles and common-mode noise rejection.

In the Virtex-4 architecture, the pin access of the global clock lines are not limited to the logic resources clock pins. The global clock lines can access other pins in the CLBs without using local interconnects. Applications requiring a very fast signal connection and large load/fanout benefit from this architecture.

Clock Regions

Virtex-4 devices improve the clocking distribution by the use of clock regions. Each clock region can have up to eight global clock domains. These eight global clocks can be driven by any combination of the 32 global clock buffers. The restrictions and rules needed in previous FPGA architectures are no longer applicable. Specifically, a clock region is not limited to four quadrants regardless of die/device size. Instead, the dimensions of a clock region are fixed to 16 CLBs tall (32 IOBs) and spanning half of the die (Figure 1-17). By fixing the dimensions of the clock region, larger Virtex-4 devices can have more clock regions. As a result, Virtex-4 devices can support many more multiple clock domains than previous FPGA architectures. Table 1-6 shows the number of clock regions in each Virtex-4 device. The logic resources in the center column (DCMs, IOBs, etc.) are located in the left clock regions.

The DCMs, if used, utilize the global clocks in the left regions as feedback lines. Up to four DCMs can be in a specific region. If used in the same region, IDELAYCTRL uses another global clock in that region. The DCM companion module PMCD, if directly connected to a global clock, will also utilize the global clocks in the same region.

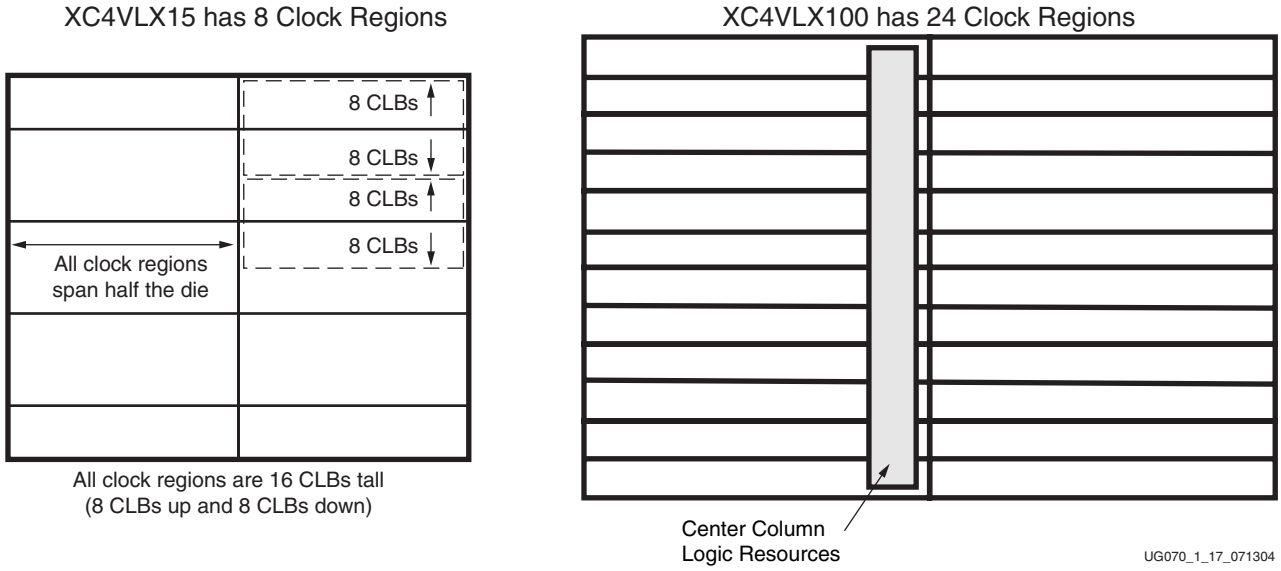


Figure 1-17: **Clock Regions**

Table 1-6: **Virtex-4 FPGA Clock Regions**

Device	Number of Clock Regions
LX Family	
XC4VLX15	8
XC4VLX25	12
XC4VLX40	16
XC4VLX60	16
XC4VLX80	20
XC4VLX100	24
XC4VLX160	24
XC4VLX200	24
SX Family	
XC4VSX25	8
XC4VSX35	12
XC4VSX55	16
FX Family	
XC4VFX12	8
XC4VFX20	8
XC4VFX40	12
XC4VFX60	16
XC4VFX100	20
XC4VFX140	24

Regional Clocking Resources

Regional clock networks are a set of clock networks independent of the global clock network. Unlike global clocks, the span of a regional clock signal is limited to three clock regions. These networks are especially useful for source-synchronous interface designs.

To understand how regional clocking works, it is important to understand the signal path of a regional clock signal. The Virtex-4 FPGA regional clocking resources and network consist of the following paths and components:

- Clock Capable I/O
- I/O Clock Buffer - BUFIO
- Regional Clock Buffer - BUFR
- Regional Clock Nets

Clock Capable I/O

In a typical clock region there are two clock capable I/O pin pairs (there are exceptions in the center column). Clock capable I/O pairs are regular I/O pairs where the LVDS output drivers have been removed to reduce the input capacitance. All global clock inputs are clock capable I/Os (i.e., they do not have LVDS output drivers). There are four dedicated clock capable I/O sites in every bank. When used as clock inputs, clock-capable pins can drive BUFIO and BUFR. They can not directly connect to the global clock buffers. When used as single-ended clock pins, then as described in "Global Clock Buffers", the P-side of the pin pair must be used because a direct connection only exists on this pin.

I/O Clock Buffer - BUFIO

The I/O clock buffer (BUFIO) is a new clock buffer available in Virtex-4 devices. The BUFIO drives a dedicated clock net within the I/O column, independent of the global clock resources. Thus, BUFIOs are ideally suited for source-synchronous data capture (forwarded/receiver clock distribution). BUFIOs can only be driven by clock capable I/Os located in the same clock region. BUFIOs can drive the two adjacent I/O clock nets (for a total of up to three clock regions) as well as the regional clock buffers (BUFR) in the same region. BUFIOs cannot drive logic resources (CLB, block RAM, etc.) because the I/O clock network only reaches the I/O column.

BUFIO Primitive

BUFIO is simply a clock in, clock out buffer. There is a phase delay between input and output. Figure 1-18 shows the BUFIO. Table 1-7 lists the BUFIO ports. A location constraint is available for BUFIO.

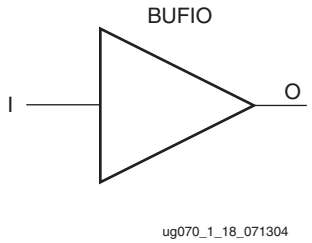


Figure 1-18: **BUFIO Primitive**

Table 1-7: **BUFIO Port List and Definitions**

Port Name	Type	Width	Definition
O	Output	1	Clock output port
I	Input	1	Clock input port

BUFIO Use Models

In Figure 1-19, a BUFIO is used to drive the I/O logic using the clock capable I/O. This implementation is ideal in source-synchronous applications where a forwarded clock is used to capture incoming data.

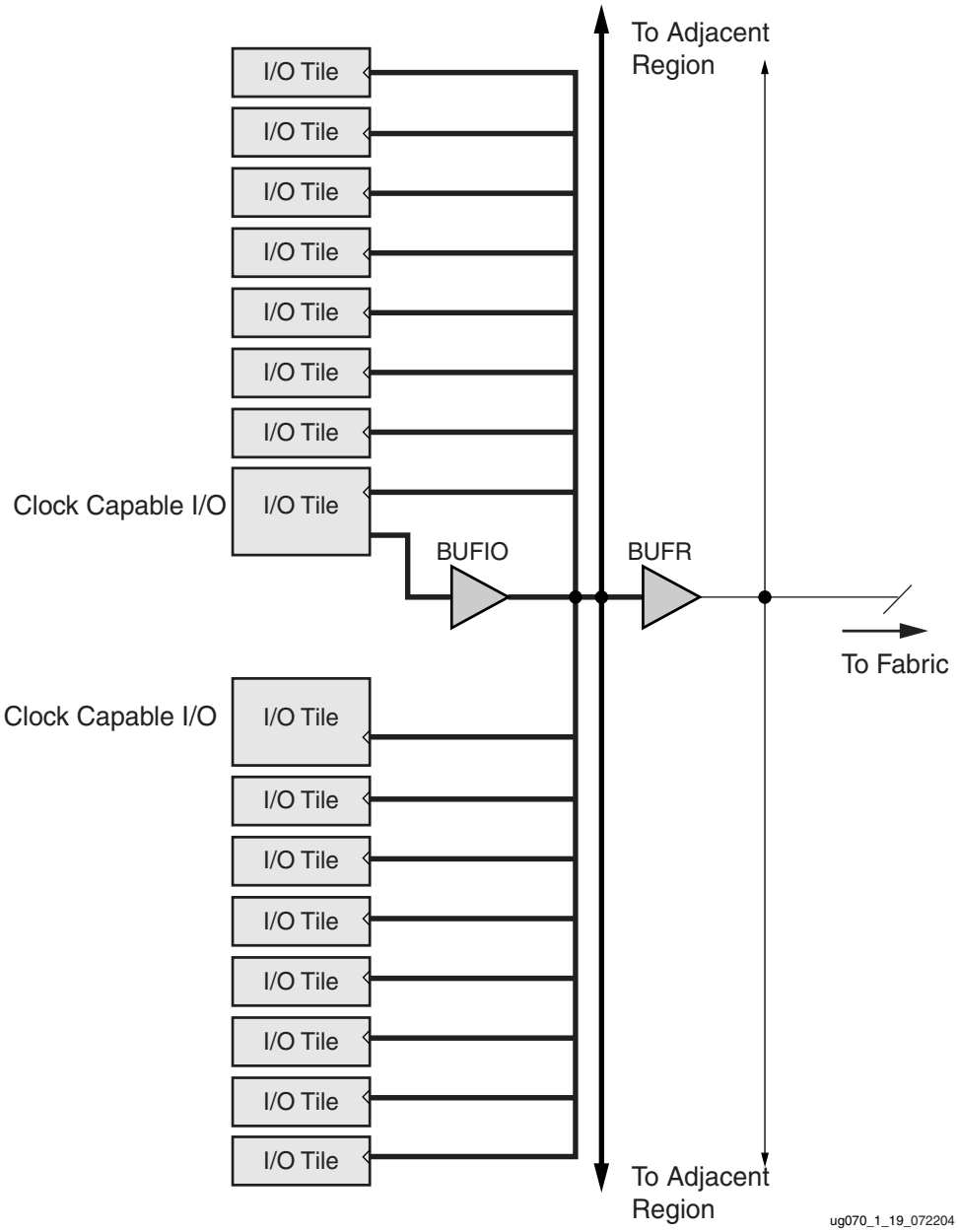


Figure 1-19: **BUFIO Driving I/O Logic In a Single Clock Region**

Regional Clock Buffer - BUFR

The regional clock buffer (BUFR) is another new clock buffer available in Virtex-4 devices. BUFRs drive clock signals to a dedicated clock net within a clock region, independent from the global clock tree. Each BUFR can drive the two regional clock nets in the region it is located, and the two clock nets in the adjacent clock regions (up to three clock regions).

Unlike BUFIOs, BUFRs can drive the I/O logic *and* logic resources (CLB, block RAM, etc.) in the existing and adjacent clock regions. BUFRs can be driven by either the output from BUFIOs or local interconnect. In addition, BUFR is capable of generating divided clock outputs with respect to the clock input. The divide values are an integer between one and eight. BUFRs are ideal for source-synchronous applications requiring clock domain crossing or serial-to-parallel conversion. There are two BUFRs in a typical clock region (two regional clock networks). The center column does not have BUFRs.

BUFR Primitive

BUFR is a clock-in/clock-out buffer with the capability to divide the input clock frequency.

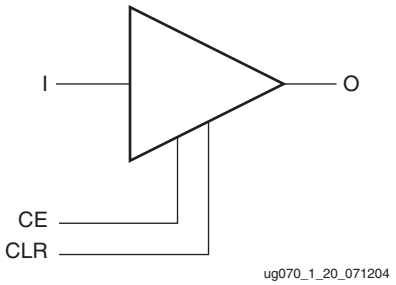


Figure 1-20: **BUFR Primitive**

Table 1-8: **BUFR Port List and Definitions**

Port Name	Type	Width	Definition
O	Output	1	Clock output port
CE	Input	1	Clock enable port. Cannot be used in BYPASS mode.
CLR	Input	1	Asynchronous clear for the divide logic, and sets the output Low. Cannot be used in BYPASS mode.
I	Input	1	Clock input port

Additional Notes on the CE Pin

When CE is asserted/deasserted, the output clock signal turns on/off four input clock cycles later. When global set/reset (GSR) signal is High, BUFR does not toggle, even if CE is held High. The BUFR output toggles four clock cycles after the GSR signal is deasserted.

BUFR Attributes and Modes

Clock division in the BUFR is controlled in software through the BUFR_DIVIDE attribute. Table 1-9 lists the possible values when using the BUFR_DIVIDE attribute.

Table 1-9: **BUFR_DIVIDE Attribute**

Attribute Name	Description	Possible Values
BUFR_DIVIDE	Defines whether the output clock is a divided version of the input clock.	1, 2, 3, 4, 5, 6, 7, 8 BYPASS (default)

Notes:

1. Location constraint is available for BUFR.

The propagation delay through BUFR is different for BUFR_DIVIDE = 1 and BUFR_DIVIDE = BYPASS. When set to 1, the delay is slightly more than BYPASS. All other divisors have the same delay BUFR_DIVIDE = 1. The phase relationship between the input clock and the output clock is the same for all possible divisions except BYPASS.

The timing relationship between the inputs and output of BUFR when using the BUFR_DIVIDE attribute is illustrated in Figure 1-21. In this example, the BUFR_DIVIDE attribute is set to three. Sometime before this diagram CLR was asserted.

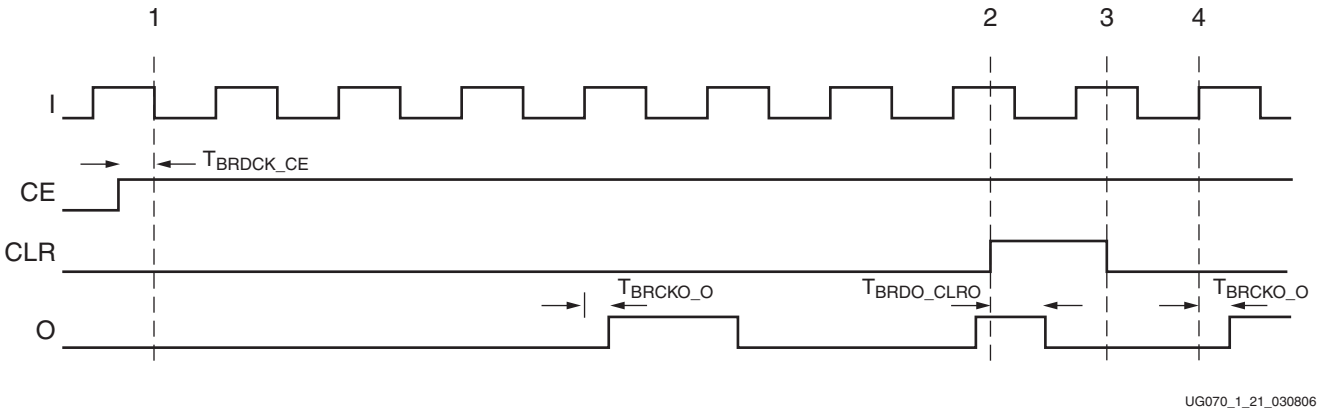


Figure 1-21: **BUFR Timing Diagrams with BUFR_DIVIDE Values**

In Figure 1-21:

- At time T_{BRDCK_CE} before clock event 1, CE is asserted High.
- Four clock cycles and T_{BRCKO_O} after CE is asserted, the output O begins toggling at the divide by three rate of the input I. T_{BRCKO_O} and other timing numbers are best found in the speed specification.

  Note: The duty cycle is not 50/50 for odd division. The Low pulse is one cycle of I longer.
- At time event 2, CLR is asserted. After T_{BRDO_CLRO} from time event 2, O stops toggling.
- At time event 3, CLR is deasserted.
- At time T_{BRCKO_O} after clock event 4, O begins toggling again at the divided by three rate of I.

BUFR Use Models

BUFRs are ideal for source-synchronous applications requiring clock domain crossing or serial-to-parallel conversion. Unlike BUFIOs, BUFRs are capable of clocking logic resources in the FPGAs other than the IOBs. Figure 1-22 is a BUFR design example.

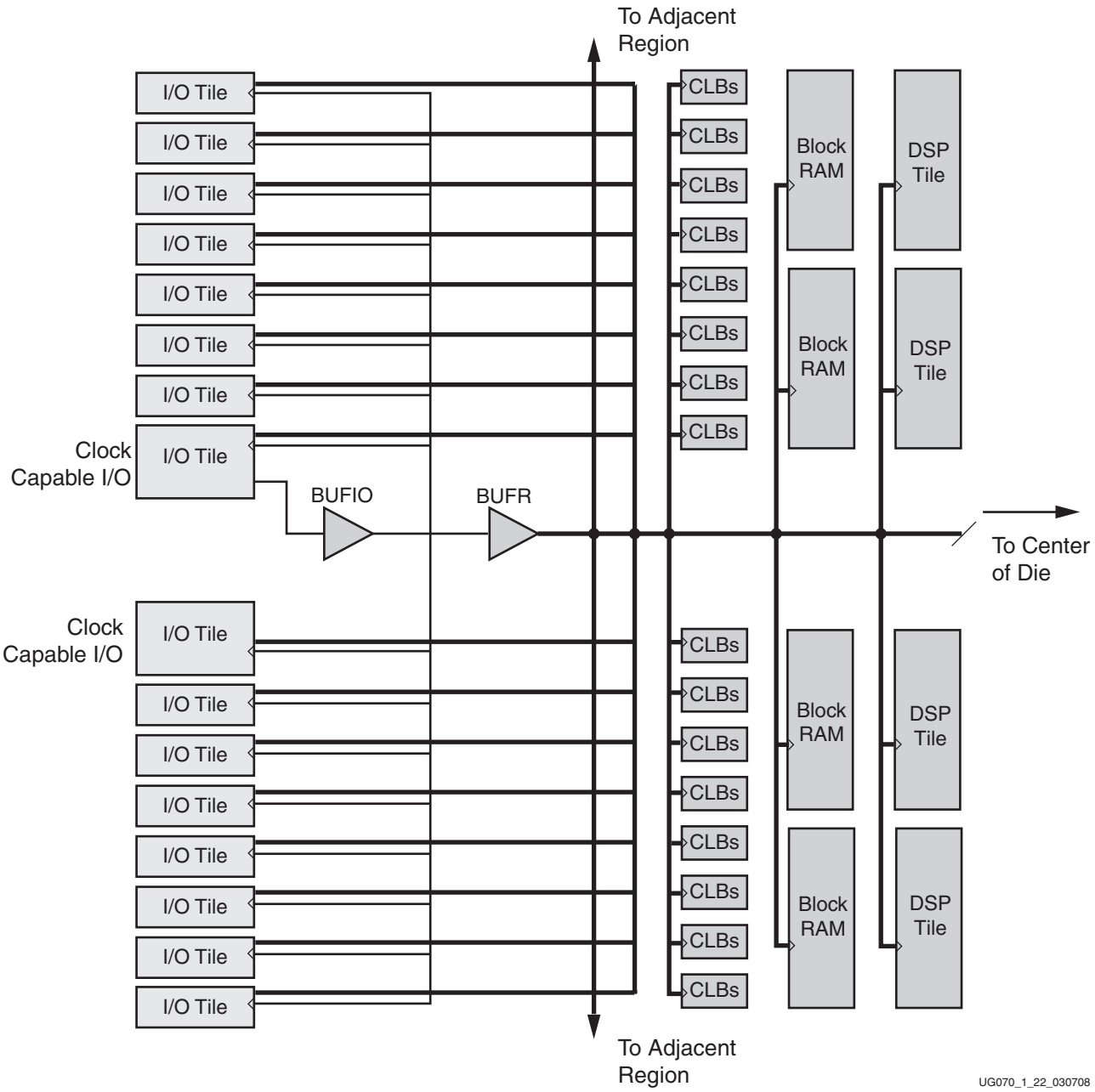


Figure 1-22: **BUFR Driving Various Logic Resources**

Regional Clock Nets

In addition to global clock trees and nets, Virtex-4 devices contain regional clock nets. These clock trees are also designed for low-skew and low-power operation. Unused branches are disconnected. The clock trees also manage the load/fanout when all the logic resources are used.

Regional clock nets do not propagate throughout the whole Virtex-4 device. Instead, they are limited to only one clock region. One clock region contains two independent regional clock nets.

To access regional clock nets, BUFRs must be instantiated. A BUFR can drive regional clocks in up to two adjacent clock regions (Figure 1-23). BUFRs in the top or bottom region can only access one adjacent region; below or above respectively.

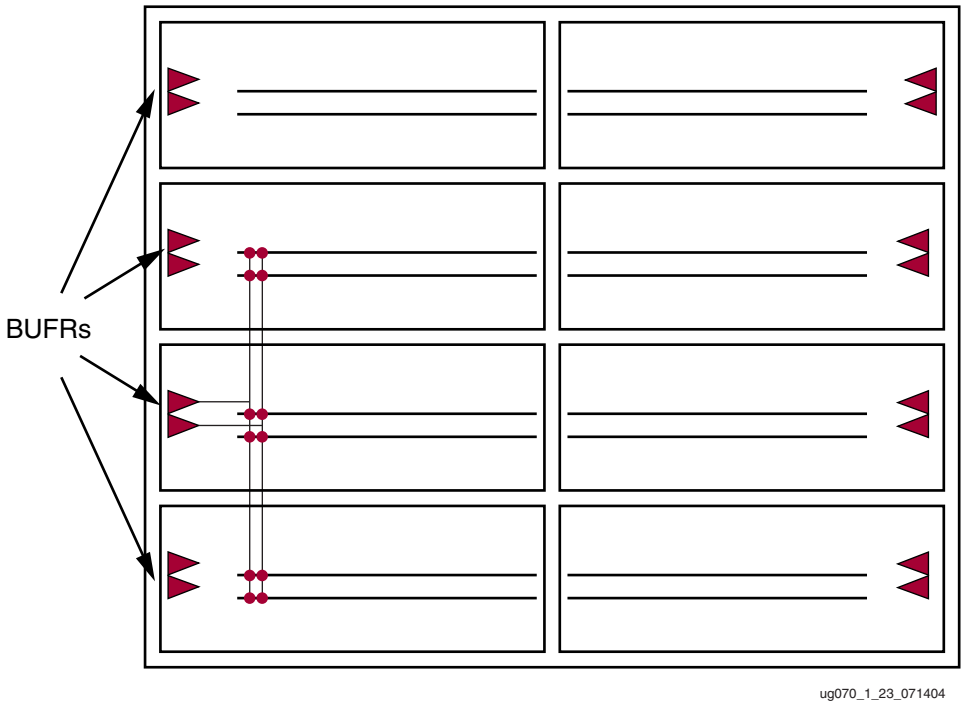


Figure 1-23: **BUFR Driving Multiple Regions**

VHDL and Verilog Templates

The VHDL and Verilog code follows for all clocking resource primitives.

BUFGCTRL VHDL and Verilog Templates

The following examples illustrate the instantiation of the BUFGCTRL module in VHDL and Verilog.

VHDL Template

```
--Example BUFGCTRL declaration

component BUFGCTRL
  generic(
      INIT_OUT     : integer := 0;
```

```
      PRESELECT_I0 : boolean := false;
      PRESELECT_I1 : boolean := false;
  );

port(
    O: out std_ulogic;
    CE0: in  std_ulogic;
    CE1: in  std_ulogic;
    I0: in  std_ulogic;
    I1        : in  std_ulogic;
    IGNORE0: in  std_ulogic;
    IGNORE1: in  std_ulogic;
    S0: in  std_ulogic;
    S1: in  std_ulogic
    );
end component;

--Example BUFGCTRL instantiation
U_BUFGCTRL : BUFGCTRL
Port map (
    O => user_o,
    CE0 => user_ce0,
    CE1 => user_ce1,
    I0 => user_i0,
    I1 => user_i1,
    IGNORE0 => user_ignore0,
    IGNORE1 => user_ignore1,
    S0 => user_s0,
    S1 => user_s1
    );

--Declaring constraints in VHDL file
attribute INIT_OUT     : integer;
attribute PRESELECT_I0 : boolean;
attribute PRESELECT_I1 : boolean;
attribute LOC : string;
attribute INIT_OUT of U_BUFGCTRL: label is 0;
attribute PRESELECT_I0 of U_BUFGCTRL: label is FALSE;
attribute PRESELECT_I1 of U_BUFGCTRL: label is FALSE;
attribute LOC of U_BUFGCTRL: label is "BUFGCTRL_X#Y#";

--where # is valid integer locations of BUFGCTRL
```

Verilog Template

```
//Example BUFGCTRL module declaration
module BUFGCTRL (O, CE0, CE1, I0, I1, IGNORE0, IGNORE1, S0, S1);
    output O;
    input CE0;
    input CE1;
    input I0;
    input I1;
    input IGNORE0;
    input IGNORE1;
    input S0;
    input S1;
    parameter INIT_OUT = 0;
    parameter PRESELECT_I0 = "FALSE";
    parameter PRESELECT_I1 = "FALSE";
```

```
endmodule;
//Example BUFGCTRL instantiation
BUFGCTRL U_BUFGCTRL (
  .O(user_o),
  .CE0(user_ce0),
  .CE1(user_ce1),
  .I0(user_i0),
  .I1(user_i1),
  .IGNORE0(user_ignore0),
  .IGNORE1(user_ignore1),
  .S0(user_s0),
  .S1(user_s1)
  );

// Declaring constraints in Verilog
// synthesis attribute INIT_OUT of U_BUFGCTRL is 0;
// synthesis attribute PRESELECT_I0 of U_BUFGCTRL is FALSE;
// synthesis attribute PRESELECT_I1 of U_BUFGCTRL is FALSE;
// synthesis attribute LOC of U_BUFGCTRL is "BUFGCTRL_X#Y#";
// where # is valid integer locations of BUFGCTRL
```

Declaring Constraints in UCF File

```
INST "U_BUFGCTRL" INIT_OUT = 0;
INST "U_BUFGCTRL" PRESELECT_I0 = FALSE;
INST "U_BUFGCTRL" PRESELECT_I1 = FALSE;
INST "U_BUFGCTRL" LOC = BUFGCTRL_X#Y#;
where # is valid integer locations of BUFGCTRL
```

BUFG VHDL and Verilog Templates

The following examples illustrate the instantiation of the BUFG module in VHDL and Verilog.

VHDL Template

```
--Example BUFG declaration
component BUFG
port(
    O: out std_ulogic;
    I: in  std_ulogic
);
end component;

--Example BUFG instantiation
U_BUFG : BUFG
Port map (
    O => user_o,
    I0 => user_i
    );

--Declaring constraints in VHDL file
attribute LOC : string;
attribute LOC of U_BUFG: label is "BUFGCTRL_X#Y#";

--where # is valid integer locations of BUFGCTRL
```

Verilog Template

```
//Example BUFG module declaration
module BUFG (O, I);
    output O;
    input I;
endmodule;
//Example BUFG instantiation
BUFG U_BUFG (
.O(user_o),
.I0(user_i)
);

// Declaring constraints in Verilog
// synthesis attribute LOC of U_BUFG is "BUFGCTRL_X#Y#";
// where # is valid integer locations of BUFGCTRL
```

Declaring Constraints in UCF File

```
INST "U_BUFG" LOC = BUFGCTRL_X#Y#;

where # is valid integer locations of BUFGCTRL
```

BUFGCE and BUFGCE_1 VHDL and Verilog Templates

The following examples illustrate the instantiation of the BUFGCE module in VHDL and Verilog. The instantiation of BUFGCE_1 is exactly the same as BUFGCE with exception of the primitive name.

VHDL Template

```
--Example BUFGCE declaration
component BUFGCE
port(
    O: out std_ulogic;
    CE: in  std_ulogic;
    I: in  std_ulogic
    );
end component;

--Example BUFGCE instantiation

U_BUFGCE : BUFGCE
Port map (
    O => user_o,
    CE => user_ce,
    I => user_i
    );

--Declaring constraints in VHDL file

attribute LOC : string;
attribute LOC of U_BUFGCE: label is "BUFGCTRL_X#Y#";

--where # is valid integer locations of BUFGCTRL
```

Verilog Template

```
//Example BUFGCE module declaration
module BUFGCE (O, CE, I);
    output O;
    input CE;
    input I;

endmodule;

//Example BUFGCE instantiation
BUFGCE U_BUFGCE (
.O(user_o),
.CE0(user_ce),
.I0(user_i)
);

// Declaring constraints in Verilog
// synthesis attribute LOC of U_BUFGCE is "BUFGCTRL_X#Y#";
// where # is valid integer locations of BUFGCTRL
```

Declaring Constraints in UCF File

```
INST "U_BUFGCE" LOC = BUFGCTRL_X#Y#;

where # is valid integer locations of BUFGCTRL
```

BUFGMUX and BUFGMUX_1 VHDL and Verilog Templates

The following examples illustrate the instantiation of the BUFGMUX module in VHDL and Verilog. The instantiation of BUFGMUX_1 is exactly the same as BUFGMUX with exception of the primitive name.

VHDL Template

```
--Example BUFGMUX declaration
component BUFGMUX
port(
    O: out std_ulogic;
    I0: in  std_ulogic;
    I1        : in  std_ulogic;
    S: in  std_ulogic
    );
end component;

--Example BUFGMUX instantiation
U_BUFGMUX : BUFGMUX
Port map (
    O => user_o,
    I0 => user_i0,
    I1 => user_i1,
    S => user_s
    );

--Declaring constraints in VHDL file
attribute LOC : string;
attribute LOC of U_BUFGMUX: label is "BUFGCTRL_X#Y#";
--where # is valid integer locations of BUFGCTRL
```

Verilog Template

```
//Example BUFGMUX module declaration

module BUFGMUX (O, I0, I1, S);

    output O;
    input I0;
    input I1;
    input S;

endmodule;

//Example BUFGMUX instantiation

BUFGMUX U_BUFGMUX (
.O(user_o),
.I0(user_i0),
.I1(user_i1),
.S0(user_s)
);

// Declaring constraints in Verilog
// synthesis attribute LOC of U_BUFGMUX is "BUFGCTRL_X#Y#";
// where # is valid integer locations of BUFGCTRL
```

Declaring Constraints in UCF File

```
INST "U_BUFGMUX" LOC = BUFGCTRL_X#Y#;

where # is valid integer locations of BUFGCTRL
```

BUFGMUX_VIRTEX4 VHDL and Verilog Templates

The following examples illustrate the instantiation of the BUFGMUX_VIRTEX4 module in VHDL and Verilog.

VHDL Template

```
--Example BUFGMUX_VIRTEX4 declaration
component BUFGMUX_VIRTEX4
port(
    O  : out std_ulogic;
    I0 : in  std_ulogic;
    I1 : in  std_ulogic;
    S  : in  std_ulogic
    );
end component;

--Example BUFGMUX_VIRTEX4 instantiation

U_BUFGMUX_VIRTEX4 : BUFGMUX_VIRTEX4
Port map (
    O => user_o,
    I0 => user_i0,
    I1 => user_i1,
    S => user_s
    );
```

```
--Declaring constraints in VHDL file

attribute INIT_OUT     : integer;
attribute PRESELECT_I0 : boolean;
attribute PRESELECT_I1 : boolean;
attribute LOC : string;

attribute INIT_OUT of U_BUFGMUX_VIRTEX4: label is 0;
attribute PRESELECT_I0 of U_BUFGMUX_VIRTEX4: label is FALSE;
attribute PRESELECT_I1 of U_BUFGMUX_VIRTEX4: label is FALSE;
attribute LOC of U_BUFGMUX_VIRTEX4: label is "BUFGCTRL_X#Y#";

--where # is valid integer locations of BUFGCTRL
```

Verilog Template

```
//Example BUFGMUX_VIRTEX4 module declaration

module BUFGMUX_VIRTEX4 (O, I0, I1, S);

    output O;
    input  I0;
    input  I1;
    input  S;

    parameter INIT_OUT = 1'b0;
    parameter PRESELECT_I0 = "TRUE";
    parameter PRESELECT_I1 = "FALSE";

endmodule;

//Example BUFGCTRL instantiation

BUFGMUX_VIRTEX4 U_BUFGMUX_VIRTEX4 (
  .O(user_o),
  .I0(user_i0),
  .I1(user_i1),
  .S(user_s)
);

// Declaring constraints in Verilog
// synthesis attribute INIT_OUT of U_BUFGMUX_VIRTEX4 is 0;
// synthesis attribute PRESELECT_I0 of U_BUFGMUX_VIRTEX4 is FALSE;
// synthesis attribute PRESELECT_I1 of U_BUFGMUX_VIRTEX4 is FALSE;
// synthesis attribute LOC of U_BUFGMUX_VIRTEX4 is "BUFGCTRL_X#Y#";
// where # is valid integer locations of BUFGCTRL
```

Declaring Constraints in UCF File

```
INST "U_BUFGMUX_VIRTEX4" INIT_OUT = 0;
INST "U_BUFGMUX_VIRTEX4" PRESELECT_I0 = FALSE;
INST "U_BUFGMUX_VIRTEX4" PRESELECT_I1 = FALSE;
INST "U_BUFGMUX_VIRTEX4" LOC = BUFGCTRL_X#Y#;

where # is valid integer locations of BUFGCTRL
```

BUFIO VHDL and Verilog Templates

The following examples illustrate the instantiation of the BUFIO module in VHDL and Verilog.

VHDL Template

```
--Example BUFIO declaration

component BUFIO
port(
    O: out std_ulogic;
    I: in  std_ulogic
    );
end component;

--Example BUFIO instantiation

U_BUFIO : BUFIO
Port map (
    O => user_o,
    I0 => user_i
    );

--Declaring constraints in VHDL file

attribute LOC : string;
attribute LOC of U_BUFIO: label is "BUFIO_X#Y#";

--where # is valid integer locations of BUFIO
```

Verilog Template

```
//Example BUFIO module declaration

module BUFIO (O, I);

    output O;
    input I;

endmodule;

//Example BUFIO instantiation

BUFIO U_BUFIO (
.O(user_o),
.I(user_i)
);

// Declaring constraints in Verilog
// synthesis attribute LOC of U_BUFIO is "BUFIO_X#Y#";
// where # is valid integer locations of BUFIO
```

Declaring Constraints in UCF File

```
INST "U_BUFIO" LOC = BUFIO_X#Y#;
where # is valid integer locations of BUFIO
```

BUFR VHDL and Verilog Templates

The following examples illustrate the instantiation of the BUFR module in VHDL and Verilog.

VHDL Template

```
--Example BUFR declaration

component BUFR
generic(
      BUFR_DIVIDE   : string := "BYPASS";
);

  port(
      O: out std_ulogic;
      CE: in  std_ulogic;
      CLR: in  std_ulogic;
      I: in  std_ulogic
    );
  end component;

--Example BUFR instantiation

U_BUFR : BUFR
Port map (
    O => user_o,
    CE => user_ce,
    CLR => user_clr,
    I => user_i
    );

--Declaring constraints in VHDL file

attribute BUFR_DIVIDE    : string;
attribute LOC : string;
attribute INIT_OUT of U_BUFR: label is BYPASS;
attribute LOC of U_BUFR: label is "BUFR_X#Y#";

--where # is valid integer locations of BUFR
```

Verilog Template

```
//Example BUFR module declaration

module BUFR (O, CE, CLR, I);
    output O;
    input CE;
    input CLR;
    input I;
    parameter BUFR_DIVIDE = "BYPASS";

endmodule;
//Example BUFR instantiation
BUFR U_BUFR (
.O(user_o),
.CE(user_ce),
.CLR(user_clr),
```

```
.I(user_i)
);

// Declaring constraints in Verilog
// synthesis attribute BUFR_DIVIDE of U_BUFR is BYPASS;
// synthesis attribute LOC of U_BUFR is "BUFR_X#Y#";
// where # is valid integer locations of BUFR
```

Declaring Constraints in UCF File

```
INST "U_BUFR" BUFR_DIVIDE=BYPASS;
INST "U_BUFR" LOC = BUFR_X#Y#;
where # is valid integer locations of BUFR
```

XILINX®

Chapter 2

Digital Clock Managers (DCMs)

DCM Summary

The Virtex®-4 FPGA Digital Clock Managers (DCMs) provide a wide range of powerful clock management features:

- **Clock Deskew**

  The DCM contains a delay-locked loop (DLL) to completely eliminate clock distribution delays, by deskewing the DCM's output clocks with respect to the input clock. The DLL contains delay elements (individual small buffers) and control logic. The incoming clock drives a chain of delay elements, thus the output of every delay element represents a version of the incoming clock delayed at a different point.

  The control logic contains a phase detector and a delay-line selector. The phase detector compares the incoming clock signal (CLKIN) against a feedback input (CLKFB) and steers the delay line selector, essentially adding delay to the output of DCM until the CLKIN and CLKFB coincide.

- **Frequency Synthesis**

  Separate outputs provide a doubled frequency (CLK2X and CLK2X180). Another output, CLKDV, provides a frequency that is a specified fraction of the input frequency.

  Two other outputs, CLKFX and CLKFX180, provide an output frequency derived from the input clock by simultaneous frequency division and multiplication. The user can specify any integer multiplier (M) and divisor (D) within the range specified in the DCM Timing Parameters section of the *Virtex-4 Data Sheet*. An internal calculator determines the appropriate tap selection, to make the output edge coincide with the input clock whenever mathematically possible. For example, M = 9 and D = 5, multiply the frequency by 1.8, and the output rising edge is coincident with the input rising edge after every fifth input period, or after every ninth output period.

- **Phase Shifting**

  The DCM allows coarse and fine-grained phase shifting. The coarse phase shifting uses the 90°, 180°, and 270° phases of CLK0 to make CLK90, CLK180, and CLK270 clock outputs. The 180° phase of CLK2X and CLKFX provide the respective CLK2X180 and CLKFX180 clock outputs.

  There are also four modes of fine-grained phase-shifting; fixed, variable-positive, variable-center, and direct modes. Fine-grained phase shifting allows all DCM output clocks to be phase-shifted with respect to CLKIN while maintaining the relationship between the coarse phase outputs. With fixed mode, a fixed fraction of phase shift can be defined during configuration and in multiples of the clock period divided by 256. Using the variable-positive and variable-center modes the phase can be dynamically and repetitively moved forward and backwards by 1/256 of the clock period. With the

direct mode the phase can be dynamically and repetitively moved forward and backwards by the value of one DCM_TAP. See the DCM Timing Parameters section in the *Virtex-4 Data Sheet*.

- **Dynamic Reconfiguration**

  There is a bus connection to the DCM to change DCM attributes without reconfiguring the rest of the device. For more information, see the Dynamic Reconfiguration chapter of the *Virtex-4 Configuration Guide*.

  The DADDR[6:0], DI[15:0], DWE, DEN, DCLK inputs and DO[15:0], and DRDY outputs are available to dynamically reconfigure select DCM functions. With dynamic reconfiguration, DCM attributes can be changed to select a different phase shift, multiply (M) or divide (D) from the currently configured settings.

Figure 2-1 shows a simplified view of the Virtex-4 FPGA center column resources including all DCM locations. Table 2-1 summarizes the availability of DCMs in each Virtex-4 device.

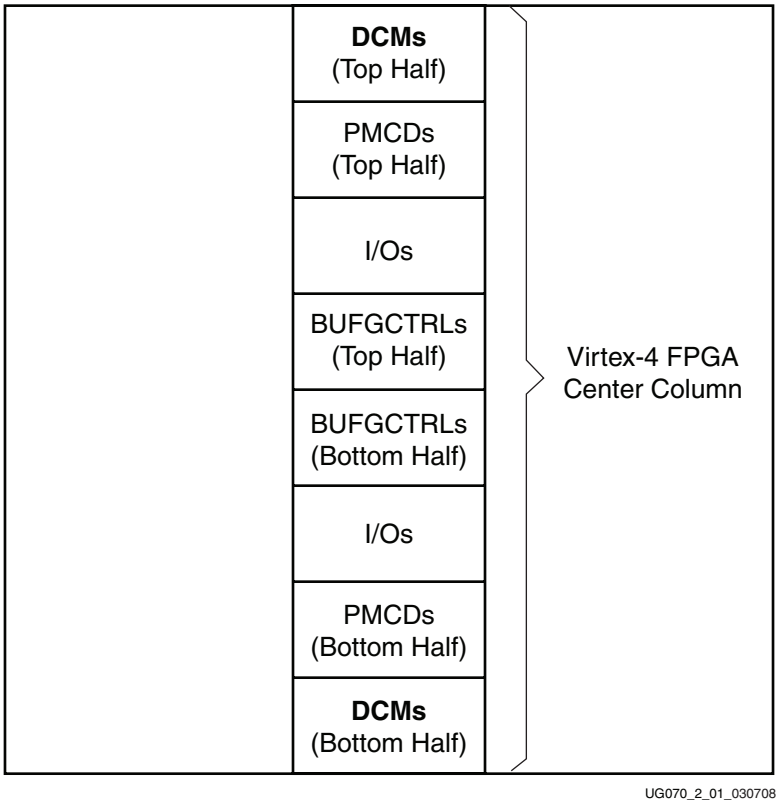


Figure 2-1: **DCM Location**

Table 2-1: **Available DCM Resources**

Device	Available DCMs	Site Names
XC4VLX15 XC4VSX25 XC4VFX12, XC4VFX20	4	*Bottom Half:* DCM_ADV_X0Y0, DCM_ADV_X0Y1 *Top Half:* DCM_ADV_X0Y2, DCM_ADV_X0Y3
XC4VLX25, XC4VLX40, XC4VLX60 XC4VSX35, XC4VSX55 XC4VFX40	8	*Bottom Half:* DCM_ADV_X0Y0, DCM_ADV_X0Y1, DCM_ADV_X0Y2 *Top Half:* DCM_ADV_X0Y3, DCM_ADV_X0Y4, DCM_ADV_X0Y5, DCM_ADV_X0Y6, DCM_ADV_X0Y7
XC4VLX80, XC4VLX100, XC4VLX160, XC4VLX200 XC4VFX60, XC4VFX100	12	*Bottom Half:* DCM_ADV_X0Y0, DCM_ADV_X0Y1, DCM_ADV_X0Y2, DCM_ADV_X0Y3, DCM_ADV_X0Y4, DCM_ADV_X0Y5 *Top Half:* DCM_ADV_X0Y6, DCM_ADV_X0Y7, DCM_ADV_X0Y8, DCM_ADV_X0Y9, DCM_ADV_X0Y10, DCM_ADV_X0Y11
XC4VFX140	20	*Bottom Half:* DCM_ADV_X0Y0, DCM_ADV_X0Y1, DCM_ADV_X0Y2, DCM_ADV_X0Y3, DCM_ADV_X0Y4, DCM_ADV_X0Y5, DCM_ADV_X0Y6, DCM_ADV_X0Y7, DCM_ADV_X0Y8, DCM_ADV_X0Y9 *Top Half:* DCM_ADV_X0Y10, DCM_ADV_X0Y11 DCM_ADV_X0Y12, DCM_ADV_X0Y13 DCM_ADV_X0Y14, DCM_ADV_X0Y15 DCM_ADV_X0Y16, DCM_ADV_X0Y17 DCM_ADV_X0Y18, DCM_ADV_X0Y19

DCM Primitives

Three DCM primitives are available: DCM_BASE, DCM_PS, and DCM_ADV (see Figure 2-2).

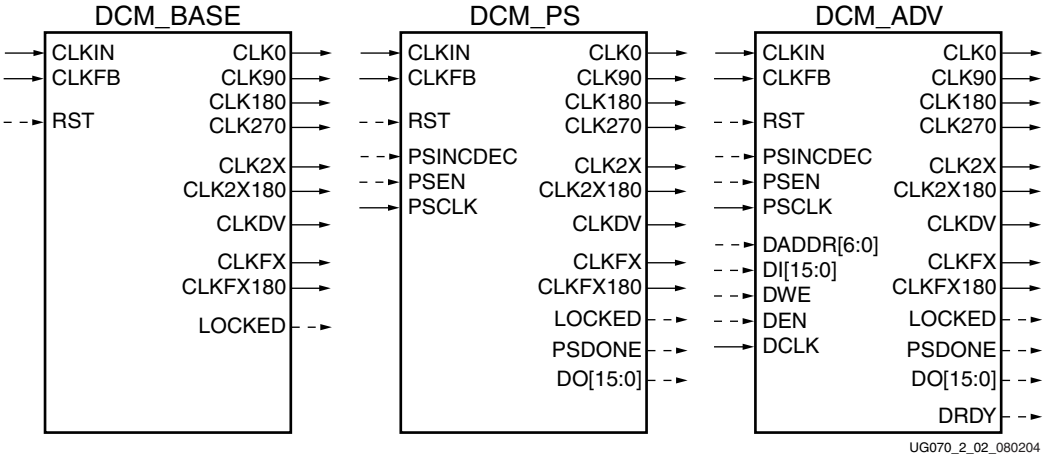


Figure 2-2: **DCM Primitives**

DCM_BASE Primitive

The DCM_BASE primitive accesses the basic frequently used DCM features and simplifies the user-interface ports. The clock deskew, frequency synthesis, and fixed-phase shifting features are available to use with DCM_BASE. Table 2-2 lists the available ports in the DCM_BASE primitive.

Table 2-2: **DCM_BASE Primitive**

Available Ports	Port Names
Clock Input	CLKIN, CLKFB
Control and Data Input	RST
Clock Output	CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKDV, CLKFX, CLKFX180
Status and Data Output	LOCKED

DCM_PS Primitive

The DCM_PS primitive accesses all DCM features and ports available in DCM_BASE plus additional ports used by the variable phase shifting feature. DCM_PS also has the following available DCM features: clock deskew, frequency synthesis, and fixed or variable phase-shifting. Table 2-3 lists the available ports in the DCM_PS primitive.

Table 2-3: **DCM_PS Primitive**

Available Ports	Port Names
Clock Input	CLKIN, CLKFB, PSCLK
Control and Data Input	RST, PSINCDEC, PSEN
Clock Output	CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKDV, CLKFX, CLKFX180
Status and Data Output	LOCKED, PSDONE, DO[15:0]

DCM_ADV Primitive

The DCM_ADV primitive has access to all DCM features and ports available in DCM_PS plus additional ports for the dynamic reconfiguration feature. It is a superset of the other two DCM primitives. DCM_ADV uses all the DCM features including clock deskew, frequency synthesis, fixed or variable phase shifting, and dynamic reconfiguration. Table 2-4 lists the available ports in the DCM_ADV primitive.

Table 2-4: **DCM_ADV Primitive**

Available Ports	Port Names
Clock Input	CLKIN, CLKFB, PSCLK, DCLK
Control and Data Input	RST, PSINCDEC, PSEN, DADDR[6:0], DI[15:0], DWE, DEN
Clock Output	CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKDV, CLKFX, CLKFX180
Status and Data Output	LOCKED, PSDONE, DO[15:0], DRDY

DCM Ports

There are four types of DCM ports available in the Virtex-4 architecture:

- Clock Input Ports
- Control and Data Input Ports
- Clock Output Ports
- Status and Data Output Ports

Clock Input Ports

Source Clock Input — CLKIN

The source clock (CLKIN) input pin provides the source clock to the DCM. The CLKIN frequency must fall in the ranges specified in the *Virtex-4 Data Sheet*. The clock input signal comes from one of the following buffers:

1. IBUFG – Global Clock Input Buffer

   The DCM compensates for the clock input path when an IBUFG on the same edge (top or bottom) of the device as the DCM is used.

2. BUFGCTRL – Internal Global Clock Buffer

   Any BUFGCTRL can drive any DCM in the Virtex-4 device using dedicated global routing. A BUFGCTRL can drive the DCM CLKIN pin when used to connect two DCMs in series.

3. IBUF – Input Buffer

   When an IBUF drives the CLKIN input, the PAD to DCM input skew is not compensated.

Feedback Clock Input — CLKFB

The feedback clock (CLKFB) input pin provides a reference or feedback signal to the DCM to delay-compensate the clock outputs, and align them with the clock input. To provide the necessary feedback to the DCM, connect only the CLK0 DCM output to the CLKFB pin. When the CLKFB pin is connected, all clock outputs are deskewed to CLKIN. When the CLKFB pin is not connected, DCM clock outputs are not deskewed to CLKIN. However, the relative phase relationship between all output clocks is preserved.

During internal feedback configuration, the CLK0 output of a DCM connects to a global buffer on the same top or bottom half of the device. The output of the global buffer connects to the CLKFB input of the same DCM.

During the external feedback configuration, the following rules apply:

1. To forward the clock, the CLK0 of the DCM must directly drive an OBUF or a BUFG-to-DDR configuration.
2. External to the FPGA, the forwarded clock signal must be connected to the IBUFG (GCLK pin) or the IBUF driving the CLKFB of the DCM. Both CLK and CLKFB should have identical I/O buffers.

Figure 2-9 and Figure 2-10, in "Application Examples," page 82, illustrate clock forwarding with external feedback configuration.

The feedback clock input signal can be driven by one of the following buffers:

1. IBUFG – Global Clock Input Buffer

   This is the preferred source for an external feedback configuration. When an IBUFG drives a CLKFB pin of a DCM in the same top or bottom half of the device, the pad to DCM skew is compensated for deskew.

2. BUFGCTRL – Internal Global Clock Buffer

   This is an internal feedback configuration.

3. IBUF – Input Buffer

   This is an external feedback configuration. When IBUF is used, the PAD to DCM input skew is not compensated.

Phase-Shift Clock Input — PSCLK

The phase-shift clock (PSCLK) input pin provides the source clock for the DCM phase shift. The PSCLK can be asynchronous (in phase and frequency) to CLKIN. The phase-shift clock signal can be driven by any clock source (external or internal), including:

1. IBUF – Input Buffer
2. IBUFG – Global Clock Input Buffer

   To access the dedicated routing, only the IBUFGs on the same edge of the device (top or bottom) as the DCM can be used to drive a PSCLK input of the DCM.

3. BUFGCTRL – An Internal Global Buffer
4. Internal Clock – Any internal clock using general purpose routing.

The frequency range of PSCLK is defined by PSCLK_FREQ_LF/HF (see the *Virtex-4 Data Sheet*). This input must be tied to ground when the CLKOUT_PHASE_SHIFT attribute is set to NONE or FIXED.

Dynamic Reconfiguration Clock Input — DCLK

The dynamic reconfiguration clock (DCLK) input pin provides the source clock for the DCM's dynamic reconfiguration circuit. The frequency of DCLK can be asynchronous (in phase and frequency) to CLKIN. The dynamic reconfiguration clock signal is driven by any clock source (external or internal), including:

1. IBUF – Input Buffer
2. IBUFG – Global Clock Input Buffer

   Only the IBUFGs on the same edge of the device (top or bottom) as the DCM can be used to drive a CLKIN input of the DCM.

3. BUFGCTRL – An Internal Global Buffer
4. Internal Clock – Any internal clock using general purpose routing.

The frequency range of DCLK is described in the *Virtex-4 Data Sheet*. When dynamic reconfiguration is not used, this input must be tied to ground. See the dynamic reconfiguration chapter in the *Virtex-4 Configuration Guide* for more information.

Control and Data Input Ports

Reset Input — RST

The reset (RST) input pin resets the DCM circuitry. The RST signal is an active High asynchronous reset. Asserting the RST signal asynchronously forces all DCM outputs Low (the LOCKED signal, all status signals, and all output clocks) after some propagation delay. When the reset is asserted, the last cycle of the clocks can exhibit a short pulse and a severely distorted duty-cycle, or no longer be deskewed with respect to one another while deasserting Low. Deasserting the RST signal starts the locking process at the next CLKIN cycle.

To ensure a proper DCM reset and locking process, the RST signal must be held until the CLKIN and CLKFB signals are present and stable for at least 200 ms. (The 200 ms requirement for CLKFB only applies when external feedback is used.)

The time it takes for the DCM to lock after a reset is specified in the *Virtex-4 Data Sheet* as LOCK_DLL (for a DLL output) and LOCK_FX (for a DFS output). These are the CLK and CLKFX outputs described in "Clock Output Ports". The DCM locks faster at higher frequencies. The worse-case numbers are specified in the *Virtex-4 Data Sheet*. In all designs, the DCM must be held in reset until CLKIN is stable.

Phase-Shift Increment/Decrement Input — PSINCDEC

The phase-shift increment/decrement (PSINCDEC) input signal must be synchronous with PSCLK. The PSINCDEC input signal is used to increment or decrement the phase-shift factor when PSEN is activated. As a result, the output clocks are shifted. The PSINCDEC signal is asserted High for increment or deasserted Low for decrement. This input must be tied to ground when the CLKOUT_PHASE_SHIFT attribute is set to NONE or FIXED.

Phase-Shift Enable Input — PSEN

The phase-shift enable (PSEN) input signal must be synchronous with PSCLK. A variable phase-shift operation is initiated by the PSEN input signal. It must be activated for one period of PSCLK. After PSEN is initiated, the phase change is gradual with completion indicated by a High pulse on PSDONE. There are no sporadic changes or glitches on any output during the phase transition. From the time PSEN is enabled until PSDONE is flagged, the DCM output clock moves bit-by-bit from its original phase shift to the target phase shift. The phase shift is complete when PSDONE is flagged. PSEN must be tied to ground when the CLKOUT_PHASE_SHIFT attribute is set to NONE or FIXED. Figure 2-7 shows the timing for this input.

Dynamic Reconfiguration Data Input — DI[15:0]

The dynamic reconfiguration data (DI) input bus provides reconfiguration data for dynamic reconfiguration. When not used, all bits must be assigned zeros. See the Dynamic Reconfiguration chapter of the *Virtex-4 Configuration Guide* for more information.

Dynamic Reconfiguration Address Input — DADDR[6:0]

The dynamic reconfiguration address (DADDR) input bus provides a reconfiguration address for the dynamic reconfiguration. When not used, all bits must be assigned zeros. The DO output bus will reflect the DCM's status. See the Dynamic Reconfiguration chapter of the *Virtex-4 Configuration Guide* for more information.

Dynamic Reconfiguration Write Enable Input — DWE

The dynamic reconfiguration write enable (DWE) input pin provides the write enable control signal to write the DI data into the DADDR address. When not used, it must be tied Low. See the Dynamic Reconfiguration chapter of the *Virtex-4 Configuration Guide* for more information.

Dynamic Reconfiguration Enable Input — DEN

The dynamic reconfiguration enable (DEN) input pin provides the enable control signal to access the dynamic reconfiguration feature. When the dynamic reconfiguration feature is not used, DEN must be tied Low. When DEN is tied Low, DO reflects the DCM status signals. See the Dynamic Reconfiguration chapter of the *Virtex-4 Configuration Guide* for more information.

Clock Output Ports

A DCM provides nine clock outputs with specific frequency and phase relationships. When CLKFB is connected, all DCM clock outputs have a fixed phase relationship to CLKIN. When CLKFB is not connected, the DCM outputs are not phase aligned. However, the phase relationship between all output clocks is preserved.

1x Output Clock — CLK0

The CLK0 output clock provides a clock with the same frequency as the DCM's effective CLKIN frequency. By default, the effective input clock frequency is equal to the CLKIN frequency. The CLKIN_DIVIDE_BY_2 attribute is set to TRUE to make the effective CLKIN frequency ½ the actual CLKIN frequency. The CLKIN_DIVIDE_BY_2 Attribute description provides further information. When CLKFB is connected, CLK0 is phase aligned to CLKIN.

1x Output Clock, 90° Phase Shift — CLK90

The CLK90 output clock provides a clock with the same frequency as the DCM's CLK0 only phase-shifted by 90°.

1x Output Clock, 180° Phase Shift — CLK180

The CLK180 output clock provides a clock with the same frequency as the DCM's CLK0 only phase-shifted by 180°.

1x Output Clock, 270° Phase Shift — CLK270

The CLK270 output clock provides a clock with the same frequency as the DCM's CLK0 only phase-shifted by 270°.

2x Output Clock — CLK2X

The CLK2X output clock provides a clock that is phase aligned to CLK0, with twice the CLK0 frequency, and with an automatic 50/50 duty-cycle correction. Until the DCM is locked, the CLK2X output appears as a 1x version of the input clock with a 25/75 duty cycle. This behavior allows the DCM to lock on the correct edge with respect to the source clock.

2x Output Clock, 180° Phase Shift — CLK2X180

The CLK2X180 output clock provides a clock with the same frequency as the DCM's CLK2X only phase-shifted by 180°.

Frequency Divide Output Clock — CLKDV

The CLKDV output clock provides a clock that is phase aligned to CLK0 with a frequency that is a fraction of the effective CLKIN frequency. The fraction is determined by the CLKDV_DIVIDE attribute. Refer to the CLKDV_DIVIDE Attribute for more information.

Frequency-Synthesis Output Clock — CLKFX

The CLKFX output clock provides a clock with the following frequency definition:

CLKFX frequency = (M/D) × effective CLKIN frequency

In this equation, M is the multiplier (numerator) with a value defined by the CLKFX_MULTIPLY attribute. D is the divisor (denominator) with a value defined by the CLKFX_DIVIDE attribute. Specifications for M and D, as well as input and output frequency ranges for the frequency synthesizer, are provided in the *Virtex-4 Data Sheet*.

The rising edge of CLKFX output is phase aligned to the rising edges of CLK0, CLK2X, and CLKDV. When M and D to have no common factor, the alignment occurs only once every D cycles of CLK0.

Frequency-Synthesis Output Clock, 180° — CLKFX180

The CLKFX180 output clock provides a clock with the same frequency as the DCM's CLKFX only phase-shifted by 180°.

Status and Data Output Ports

Locked Output — LOCKED

The LOCKED output indicates whether the DCM clock outputs are valid, i.e., the outputs exhibit the proper frequency and phase. After a reset, the DCM samples several thousand clock cycles to achieve lock. After the DCM achieves lock, the LOCKED signal is asserted High. The DCM timing parameters section of the *Virtex-4 Data Sheet* provides estimates for locking times.

To guarantee an established system clock at the end of the start-up cycle, the DCM can delay the completion of the device configuration process until after the DCM is locked. The STARTUP_WAIT attribute activates this feature. The STARTUP_WAIT Attribute description provides further information.

Until the LOCKED signal is asserted High, the DCM output clocks are not valid and can exhibit glitches, spikes, or other spurious movement. In particular, the CLK2X output appears as a 1x clock with a 25/75 duty cycle.

Phase-Shift Done Output — PSDONE

The phase-shift done (PSDONE) output signal is synchronous to PSCLK. At the completion of the requested phase shift, PSDONE pulses High for one period of PSCLK. This signal also indicates a new change to the phase shift can be initiated. The PSDONE output signal is not valid if the phase-shift feature is not being used or is in fixed mode.

Status or Dynamic Reconfiguration Data Output — DO[15:0]

The DO output bus provides DCM status or data output when using dynamic reconfiguration (Table 2-5). Further information on using DO as the data output is available in the Dynamic Reconfiguration chapter of the *Virtex-4 Configuration Guide* for more information.

If the dynamic reconfiguration port is not used, using DCM_BASE or DCM_PS instead of DCM_ADV is strongly recommended.

Table 2-5: **DCM Status Mapping to DO Bus**

DO Bit	Status	Description
DO[0]	Phase-shift overflow	Asserted when the DCM is phase-shifted beyond the allowed phase-shift value or when the absolute delay range of the phase-shift delay line is exceeded.
DO[1]	CLKIN stopped	Asserted when the input clock is stopped (CLKIN remains High or Low for one or more clock cycles). When CLKIN is stopped, the DO[1] CLKIN stopped status is asserted within nine CLKIN cycles. When CLKIN is restarted, CLK0 starts toggling and DO[1] is deasserted within nine clock cycles.
DO[2]	CLKFX stopped	Asserted when CLKFX stops. The DO[2] CLKFX stopped status is asserted within 257 to 260 CLKIN cycles after CLKFX stopped. CLKFX will not resume, and DO[2] is not deasserted until the DCM is reset.
DO[3]	CLKFB stopped	Asserted when the feedback clock is stopped (CLKFB remains High or Low for one or more clock cycles). The DO[3] CLKFB stopped status is asserted within six CLKIN cycles after CLKFB is stopped. CLKFB stopped is deasserted within six CLKIN cycles when CLKFB resumes after being stopped momentarily. An occasionally skipped CLKFB will not affect the DCM operation. However, stopping CLKFB for a long time can result in the DCM losing LOCKED. When LOCKED is lost, the DCM needs to be reset to resume operation.
DO[15:4]	Not assigned	

When LOCKED is Low (during reset or the locking process), all the status signals deassert Low.

Dynamic Reconfiguration Ready Output — DRDY

The dynamic reconfiguration ready (DRDY) output pin provides the response to the DEN signal for the DCM's dynamic reconfiguration feature. Further information on the DRDY pin is available in the dynamic reconfiguration section in the *Virtex-4 Configuration Guide*.

DCM Attributes

A handful of DCM attributes govern the DCM functionality. Table 2-6 summarizes all the applicable DCM attributes. This section provides a detailed description of each attribute. For more information on applying these attributes in UCF, VHDL, or Verilog code, refer to the Constraints Guide at:

http://www.support.xilinx.com/support/software_manuals.htm

Table 2-6: **DCM Attributes**

DCM Attribute Name	Description	Values	Default Value
CLK_FEEDBACK	Determines the type of feedback applied to CLKFB.	String: “1X” or “NONE”	1X
CLKDV_DIVIDE	Controls CLKDV such that the source clock is divided by N. This feature provides automatic duty cycle correction such that the CLKDV output pin has a 50/50 duty cycle always in low-frequency mode, as well as for all integer values of the division factor N in high-frequency mode.	Real: 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8, 9, 10, 11, 12, 13, 14, 15, 16	2.0
CLKFX_DIVIDE	Sets the divisor (D) value of CLKFX. The CLKFX frequency equals the effective CLKIN frequency multiplied by M/D.	Integer: 1 to 32	1
CLKFX_MULTIPLY	Sets the multiply (M) of CLKFX. The CLKFX frequency equals the effective CLKIN frequency multiplied by M/D.	Integer: 2 to 32	4
CLKIN_DIVIDE_BY_2	Allows for the input clock frequency to be divided in half when necessary to meet the DCM input clock frequency requirements.	Boolean: FALSE or TRUE	FALSE
CLKIN_PERIOD	Specifies the source clock period to help the DCM adjust for optimum CLKFX/CLKFX180 outputs.	Real in ns	0.0
CLKOUT_PHASE_SHIFT	Specifies the phase-shift mode.	String: “NONE”, “FIXED”, “VARIABLE_POSITIVE”, “VARIABLE_CENTER”, or “DIRECT”	NONE

Table 2-6: **DCM Attributes *(Continued)***

DCM Attribute Name	Description	Values	Default Value
DCM_AUTOCALIBRATION	When this attribute is TRUE, the DCM is protected from the effects of negative bias temperature instability (NBTI). This attribute cannot be set to FALSE unless CLKIN and CLKFB (if external feedback is used) are guaranteed to never stop. The macro can also be disabled if the user can guarantee to hold DCM in reset during clock stoppage. If this attribute is set to FALSE, the reset requirement is three clock cycles.	Boolean: TRUE or FALSE	TRUE
DCM_PERFORMANCE_MODE	Allows selection between maximum frequency/ minimum jitter and low frequency/maximum phase-shift range.	String: “MAX_SPEED” or “MAX_RANGE”	MAX_SPEED
DESKEW_ADJUST	Affects the amount of delay in the feedback path, and should be used for source-synchronous interfaces.	String: “SYSTEM_SYNCHRONOUS” or “SOURCE_SYNCHRONOUS”	SYSTEM_ SYNCHRONOUS
DFS_FREQUENCY_MODE	Specifies the frequency mode of the frequency synthesizer.	String: “LOW” or “HIGH”	LOW
DLL_FREQUENCY_MODE	Specifies the frequency mode of the DLL.	String: “LOW” or “HIGH”	LOW
DUTY_CYCLE_CORRECTION	Controls the DCM 1X outputs (CLK0, CLK90, CLK180, and CLK270), to exhibit a 50/50 duty cycle. Leave this attribute set at the default value.	Boolean: TRUE or FALSE	TRUE
FACTORY_JF	Controls the DCM tap update rate. Value depends on DLL_FREQUENCY_MODE setting.	BIT_VECTOR	`F0F0`
PHASE_SHIFT	Specifies the phase-shift numerator. The value range depends on CLKOUT_PHASE_SHIFT and clock frequency.	Integer: –255 to 255 or 0 to 1023	0
STARTUP_WAIT	When this attribute is set to TRUE, the configuration startup sequence waits in the specified cycle until the DCM locks.	Boolean: FALSE or TRUE	FALSE

CLK_FEEDBACK Attribute

The CLK_FEEDBACK attribute determines the type of feedback applied to the CLKFB. The possible values are 1X or NONE. The default value is 1X. When this attribute is set to 1X, the CLKFB pin must be driven by CLK0. When this attribute is set to NONE, the CLKFB pin must be unconnected.

CLKDV_DIVIDE Attribute

The CLKDV_DIVIDE attribute controls the CLKDV frequency. The source clock frequency is divided by the value of this attribute. The possible values for CLKDV_DIVIDE are: 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 9, 10, 11, 12, 13, 14, 15, or 16. The default value is 2. In the low frequency mode, any CLKDV_DIVIDE value produces a CLKDV output with a 50/50 duty-cycle. In the high frequency mode, the CLKDV_DIVIDE value must be set to an integer value to produce a CLKDV output with a 50/50 duty-cycle. For non-integer CLKDV_DIVIDE values, the CLKDV output duty cycle is shown in Table 2-7.

Table 2-7: **Non-Integer CLKDV_DIVIDE**

CLKDV_DIVIDE Value	CLKDV Duty Cycle in High Frequency Mode (High Pulse/Low Pulse Value)
1.5	1/3
2.5	2/5
3.5	3/7
4.5	4/9
5.5	5/11
6.5	6/13
7.5	7/15

CLKFX_MULTIPLY and CLKFX_DIVIDE Attributes

The CLKFX_MULTIPLY attribute sets the multiply value (M) of the CLKFX output. The CLKFX_DIVIDE attribute sets the divisor (D) value of the CLKFX output. Both control the CLKFX output making the CLKFX frequency equal the effective CLKIN (source clock) frequency multiplied by M/D. The possible values for M are any integer from 2 to 32. The possible values for D are any integer from 1 to 32. The default settings are M = 4 and D = 1.

CLKIN_DIVIDE_BY_2 Attribute

The CLKIN_DIVIDE_BY_2 attribute is used to enable a toggle flip-flop in the input clock path to the DCM. When set to FALSE, the effective CLKIN frequency of the DCM equals the source clock frequency driving the CLKIN input. When set to TRUE, the CLKIN frequency is divided by two before it reaches the rest of the DCM. Thus, the DCM sees half the frequency applied to the CLKIN input and operates based on this frequency. For example, if a 100 MHz clock drives CLKIN, and CLKIN_DIVIDE_BY_2 is set to TRUE; then the effective CLKIN frequency is 50 MHz. Thus, CLK0 output is 50 MHz and CLK2X output is 100 MHz. The effective CLKIN frequency must be used to evaluate any operation or specification derived from CLKIN frequency. The possible values for CLKIN_DIVIDE_BY_2 are TRUE and FALSE. The default value is FALSE.

CLKIN_PERIOD Attribute

The CLKIN_PERIOD attribute specifies the source clock period (in nanoseconds). The default value is 0.0 ns.

CLKOUT_PHASE_SHIFT Attribute

The CLKOUT_PHASE_SHIFT attribute indicates the mode of the phase shift applied to the DCM outputs. The possible values are NONE, FIXED, VARIABLE_POSITIVE, VARIABLE_CENTER, or DIRECT. The default value is NONE.

When set to NONE, a phase shift cannot be performed and a phase-shift value has no effect on the DCM outputs. When set to FIXED, the DCM outputs are phase-shifted by a fixed phase from the CLKIN. The phase-shift value is determined by the PHASE_SHIFT attribute. If the CLKOUT_PHASE_SHIFT attribute is set to FIXED or NONE, then the PSEN, PSINCDEC, and the PSCLK inputs must be tied to ground.

When set to VARIABLE_POSITIVE, the DCM outputs can be phase-shifted in variable mode in the positive range with respect to CLKIN. When set to VARIABLE_CENTER, the DCM outputs can be phase-shifted in variable mode, in the positive and negative range with respect to CLKIN. If set to VARIABLE_POSITIVE or VARIABLE_CENTER, each phase-shift increment (or decrement) will increase (or decrease) the phase shift by a period of 1/256 x CLKIN period.

When set to DIRECT, the DCM output can be phase-shifted in variable mode in the positive range with respect to CLKIN. Each phase-shift increment/decrement will increase/decrease the phase shift by one DCM_TAP (see the *Virtex-4 Data Sheet*).

The starting phase in the VARIABLE_POSITIVE and VARIABLE_CENTER modes is determined by the phase-shift value. The starting phase in the DIRECT mode is always zero, regardless of the value specified by the PHASE_SHIFT attribute. Thus, the PHASE_SHIFT attribute should be set to zero when DIRECT mode is used. A non-zero phase-shift value for DIRECT mode can be loaded to the DCM using Dynamic Reconfiguration Ports in the *Virtex-4 Configuration Guide*.

DCM_AUTOCALIBRATION Attribute

The autocalibration block protects the DCM from the effects of negative bias temperature instability (NBTI). This attribute cannot be set to FALSE unless the user guarantees that CLKIN and CLKFB (if external feedback is used) never stop. The macro can also be disabled if the user can guarantee that DCM is held in reset when the clocks are stopped. If this attribute is set to FALSE, the reset requirement is three clock cycles.

DCM_PERFORMANCE_MODE Attribute

The DCM_PERFORMANCE_MODE attribute allows the choice of optimizing the DCM either for high frequency and low jitter or for low frequency and a wide phase-shift range. The attribute values are MAX_SPEED and MAX_RANGE. The default value is MAX_SPEED. When set to MAX_SPEED, the DCM is optimized to produce high frequency clocks with low jitter. However, the phase-shift range is smaller than when MAX_RANGE is selected. When set to MAX_RANGE, the DCM is optimized to produce low frequency clocks with a wider phase-shift range. The DCM_PERFORMANCE_MODE affects the following specifications: DCM input and output frequency range, phase-shift range, output jitter, DCM_TAP, CLKIN_CLKFB_PHASE, CLKOUT_PHASE, and duty-cycle precision. The *Virtex-4 Data Sheet* specifies these values.

For most cases, the DCM_PERFORMANCE_MODE attribute should be set to MAX_SPEED (default). Consider changing to MAX_RANGE only in these situations:

- The frequency needs to be below the low frequency limit of the MAX_SPEED setting.
- A greater absolute phase-shift range is required.

DESKEW_ADJUST Attribute

The DESKEW_ADJUST attribute affects the amount of delay in the feedback path. The possible values are SYSTEM_SYNCHRONOUS, SOURCE_SYNCHRONOUS, 0, 1, 2, 3, ..., or 31. The default value is SYSTEM_SYNCHRONOUS.

For most designs, the default value is appropriate. In a source-synchronous design, set this attribute to SOURCE_SYNCHRONOUS. The remaining values should only be used after consulting with Xilinx. For more information consult the "Source-Synchronous Setting"section.

DFS_FREQUENCY_MODE Attribute

The DFS_FREQUENCY_MODE attribute specifies the frequency mode of the digital frequency synthesizer (DFS). The possible values are LOW and HIGH. The default value is LOW. The frequency ranges for both frequency modes are specified in the *Virtex-4 Data Sheet*. DFS_FREQUENCY_MODE determines the frequency range of CLKIN, CLKFX, and CLKFX180.

DLL_FREQUENCY_MODE Attribute

The DLL_FREQUENCY_MODE attribute specifies either the HIGH or LOW frequency mode of the delay-locked loop (DLL). The default value is LOW. The frequency ranges for both frequency modes are specified in the *Virtex-4 Data Sheet*.

DUTY_CYCLE_CORRECTION Attribute

The DUTY_CYCLE_CORRECTION attribute controls the duty cycle correction of the 1x clock outputs: CLK0, CLK90, CLK180, and CLK270. The possible values are TRUE and FALSE. The default value is TRUE. When set to TRUE, the 1x clock outputs are duty cycle corrected to be within specified limits (see the *Virtex-4 Data Sheet* for details). It is strongly recommended to always set the DUTY_CYCLE_CORRECTION attribute to TRUE. Setting this attribute to FALSE does not necessarily produce output clocks with the same duty cycle as the source clock.

FACTORY_JF Attribute

The Factory_JF attribute affects the DCMs jitter filter characteristics. This attribute controls the DCM tap update rate. Factory_JF must be set to a specific value depending on the DLL_FREQUENCY_MODE setting. The default value is `F0F0` corresponding to DLL_FREQUENCY_MODE = LOW (default). Factory_JF must be manually set to `F0F0` when DLL_FREQUENCY_MODE = HIGH. The ISE® software tool will issue a warning if FACTORY_JF is not set as stated.

PHASE_SHIFT Attribute

The PHASE_SHIFT attribute determines the amount of phase shift applied to the DCM outputs. This attribute can be used in both fixed or variable phase-shift mode. If used with variable mode, the attribute sets the starting phase shift. When CLKOUT_PHASE_SHIFT = VARIABLE_POSITIVE, the PHASE_SHIFT value range is 0 to 255. When CLKOUT_PHASE_SHIFT = VARIABLE_CENTER or FIXED, the PHASE_SHIFT value range is –255 to 255. When CLKOUT_PHASE_SHIFT = DIRECT, the PHASE_SHIFT value range is 0 to 1023. The default value is 0.

Refer to "Phase Shifting," page 76 for information on the phase-shifting operation and its relationship with the CLKOUT_PHASE_SHIFT and PHASE_SHIFT attributes.

STARTUP_WAIT Attribute

The STARTUP_WAIT attribute determines whether the DCM waits in one of the startup cycles for the DCM to lock. The possible values for this attribute are TRUE and FALSE. The default value is FALSE. When STARTUP_WAIT is set to TRUE, and the LCK_cycle BitGen option is used, then the configuration startup sequence waits in the startup cycle specified by LCK_cycle until the DCM is locked.

DCM Design Guidelines

This section provides a detailed guidelines on using the Virtex-4 FPGA DCM.

Clock Deskew

The Virtex-4 FPGA DCM offers a fully digital, dedicated, on-chip clock deskew. The deskew feature provides zero propagation delay between the source clock and output clock, low clock skew among output clock signals distributed throughout the device, and advanced clock domain control.

The deskew feature also functions as a clock mirror of a board-level clock serving multiple devices. This is achieved by driving the CLK0 output off-chip to the board (and to other devices on the board) and then bringing the clock back in as a feedback clock. See the "Application Examples" section. Taking advantage of the deskew feature greatly simplifies and improves system-level design involving high-fanout, high-performance clocks.

Clock Deskew Operation

The deskew feature utilizes the DLL circuit in the DCM. In its simplest form, the DLL consists of a single variable delay line (containing individual small delay elements or buffers) and control logic. The incoming clock drives the delay line. The output of every delay element represents a version of the incoming clock (CLKIN) delayed at a different point. The clock distribution network routes the clock to all internal registers and to the clock feedback CLKFB pin. The control logic contains a phase detector and a delay-line selector. The phase detector compares the incoming clock signal (CLKIN) against a feedback input (CLKFB) and steers the delay-line selector, essentially adding delay to the DCM output until the CLKIN and CLKFB coincide, putting the two clocks 360° out-of-phase, (thus, in phase). When the edges from the input clock line up with the edges from the feedback clock, the DCM achieves a lock. The two clocks have no discernible difference. Thus, the DCM output clock compensates for the delay in the clock distribution network, effectively removing the delay between the source clock and its loads. The size of each intrinsic delay element is a DCM_TAP (see the AC Characteristics table in the *Virtex-4 Data Sheet*). Figure 2-3 illustrates a simplified DLL circuit.

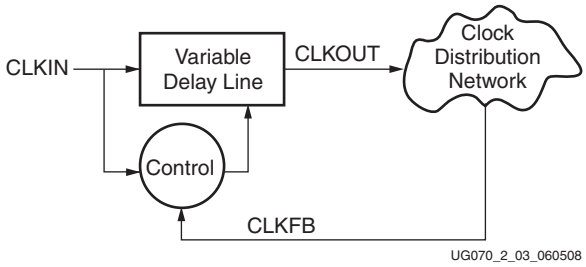


Figure 2-3: **Simplified DLL Circuit**

To provide the correct clock deskew, the DCM depends on the dedicated routing and resources used at the clock source and feedback input. An additional delay element (see "Deskew Adjust") is available to compensate for the clock source or feedback path. The ISE tools analyze the routing around the DCM to determine if a delay must be inserted to compensate for the clock source or feedback path. Thus, using dedicated routing is required to achieve predictable deskew. All nine DCM output clocks are deskewed when the CLKFB pin is used.

Input Clock Requirements

The clock input of the DCM can be driven either by an IBUFG/IBUFGDS, IBUF, BUFGMUX, or a BUFGCNTL. Since there is no dedicated routing between an IBUF and a DCM clock input, using an IBUF causes additional input delay that is not compensated by the DCM.

The DCM output clock signal is essentially a delayed version of the input clock signal. It reflects any instability on the input clock in the output waveform. The DCM input clock requirements are specified in the *Virtex-4 Data Sheet*.

Once locked, the DCM can tolerate input clock period variations of up to the value specified by CLKIN_PER_JITT_DLL_HF (at high frequencies) or CLKIN_PER_JITT_DLL_LF (at low frequencies). Larger jitter (period changes) can cause the DCM to lose lock, indicated by the LOCKED output deasserting. The user must then reset the DCM. The cycle-to-cycle input jitter must be kept to less than CLKIN_CYC_JITT_DLL_LF in the low frequencies and CLKIN_CYC_JITT_DLL_HF for the high frequencies.

Input Clock Changes

Changing the period of the input clock beyond the maximum input period jitter specification requires a manual reset of the DCM. Failure to reset the DCM produces an unreliable LOCKED signal and output clock. It is possible to temporarily stop the input clock and feedback clock with little impact to the deskew circuit, as long as CLKFX or CLKFX180 is not used.

If the input clock is stopped and CLKFX or CLKFX180 is used, the CLKFX or CLKFX180 outputs might stop toggling, and DO[2] (CLKFX Stopped) is asserted. The DCM must be reset to recover from this event.

The DO[2] CLKFX stopped status is asserted in 257 to 260 CLKIN cycles after CLKFX is stopped. CLKFX does not resume and DO[2] will not deassert until the DCM is reset.

In any other case, the clock should not be stopped for more than 100 ms to minimize the effect of device cooling; otherwise, the tap delays might change. The clock should be stopped during a Low or a High phase, and must be restored with the same input clock period/frequency. During this time, LOCKED stays High and remains High when the

clock is restored. Thus, a High on LOCKED does not necessarily mean that a valid clock is available.

When stopping the input clock (CLKIN remains High or Low for one or more clock cycles), one to nine more output clock cycles are still generated as the delay line is flushed. When the output clock stops, the CLKIN stopped (DO[1]) signal is asserted. When the clock is restarted, the output clock cycles are not generated for one to eight clocks while the delay line is filled. Similarly, the DO[1] signal is deasserted once the output clock is generated. The most common case is two or three clocks. CLKIN can be restarted with any phase relationship to the previous clock. If the frequency has changed, the DCM requires a reset. The DO[1] is forced Low whenever LOCKED is Low. When the DCM is in the locking process, DO[1] status is held Low until LOCKED is achieved.

Output Clocks

Any or all of the DCM's nine clock outputs can be used to drive a global clock network. The fully-buffered global clock distribution network minimizes clock skew caused by loading differences. By monitoring a sample of the output clock (CLK0), the deskew circuit compensates for the delay on the routing network, effectively eliminating the delay from the external input port to the individual clock loads within the device.

All DCM outputs can drive general interconnect; however, these connections are not suitable for critical clock signals. It is recommended that all clock signals should be within the global or regional clock network. Refer to Chapter 1, "Clock Resources" for more information on using clock networks.

Output pin connectivity carries some restrictions. The DCM clock outputs can each drive an OBUF, a global clock buffer BUFGCTRL, or they can route directly to the clock input of a synchronous element. To use dedicated routing, the DCM clock outputs must drive BUFGCTRLs on the same top or bottom half of the device. If the DCM and BUFGCTRL are not on the same top or bottom half, local routing is used and the DCM might not deskew properly.

Do not use the DCM output clock signals until after activation of the LOCKED signal. Prior to the activation of the LOCKED signal, the DCM output clocks are not valid.

DCM During Configuration and Startup

During the FPGA configuration, the DCM is in reset and starts to lock at the beginning of the startup sequence. A DCM requires both CLKIN and CLKFB input clocks to be present and stable when the DCM begins to lock. If the device enters the configuration startup sequence without an input clock, or with an unstable input clock, then the DCM must be reset after configuration with a stable clock.

The following startup cycle dependencies are of note:

1. The default value is **-g LCK_cycle:NoWait**. When this setting is used, the startup sequence does not wait for the DCM to lock. WHen the LCK_cycle is set to other values, the configuration startup remains in the specified startup cycle until the DCM is locked.
2. Before setting the **LCK_cycle** option to a startup cycle in BitGen, the DCM's STARTUP_WAIT attribute must be set to TRUE.
3. If the startup sequence is altered (by using the BitGen option), do not place the **LCK_cycle** (wait for the DCM to lock) before the **GTS_cycle** (deassert GTS). Incorrect implementation will result in the DCM not locking and an incomplete configuration.

Deskew Adjust

The DESKEW_ADJUST attribute sets the value for a configurable, variable-tap delay element to control the amount of delay added to the DCM feedback path (see Figure 2-4).

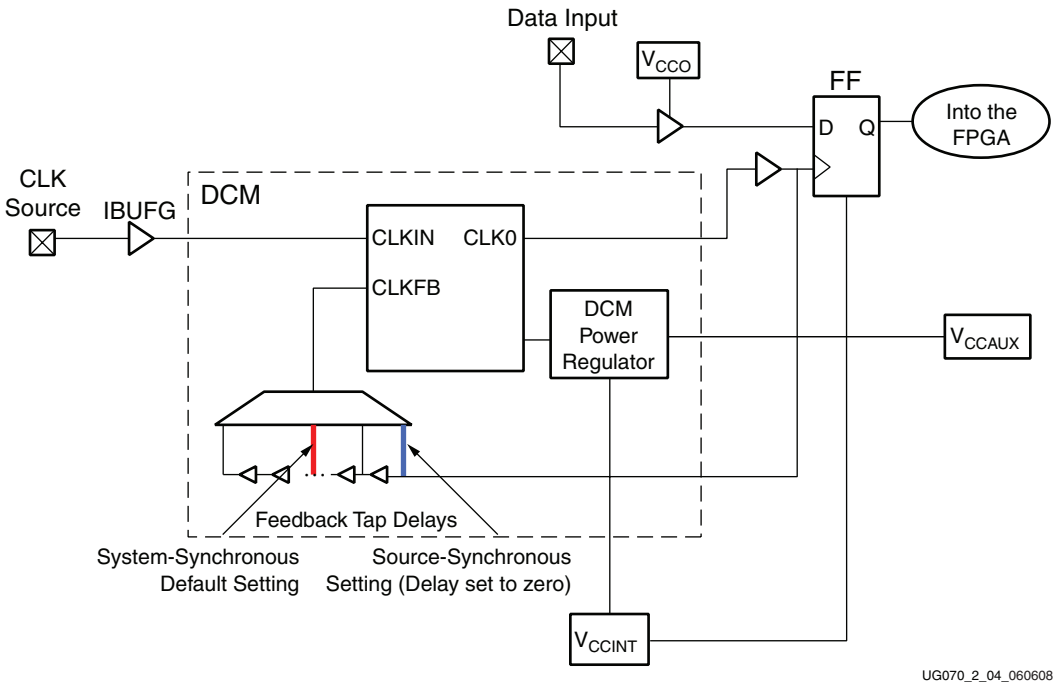


Figure 2-4: **DCM and Feedback Tap-Delay Elements**

This delay element allows adjustment of the effective clock delay between the clock source and CLK0 to guarantee non-positive hold times of IOB input flip-flop in the device. Adding more delay to the DCM feedback path decreases the effective delay of the actual clock path from the FPGA clock input pin to the clock input of any flip-flop. Decreasing the clock delay increases the setup time represented in the input flip-flop, and reduces any positive hold times required. The clock path delay includes the delay through the IBUFG, route, DCM, BUFG, and clock-tree to the destination flip-flop. If the feedback delay equals the clock-path delay, the effective clock-path delay is zero.

System-Synchronous Setting (Default)

By default, the feedback delay is set to system-synchronous mode. The primary timing requirements for a system-synchronous system are non-positive hold times (or minimally positive hold times) and minimal clock-to-out and setup times. Faster clock-to-out and setup times allow shorter system clock periods. Ideally, the purpose of a DLL is to zero-out the clock delay to produce faster clock-to-out and non-positive hold times. The system-synchronous setting (default) for DESKEW_ADJUST configures the feedback delay element to guarantee non-positive hold times for all input IOB registers. The exact delay number added to the feedback path is device size dependent. This is determined by characterization. In the timing report, this is included as timing reduction to input clock path represented by the T_{DCMINO} parameter. As shown in Figure 2-4, the feedback path includes tap delays in the default setting (red line). The pin-to-pin timing parameters (with DCM) on the *Virtex-4 Data Sheet* reflects the setup/hold and clock-to-out times when the DCM is in system-synchronous mode.

In some situations, the DCM does not add this extra delay, and the DESKEW_ADJUST parameter has no effect. BitGen selects the appropriate DCM tap settings. These situations include:

- downstream DCMs when two or more DCMs are cascaded
- DCMs with external feedback
- DCMs with an external CLKIN that does not come from a dedicated clock input pin

Source-Synchronous Setting

When DESKEW_ADJUST is set to source-synchronous mode, the DCM feedback delay element is set to zero. As shown in Figure 2-4, in source-synchronous mode, the DCM clock feedback delay element is set to minimize the sampling window. This results in a more positive hold time and a longer clock-to-out compared to system-synchronous mode. The source-synchronous switching characteristics section in the *Virtex-4 Data Sheet* reflects the various timing parameters for the source-synchronous design when the DCM is in source-synchronous mode.

Characteristics of the Deskew Circuit

- Eliminate clock distribution delay by effectively adding one clock period delay. Clocks are deskewed to within CLKOUT_PHASE, specified in the *Virtex-4 Data Sheet*.
- Eliminate on-chip as well as off-chip clock delay.
- No restrictions on the delay in the feedback clock path.
- Requires a continuously running input clock.
- Adapts to a wide range of frequencies. However, once locked to a frequency, large input frequency variations are not tolerated.
- Does not eliminate jitter. The deskew circuit output jitter is the accumulation of input jitter and any added jitter value due to the deskew circuit.
- The completion of configuration can be delayed until after DCM locks to guarantee the system clock is established prior to initiating the device.

Cascading DCMs

Xilinx does not recommend cascading DCMs because jitter accumulates as a result—in other words, the output clock jitter of the second-stage DCM is worse than the output clock jitter of the first-stage DCM. If possible, use two DCMs in parallel instead of in series.

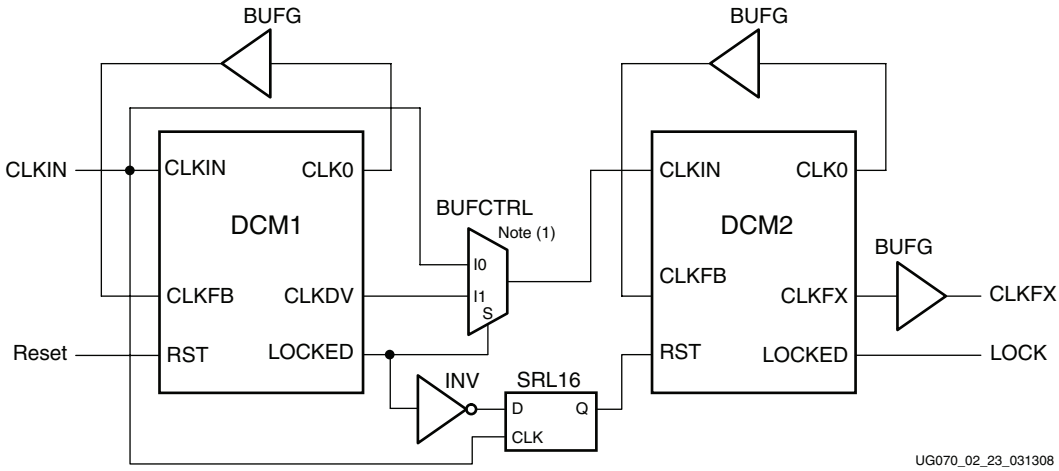
If it is absolutely necessary to cascade DCMs, the following rules *must* be observed:

- The output jitter specifications for DLL outputs are provided in the data sheet. Use the Jitter Calculator to determine the jitter for CLKFX. If possible, avoid cascading CLKFX to CLKFX in high-frequency mode. In general, jitter accumulates based on the following equation:

$$\text{Total Jitter} = \sqrt{(\text{Jitter1})^2 + (\text{Jitter2})^2}$$

- The input and output frequency and jitter specifications for each DCM must be met. If the frequency of the DCM inputs allows it, use feedback for both DCMs.
- Use the LOCKED output from DCM1 to create a Reset for DCM2. The recommended length of a Reset pulse is 200ms. The LOCKED signal from DCM1 should be inverted and provide the Reset input to DCM2. Connect the output of DCM1 to CLKIN of DCM2 through a BUFGCTRL. CLKIN and the DCM output clock (CLKDV in this case) feed a BUFGCTRL acting as an asynchronous mux. When DCM1 is in reset and

while acquiring LOCK the CLKIN clock feeds DCM2. After the DCM1 locks the DCM1 output clock feeds DCM2. DCM2 is held in reset for 16 additional CLKIN cycles. Figure 2-5 illustrates this approach.



1. This is an asynchronous clock mux as shown in Figure 1-13, page 36.

Figure 2-5: **Cascading DCMs**

- It is recommended that R1 > R2, where:

  R1 = M/D ratio for DCM1
  R2 = M/D ratio for DCM2

  The ranges of M and D values are given in the data sheet.

Frequency Synthesis

The DCM provides several flexible methods for generating new clock frequencies. Each method has a different operating frequency range and different AC characteristics. The CLK2X and CLK2X180 outputs double the clock frequency. The CLKDV output provides a divided output clock (lower frequency) with division options of 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 9, 10, 11, 12, 13, 14, 15, and 16.

The DCM also offers fully digital, dedicated frequency-synthesizer outputs CLKFX and its opposite phase CLKFX180. The output frequency can be any function of the input clock frequency described by M ÷ D, where M is the multiplier (numerator) and D is the divisor (denominator).

The frequency synthesized outputs can drive the global-clock routing networks within the device. The well-buffered global-clock distribution network minimizes clock skew due to differences in distance or loading.

Frequency Synthesis Operation

The DCM clock output CLKFX is any M ÷ D factor of the clock input to the DCM. Specifications for M and D, as well as input and output frequency ranges for the frequency synthesizer, are provided in the *Virtex-4 Data Sheet*.

Only when feedback is provided to the CLKFB input of the DCM is the frequency synthesizer output phase aligned to the clock output, CLK0.

The internal operation of the frequency synthesizer is complex and beyond the scope of this document. As long as the frequency synthesizer is within the range specified in the

Virtex-4 Data Sheet, it multiplies the incoming frequencies by the pre-calculated quotient M ÷ D and generates the correct output frequencies.

For example, assume an input frequency of 50 MHz, M = 25, and D = 8 (M and D values do not have common factors and cannot be reduced). The output frequency is 156.25 MHz although separate calculations, 25 x 50 MHz = 1.25 GHz and 50 MHz ÷ 8 = 6.25 MHz, seem to produce separate values outside the range of the input frequency.

Frequency Synthesizer Characteristics

- The frequency synthesizer provides an output frequency equal to the input frequency multiplied by M and divided by D.
- The outputs CLKFX and CLKFX180 always have a 50/50 duty-cycle.
- Smaller M and D values achieve faster lock times. Whenever possible, divide M and D by the largest common factor to get the smallest values. (e.g., if the required CLKFX = 9/6 x CLKIN, instead of using M = 9 and D = 6, use M = 3 and D = 2.)
- When CLKFB is connected, CLKFX is phase aligned with CLK0 every D cycles of CLK0 and every M cycles of CLKFX if M/D is a reduced fraction.
- In the case where only the DFS outputs are used (CLKFB is not connected) and the CLKIN of the DCM is outside the range of the DLL outputs, the DCM_AUTOCALIBRATION attribute must be set to FALSE and the CONFIG STEPPING constraint set to the *proper production stepping level*.
- In the case where only DFS outputs are used, and when CLKIN of the DCM is outside of the range for DLL outputs, a macro must be used to properly monitor the LOCKED signal. Verilog and VHDL versions of the macro can be downloaded from https://secure.xilinx.com/webreg/clickthrough.do?cid=30163.

  Note: This macro is not required for Step 1 and later XC4VLX and XC4VSX devices and SCD1 and later XC4VFX devices.

Phase Shifting

The DCM provides coarse and fine-grained phase shifting. For coarse-phase control, the CLK0, CLK90, CLK180, and CLK270 outputs are each phase-shifted by ¼ of the input clock period relative to each other. Similarly, CLK2X180 and CLKFX180 provide a 180° coarse phase shift of CLK2X and CLKFX, respectively. The coarse phase-shifted clocks are produced from the delay lines of the DLL circuit. The phase relationship of these clocks is retained when CLKFB is not connected.

Fine-grained phase shifting uses the CLKOUT_PHASE_SHIFT and PHASE_SHIFT attributes to phase-shift DCM output clocks relative to CLKIN. Since the CLKIN is used as the reference clock, the feedback (CLKFB) connection is required for the phase-shifting circuit to compare the incoming clock with the phase-shifted clock. The rest of this section describes fine-grained phase shifting in the Virtex-4 FPGA DCM.

Phase-Shifting Operation

All nine DCM output clocks are adjusted when fine-grained phase shifting is activated. The phase shift between the rising edges of CLKIN and CLKFB is a specified fraction of the input clock period or a specific amount of DCM_TAP. All other DCM output clocks retain their phase relation to CLK0.

Phase-Shift Range

The allowed phase shift between CLKIN and CLKFB is limited by the phase-shift range. There are two separate phase-shift range components:

- PHASE_SHIFT attribute range
- FINE_SHIFT_RANGE DCM timing parameter range

In the FIXED, VARIABLE_POSITIVE, and VARIABLE_CENTER phase-shift mode, the PHASE_SHIFT attribute is in the numerator of the following equation:

$$\text{Phase Shift (ns)} = (\text{PHASE\_SHIFT}/256) \times \text{PERIOD}_{\text{CLKIN}}$$

where $\text{PERIOD}_{\text{CLKIN}}$ denotes the effective CLKIN frequency.

In VARIABLE_CENTER and FIXED modes, the full range of the PHASE_SHIFT attribute is always –255 to +255. In the VARIABLE_POSITIVE mode, the range of the PHASE_SHIFT attribute is 0 to +255.

In the DIRECT phase-shift mode, the PHASE_SHIFT attribute is the multiplication factor in the following equation:

$$\text{Phase Shift (ns)} = \text{PHASE\_SHIFT} \times \text{DCM\_TAP}$$

In DIRECT modes, the full range of the PHASE_SHIFT attribute is 0 to 1023.

The FINE_SHIFT_RANGE component represents the total delay achievable by the phase-shift delay line. Total delay is a function of the number of delay taps used in the circuit. The absolute range is specified in the DCM Timing Parameters section of the *Virtex-4 Data Sheet* across process, voltage, and temperature. The different absolute ranges are outlined in this section.

The fixed mode allows the DCM to insert a delay line in the CLKFB or the CLKIN path. This gives access to the +FINE_SHIFT_RANGE when the PHASE_SHIFT attribute is set to a positive value, and –FINE_SHIFT_RANGE when the PHASE_SHIFT attribute is set to a negative value.

Absolute Range (Variable-Center Mode) = ± FINE_SHIFT_RANGE ÷ 2

The variable-center mode allows symmetric, dynamic sweeps from –255/256 to +255/256, by having the DCM set the zero-phase-skew point in the middle of the delay line. This divides the total delay-line range in half.

Absolute Range (Fixed) = ± FINE_SHIFT_RANGE

In the fixed mode, a phase shift is set during configuration in the range of –255/256 to +255/256.

Absolute Range (Variable-Positive and Direct Modes) = + FINE_SHIFT_RANGE

In the variable-positive and direct modes, the phase-shift only operates in the positive range. The DCM sets the zero-phase-skew point at the beginning of the delay line. This produces a full delay line in one direction.

Both the PHASE_SHIFT attribute and the FINE_SHIFT_RANGE parameter need to be considered to determine the limiting range of each application. The "Phase-Shift Examples" section illustrates possible scenarios.

In variable and direct mode, the PHASE_SHIFT value can dynamically increment or decrement as determined by PSINCDEC synchronously to PSCLK, when the PSEN input is active.

Phase-Shift Examples

The following usage examples take both the PHASE_SHIFT attribute and the FINE_SHIFT_RANGE components into consideration:

- If PERIODCLKIN = 2 × FINE_SHIFT_RANGE, then the PHASE_SHIFT in fixed mode is limited to ±128. In variable-positive mode, PHASE_SHIFT is limited to +128. In variable-center mode the PHASE_SHIFT is limited to ±64.
- If PERIODCLKIN = FINE_SHIFT_RANGE, then the PHASE_SHIFT in variable-positive mode is limited to +255. In fixed and variable-center mode the PHASE_SHIFT is limited to ±255.
- If PERIODCLKIN ≤ FINE_SHIFT_RANGE, then the PHASE_SHIFT in variable-positive mode is limited to +255. In fixed and variable-center mode the PHASE_SHIFT is limited to ±255.
- For all previously described cases, the direct mode is always limited to +1023.

If the phase shift is limited by the FINE_SHIFT_RANGE, use the coarse-grained phase shift to extend the phase-shift range or set DCM_PERFORAMANCE_MODE attribute to MAX_RANGE to increase the FINE_SHIFT_RANGE. Figure 2-6 illustrates using CLK90, CLK180, and CLK270 outputs assuming FINE_SHIFT_RANGE = 10 ns.

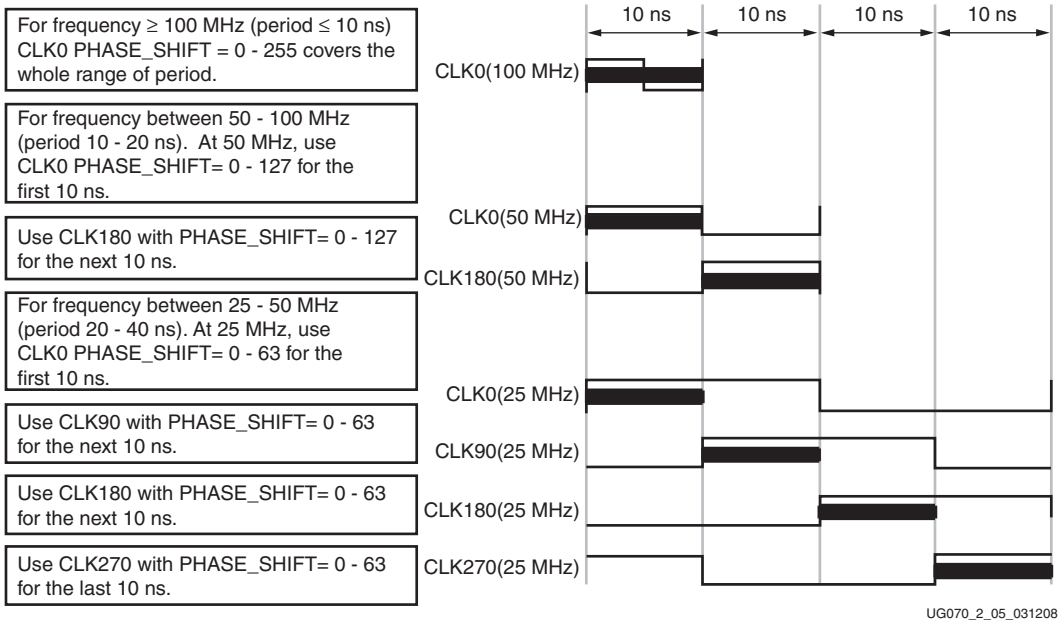


Figure 2-6: **Fixed Phase-Shift Examples**

In variable mode, the phase-shift factor is changed by activating PSEN for one period of PSCLK. At the PSCLK clock cycle where PSEN is activated, the level of PSINCDEC input determines whether the phase-shift increases or decreases. A High on PSINCDEC increases the phase shift, and a Low decreases the phase shift.

After the deskew circuit increments or decrements, the signal PSDONE is asserted High for a single PSCLK cycle. This allows the next change to be performed.

The user interface and the physical implementation are different. The user interface describes the phase shift as a fraction of the clock period (N/256). The physical implementation adds the appropriate number of buffer stages (each DCM_TAP) to the clock delay. The DCM_TAP granularity limits the phase resolution at higher clock frequencies.

All phase-shift modes, with the exception of DIRECT mode, are temperature and voltage adjusted. Hence, a V_{CC} or temperature adjustment will not change the phase shift. The DIRECT phase shift is not temperature or voltage adjusted since it directly controls DCM_TAP. Changing the ratio of V_{CC}/temperature results in a phase-shift change proportional to the size of the DCM_TAP at the specific voltage and temperature.

Interaction of PSEN, PSINCDEC, PSCLK, and PSDONE

The variable and direct phase-shift modes are controlled by the PSEN, PSINCDEC, PSCLK, and PSDONE ports. In addition, a phase-shift overflow (DO[0]) status indicates when the phase-shift counter has reached the end of the phase-shift delay line or the maximum value (±255 for variable mode, +1023 for direct mode).

After the DCM locks, the initial phase in the VARIABLE_POSITIVE and VARIABLE_CENTER modes is determined by the PHASE_SHIFT value. The initial phase in the DIRECT mode is always 0, regardless of the value specified by the PHASE_SHIFT attribute The non-zero PHASE_SHIFT value for DIRECT mode can only be loaded to the DCM when a specific "load phase shift value" command is given by Dynamic Reconfiguration. Refer to the "Techniques" section in the *Virtex-4 Configuration Guide* for more information. The phase of DCM output clock will be incremented/decremented according to the interaction of PSEN, PSINCDEC, PSCLK, and PSDONE from the initial or dynamically reconfigured phase.

PSEN, PSINCDEC, and PSDONE are synchronous to PSCLK. When PSEN is asserted for one PSCLK clock period, a phase-shift increment/decrement is initiated. When PSINCDEC is High, an increment is initiated and when PSINCDEC is Low, a decrement is initiated. Each increment adds to the phase shift of DCM clock outputs by 1/256 of the CLKIN period. Similarly, each decrement decreases the phase shift by 1/256 of the CLKIN period. PSEN must be active for exactly one PSCLK period; otherwise, a single phase-shift increment/decrement is not guaranteed. PSDONE is High for exactly one clock period when the phase shift is complete. The time required to complete a phase-shift operation varies. As a result, PSDONE must be monitored for phase-shift status. Between enabling PSEN and PSDONE is flagged, the DCM output clocks will gradually change from their original phase shift to the incremented/decremented phase shift. The completion of the increment or decrement is signaled when PSDONE asserts High. After PSDONE has pulsed High, another increment/decrement can be initiated.

Figure 2-7 illustrates the interaction of phase-shift ports.

When PSEN is activated after the phase-shift counter has reached the maximum value of PHASE_SHIFT, the PSDONE will still be pulsed High for one PSCLK period some time after the PSEN is activated (as illustrated in Figure 2-7). However, the phase-shift overflow pin, STATUS(0), or DO(0) will be High to flag this condition, and no phase adjustment is performed.

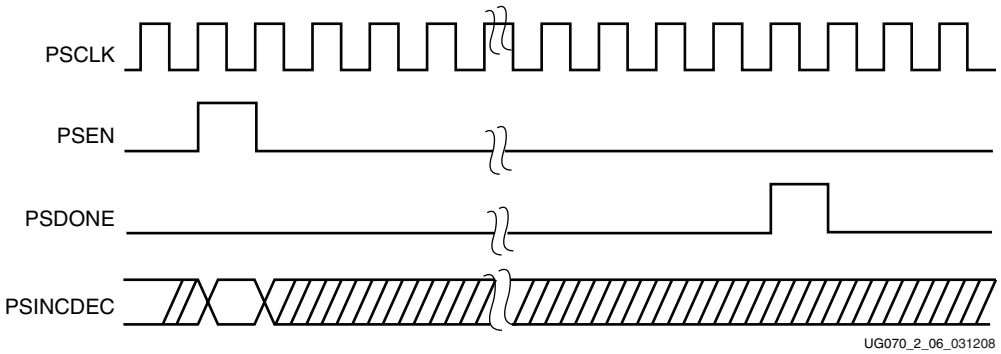


Figure 2-7: **Phase-Shift Timing Diagram**

Phase-Shift Overflow

The phase-shift overflow (DO[0]) status signal is asserted when either of the following conditions are true.

The DCM is phase-shifted beyond the allowed phase-shift value. In this case, the phase-shift overflow signal will be asserted High when the phase shift is decremented beyond –255 and incremented beyond +255 for VARIABLE_CENTER mode, incremented beyond +255 for VARIABLE_POSITIVE mode, or decremented beyond 0 and incremented beyond 1023 for DIRECT mode.

The DCM is phase-shifted beyond the absolute range of the phase-shift delay line. In this case, the phase-shift overflow signal will be assert High when the phase-shift in time (ns) exceeds the ±FINE_SHIFT_RANGE/2 in the VARIABLE_CENTER mode, the +FINE_SHIFT_RANGE in the VARIABLE_POSITIVE mode, or exceeds 0 to +FINE_SHIFT_RANGE in the DIRECT mode. The phase-shift overflow signal can toggle once it is asserted. The condition determining if the delay line is exceeded is calibrated dynamically. Therefore, at the boundary of exceeding the delay line, it is possible for the phase-shift overflow signal to assert and de-assert without a change in phase shift. Once asserted, it will remain asserted for at least 40 CLKIN cycles. If the DCM is operating near the FINE_SHIFT_RANGE limit, do not use the phase-shift overflow signal as a flag to reverse the phase shift direction. When the phase-shift overflow is asserted, de-asserted, then asserted again in a short phase shift range, it can falsely reverse the phase shift direction. Instead, use a simple counter to track the phase shift value and reverse the phase shift direction (PSINCDEC) only when the counter reaches a previously determined maximum/minimum phase shift value. For example, if the phase shift must be within 0 to 128, set the counter to toggle PSINCDEC when it reaches 0 or 128.

Phase-Shift Characteristics

- Offers fine-phase adjustment with a resolution of ±1/256 of the clock period (or ± one DCM_TAP, whichever is greater). It can be dynamically changed under user control.
- The phase-shift settings affect all nine DCM outputs.
- V_{CC} and temperature do not affect the phase shift except in direct phase-shift mode.
- In either fixed or variable mode, the phase-shift range can be extended by choosing CLK90, CLK180, or CLK270, rather than CLK0, choosing CLK2X180 rather than CLK2X, or choosing CLKFX180 rather than CLKFX. Even at 25 MHz (40 ns period), the fixed mode coupled with the various CLK phases allows shifting throughout the entire input clock period range.
- MAX_RANGE mode extends the phase-shift range.

- The phase-shifting (DPS) function in the DCM requires the CLKFB for delay adjustment.

  Because CLKFB must be from CLK0, the DLL output is used. The minimum CLKIN frequency for the DPS function is determined by DLL frequency mode.

Dynamic Reconfiguration

The Dynamic Reconfiguration Ports (DRPs) can update the initial DCM settings without reloading a new bitstream to the FPGA. The *Virtex-4 Configuration Guide* provides more information on using DRPs. Specific to the DCM, DRPs can perform the following functions:

- Allow dynamic adjustment of CLKFX_MULTIPLY(M) and CLKFX_DIVIDE(D) value to produce a new CLKFX frequency.
- Allow dynamic adjustment of PHASE_SHIFT value to produce a new phase shift. This feature can be used with the fixed, variable, or direct phase-shift modes to set a specific phase-shift value.

The following steps are required when using DRPs to load new M and D values:

- Subtract the desired M and D values by one. For example, if the desired M/D = 9/4, then load M/D = 8/3.
- Hold DCM in reset (assert RST signal) and release it after the new M and D values are written. The CLKFX outputs can be used after LOCKED is asserted High again.

Connecting DCMs to Other Clock Resources in Virtex-4 Devices

Most DCM functions require connection to dedicated clock resources, including dedicated clock I/O (IBUFG), clock buffers (BUFGCTRLs), and PMCD. These clock resources are located in the center column of the Virtex-4 devices. This section provides guidelines on connecting the DCM to dedicated clock resources.

IBUFG to DCM

Virtex-4 devices contain either 16 or 32 clock inputs. These clock inputs are accessible by instantiating the IBUFG component. Each top and bottom half of a Virtex-4 device contains eight or 16 IBUFGs. Any of the IBUFG in top or bottom half of the Virtex-4 device can drive the clock input pins (CLKIN, CLKFB, PSCLK, or DCLK) of a DCM located in the same top/bottom half of the device.

DCM to BUFGCTRL

Any DCM clock output can drive any BUFGCTRL input in the same top/bottom half of the device. There are no restrictions on how many DCM outputs can be used simultaneously.

BUFGCTRL to DCM

Any BUFGCTRL can drive any DCM in the Virtex-4 devices. However, only up to eight dedicated clock routing resources exist in a particular clock region. Since the clock routing is accessed via the BUFGCTRL outputs, this indirectly limits the BUFGCTRL to DCM connection. If eight BUFGCTRL outputs are already accessing a clock region, and a DCM is in that region, then no additional BUFGCTRL can be used in that region, including a connection to the FB pin of the DCM.

DCM to and from PMCD

Refer to the PMCD chapter: "Phase-Matched Clock Dividers (PMCDs)".

Application Examples

The Virtex-4 FPGA DCM can be used in a variety of creative and useful applications. The following examples show some of the more common applications.

Standard Usage

The circuit in Figure 2-8 shows DCM_BASE implemented with internal feedback and access to RST and LOCKED pins. This example shows the simplest use case for a DCM.

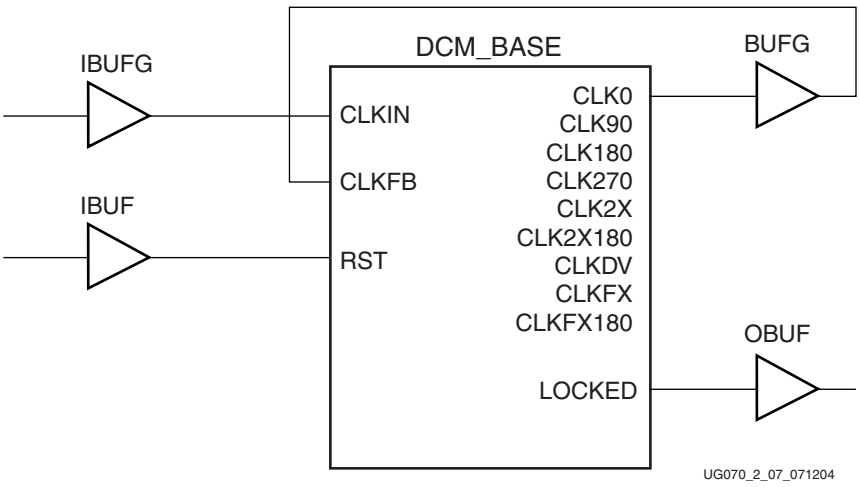


Figure 2-8: **Standard Usage**

Board-Level Clock Generation

The board-level clock generation example in Figure 2-9 illustrates how to use a DCM to generate output clocks for other components on the board. This clock can then be used to interface with other devices. In this example, a DDR register is used with its inputs connected to GND and V_{CC}. Because the output of the DCM is routed to BUFG, the clock stays within global routing until it reaches the output register. The quality of the clock is maintained.

If the design requires global buffers in other areas, use an OBUF instead of BUFG and ODDR (Figure 2-10).

However, the clock quality will not be as well preserved as when connected using a global buffer and a DDR register (Figure 2-11).

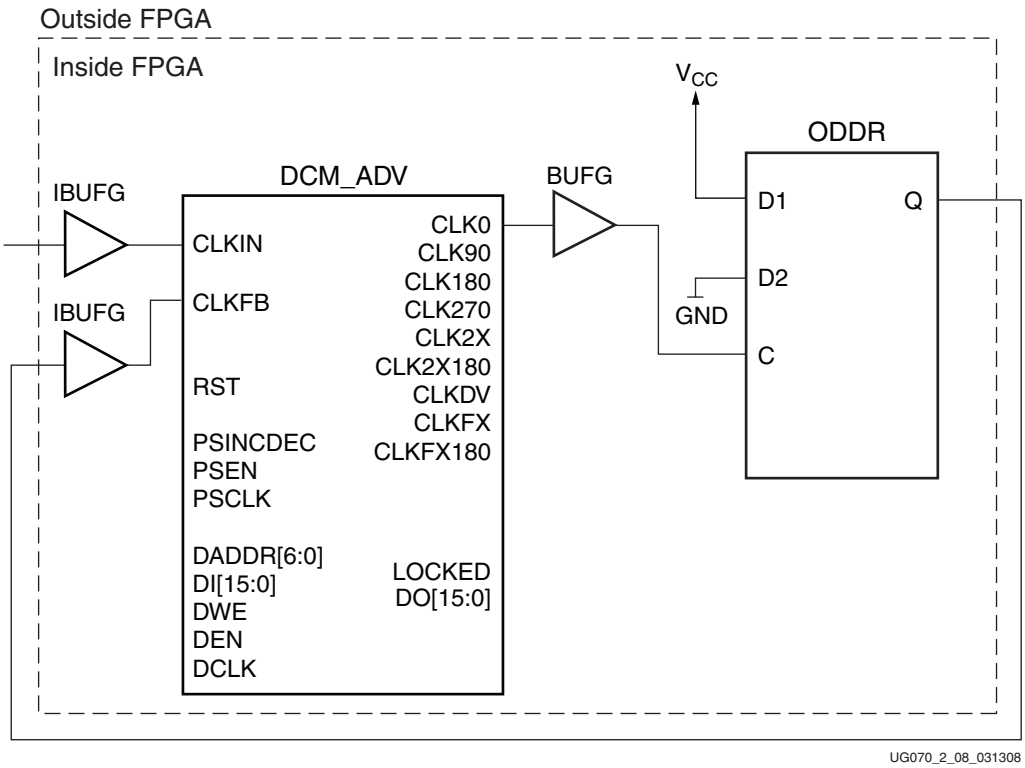


Figure 2-9: **Board-Level Clock Using DDR Register with External Feedback**

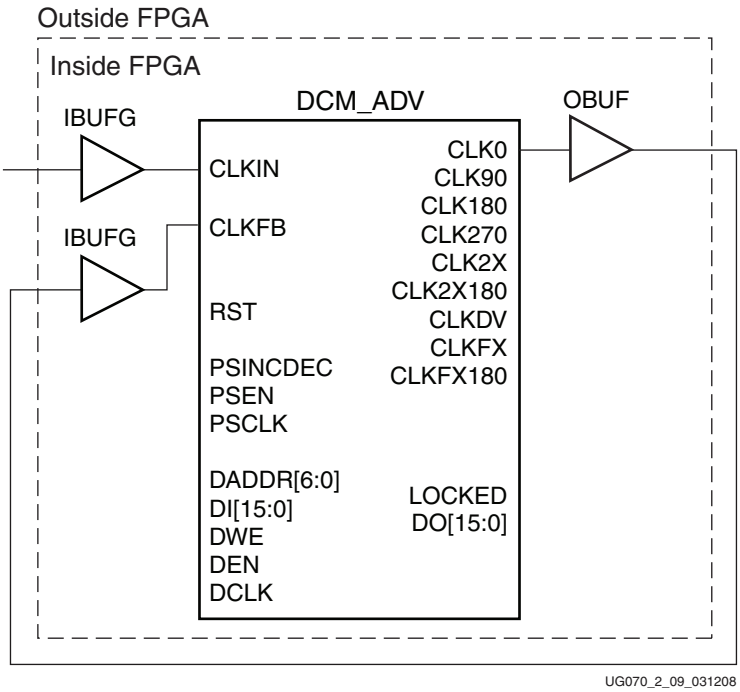


Figure 2-10: **Board-Level Clock Using OBUF with External Feedback**

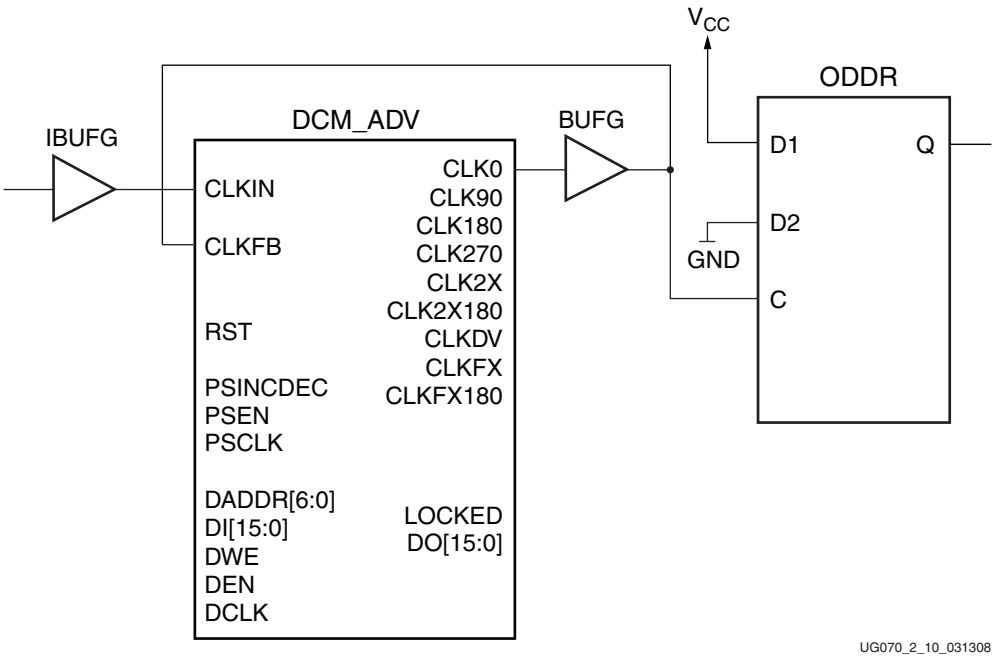


Figure 2-11: **Board-Level Clock with Internal Feedback (Clock Forwarding)**

Board Deskew with Internal Deskew

Some applications require board deskew with internal deskew to interface with other devices. These applications can be implemented using two or more DCM. The circuit shown in Figure 2-12 can be used to deskew a system clock between multiple Virtex devices in the same system.

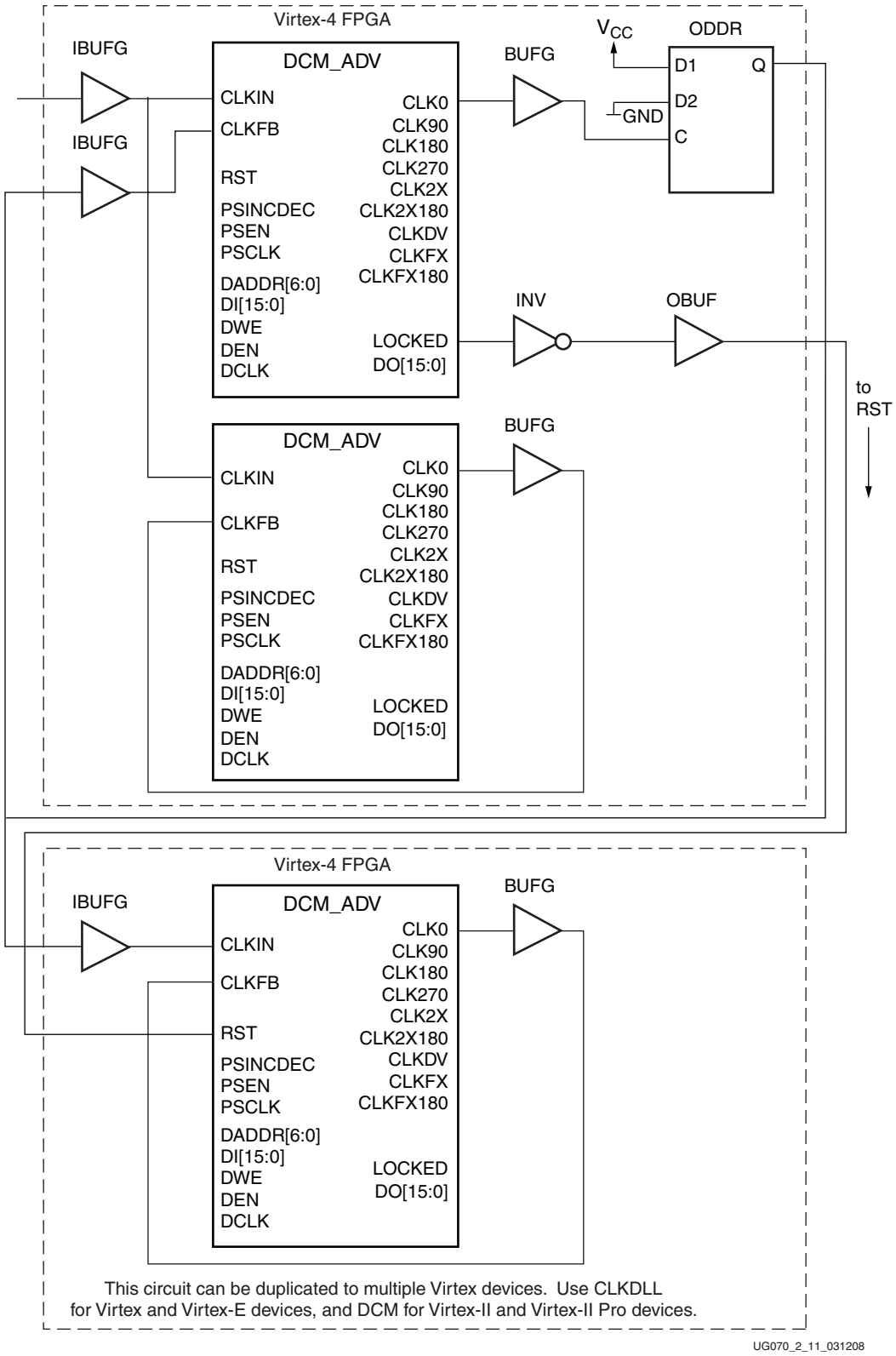


Figure 2-12: **Board Deskew with Internal Deskew Interfacing to Other Virtex Devices**

The example in Figure 2-13 shows an interface from Virtex-4 FPGAs to non-Virtex devices.

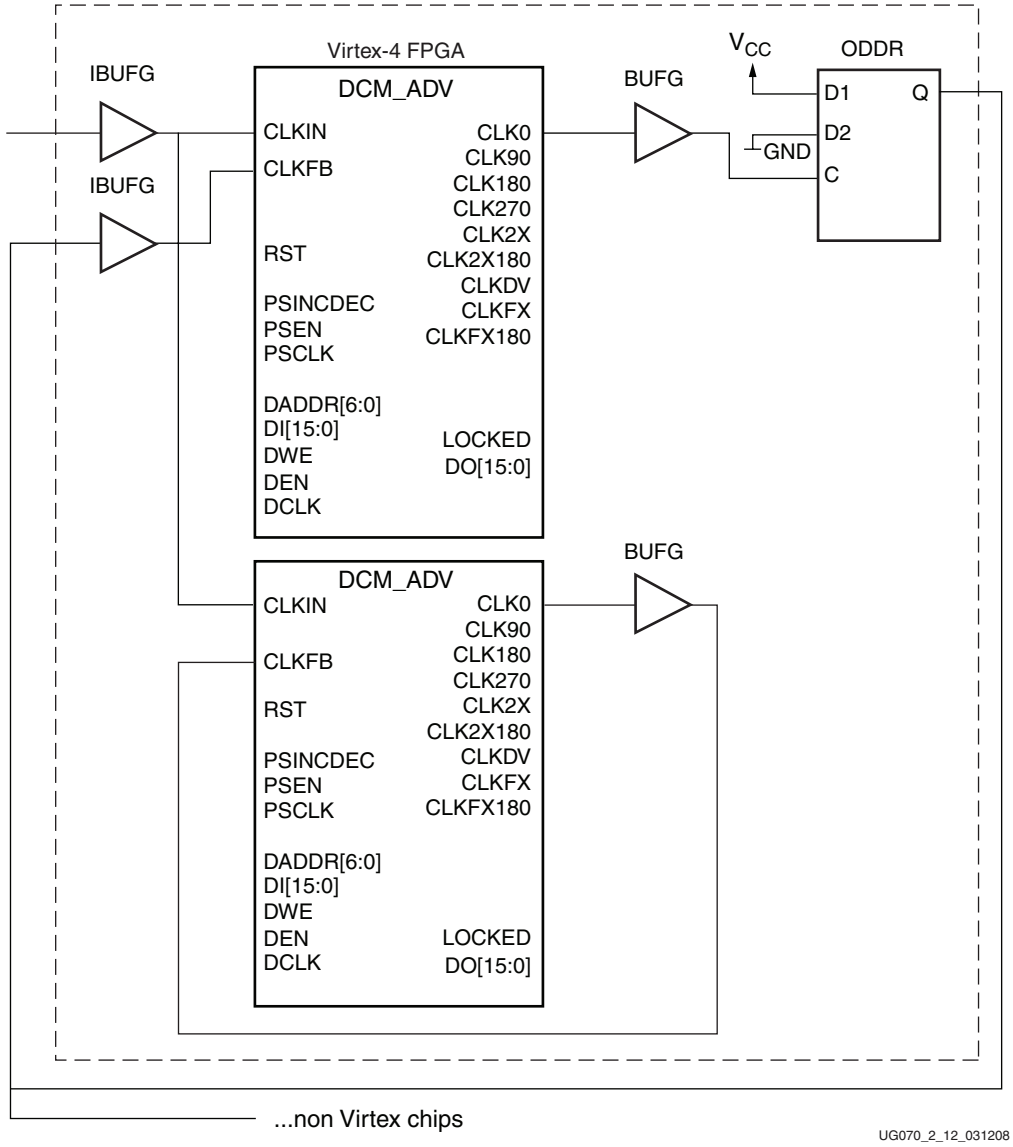


Figure 2-13: **Board Deskew with Internal Deskew Interfacing to Non-Virtex Devices**

Clock Switching Between Two DCMs

Figure 2-14 illustrates switching between two clocks from two DCMs while keeping both DCMs locked.

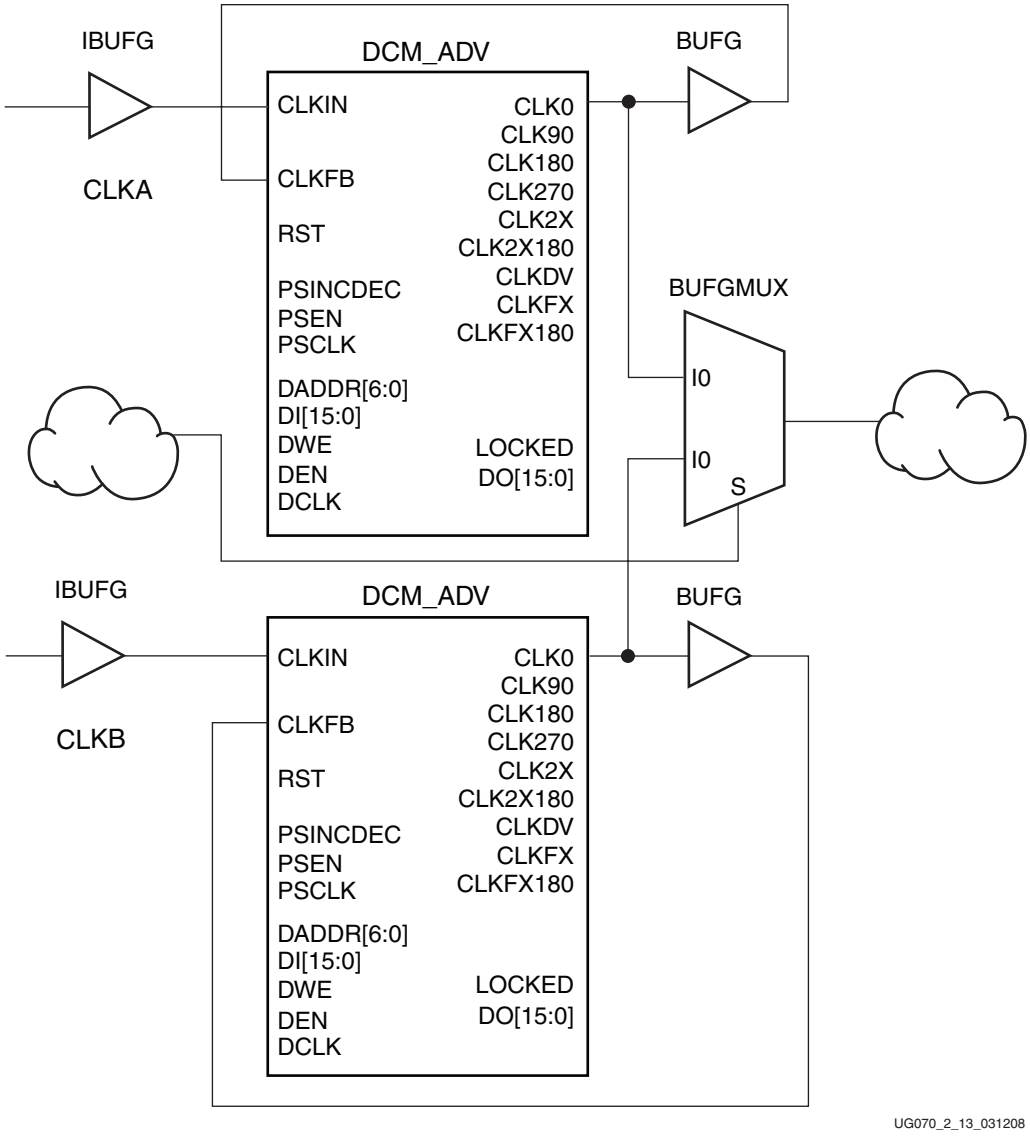


Figure 2-14: **Clock Switching Between Two DCMs**

VHDL and Verilog Templates, and the Clocking Wizard

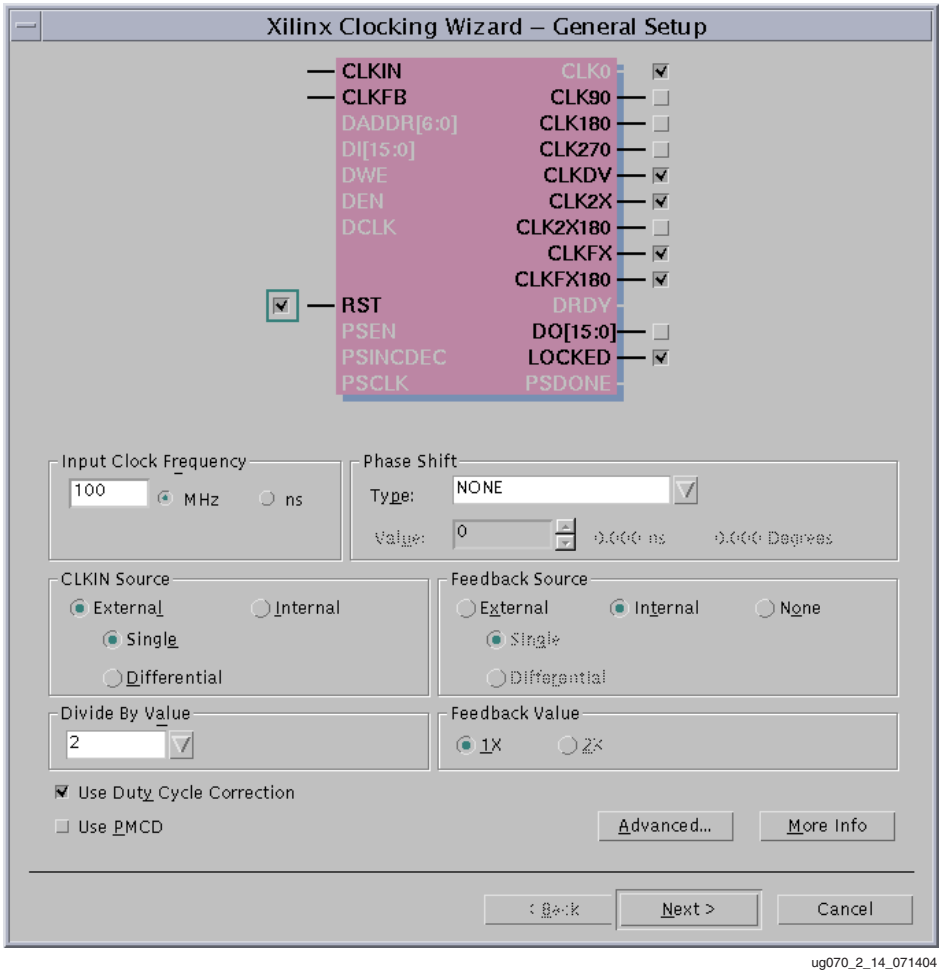
VHDL and Verilog instantiation templates are available in the *Libraries Guide* for all primitives. In addition, VHDL and Verilog files are generated by the Xilinx® Clocking Wizard in the ISE software tool. The Clocking Wizard sets appropriate DCM attributes, input/output clocks, and buffers for general use cases.

The Clocking Wizard is accessed using the ISE software tool, in the Project Navigator. Refer to the Xilinx Software Manuals for more information on ISE software.

1. From the Project Navigator menu, select **Project** → **New Source**. The New Source window appears.
2. Enter a file name and select **IP** (CoreGen and Architecture Wizard).
3. Click **Next**. The Select Core Type window appears.
4. Select **Clocking** → **Single DCM_ADV** and click **Next**. The New Source Information window appears.
5. Click **Finish**.
6. The Clocking Wizard starts.

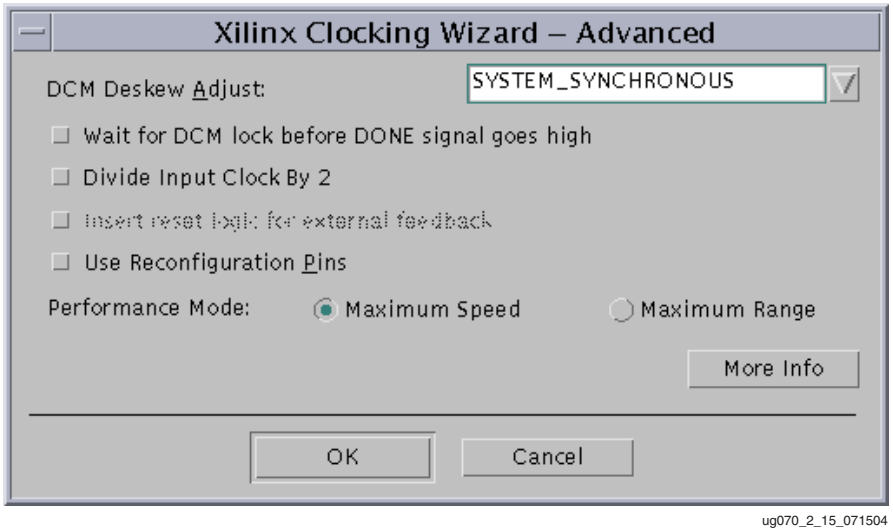
   Figure 2-15 to Figure 2-19 show the settings available in the Clocking Wizard.

   - Figure 2-15 provides the general settings for the DCM.
   - After choosing the **Advanced** button, the window shown in Figure 2-16 provides the advanced setting choices.
   - The windows in Figure 2-17 and Figure 2-18 show the settings for the global buffers using the previously selected DCM clock outputs.
   - When **CLKFX** or **CLKFX180** is selected, the Clock Frequency Synthesizer window shown in Figure 2-19 appears. This window provides the CLKFX jitter calculation. To access further information on available settings, choose the **More Info** button in each window.



ug070_2_14_071404

Figure 2-15: **Xilinx Clocking Wizard — General Setup**



ug070_2_15_071504

Figure 2-16: **Xilinx Clocking Wizard — Advanced**

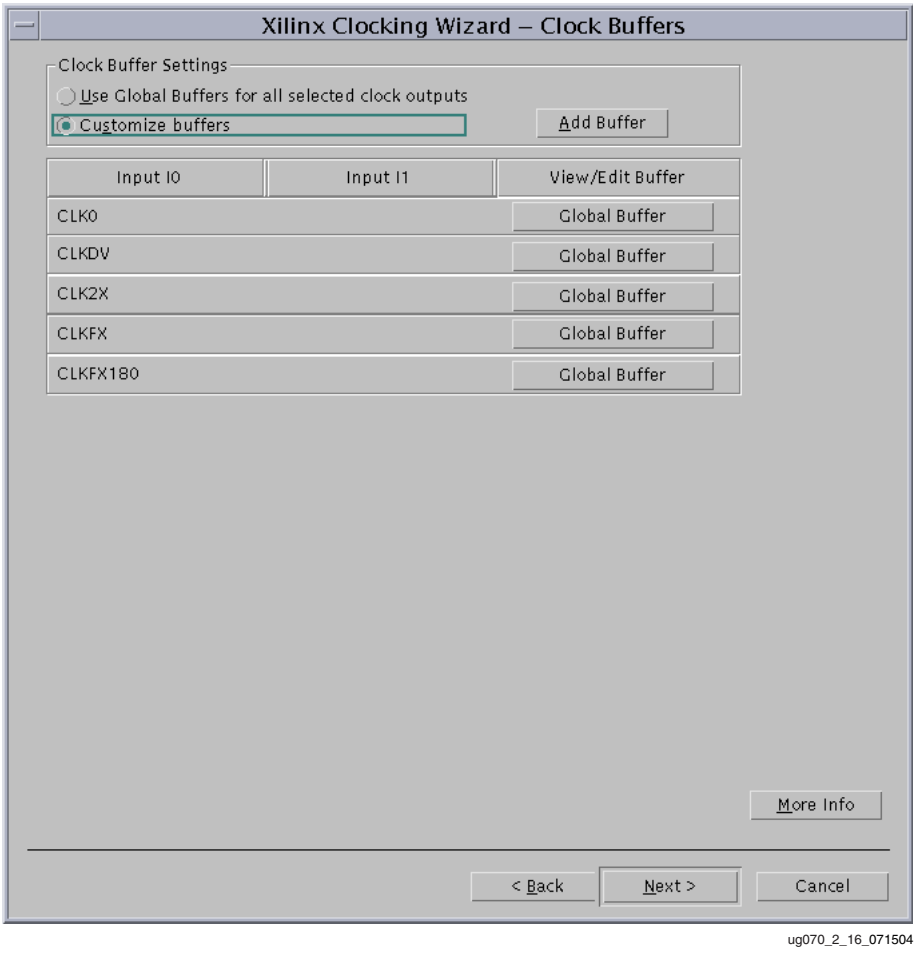


Figure 2-17: **Xilinx Clocking Wizard — Clock Buffers**

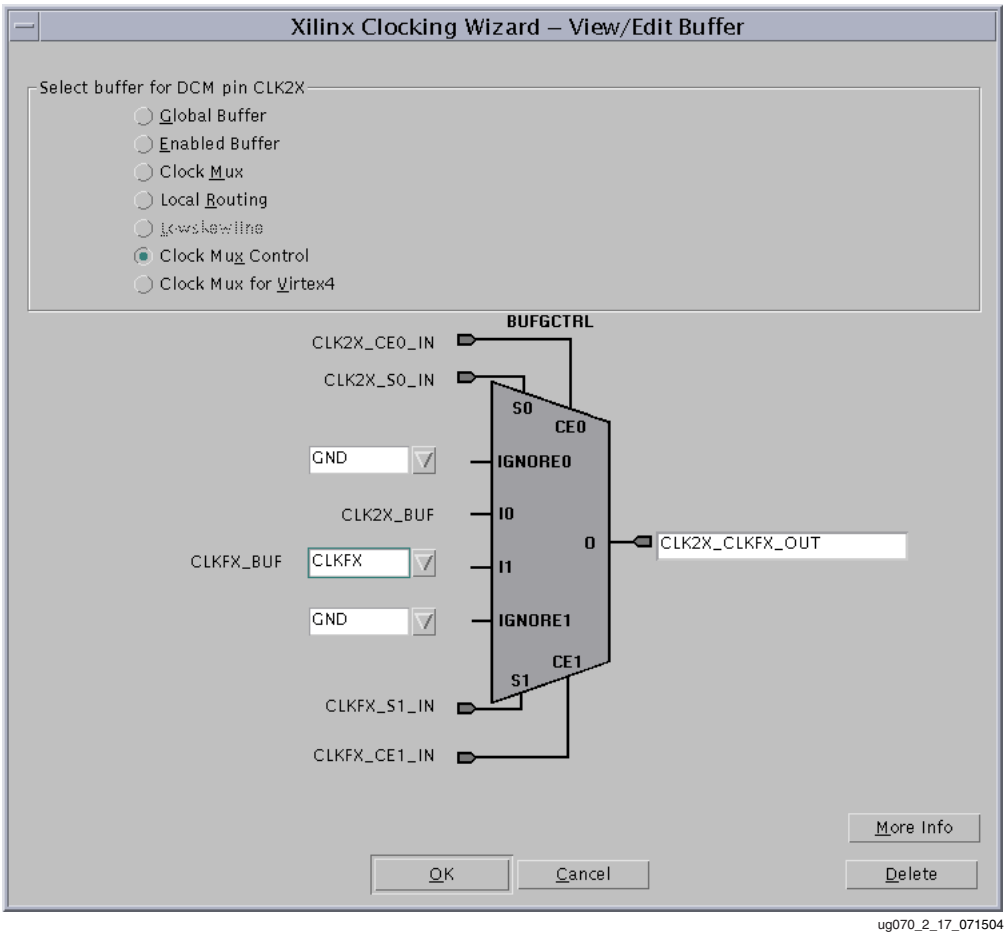


Figure 2-18: **Xilinx Clocking Wizard — View/Edit Buffer**

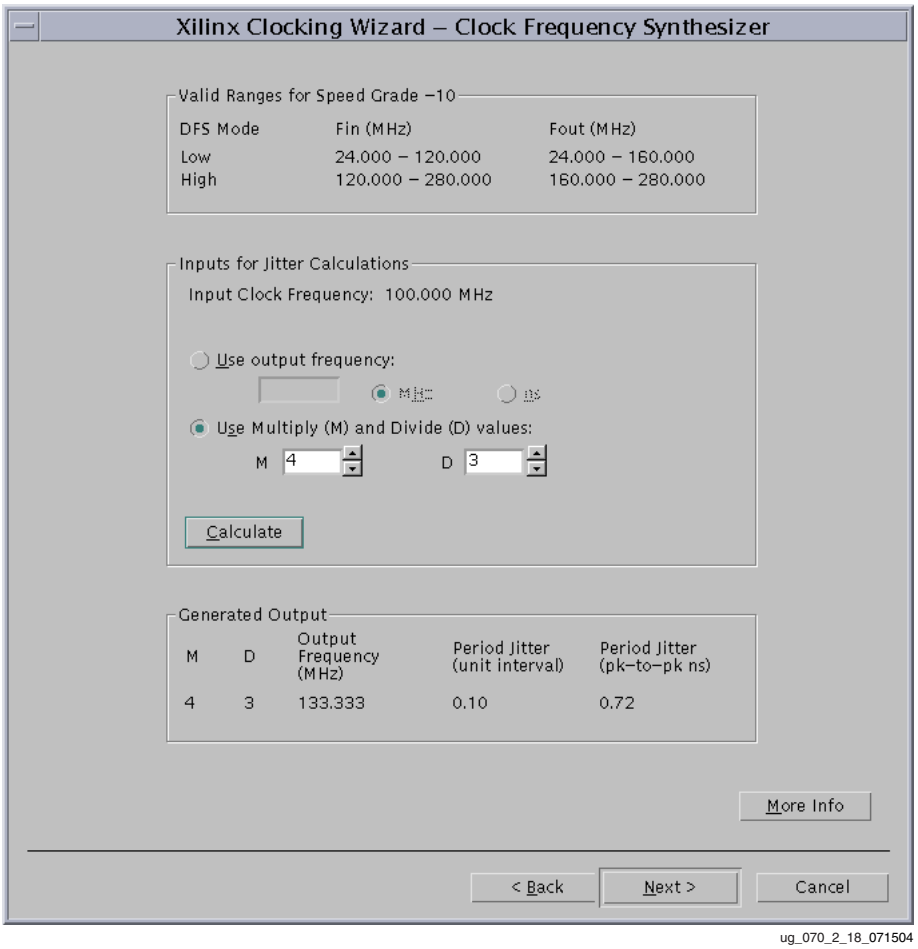


Figure 2-19: **Xilinx Clocking Wizard — Frequency Synthesizer**

7. When all the desired settings are selected, choose the **Finish** button.
8. The Clocking Wizard closes and the Project Navigator window returns.
   - The Clocking Wizard writes the selected settings into an .XAW file.
   - The .XAW file appears in the Sources in Project window list.
   - Select the .XAW file. In the Processes for Source window, double-click on **View HDL Source** or **View HDL Instantiation Template**. The HDL source or instantiation template will be generated. These are read-only files for inclusion or instantiation in a design.
   - To return to the Clocking Wizard, double-click on the .XAW file. The Clocking Wizard appears with the previously selected settings. These settings can be changed and the .XAW file updated to accommodate design changes.

DCM Timing Models

The following timing diagrams describe the behavior of the DCM clock outputs under four different conditions.

1. Reset/Lock
2. Fixed-Phase Shifting
3. Variable-Phase Shifting
4. Status Flags

Reset/Lock

In Figure 2-20, the DCM is already locked. After the reset signal is applied, all output clocks are stabilized to the desired values, and the LOCKED signal is asserted.

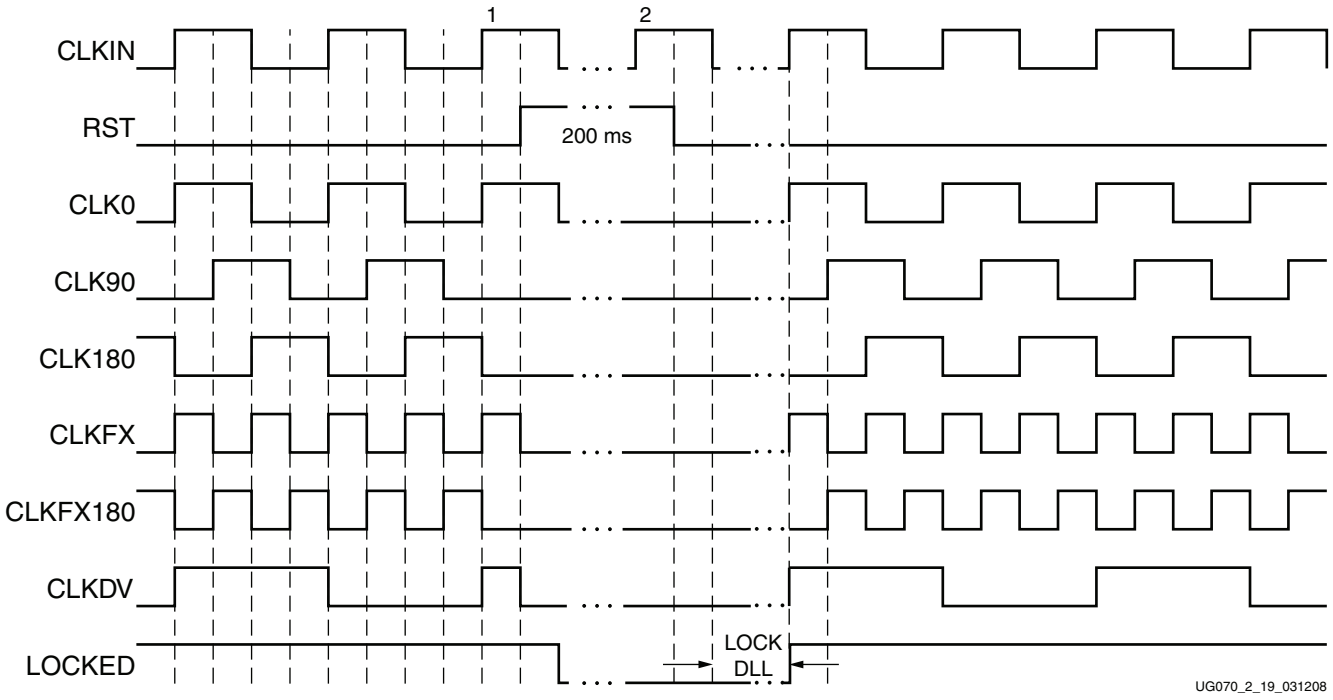


Figure 2-20: **RESET/LOCK Example**

- **Prior to Clock Event 1**

  Prior to clock event 1, the DCM is locked. All clock outputs are in phase with the correct frequency and behavior.

- **Clock Event 1**

  Some time after clock event 1 the reset signal is asserted at the (RST) pin. While reset is asserted, all clock outputs become a logic zero. The reset signal is an asynchronous reset. Note: the diagram is not shown to scale. For the DCM to operate properly, the reset signal must be asserted for at least 200 ms.

- **Clock Event 2**

  Clock event 2 occurs a few cycles after reset is asserted and deasserted. At clock event 2, the lock process begins. At time LOCK_DLL, after clock event 2, if no fixed phase shift was selected then all clock outputs are stable and in phase. LOCKED is also asserted to signal completion.

Fixed-Phase Shifting

In Figure 2-21, the DCM outputs the correct frequency. However, the clock outputs are not in phase with the desired clock phase. The clock outputs are phase-shifted to appear sometime later than the input clock, and the LOCKED signal is asserted.

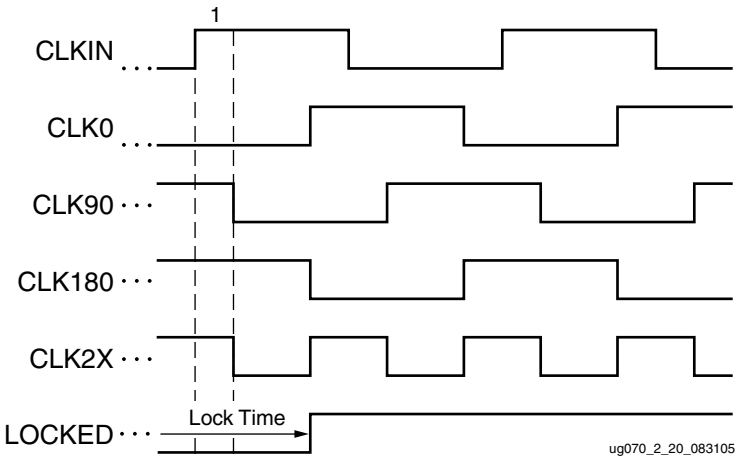


Figure 2-21: **Phase Shift Example: Fixed**

- **Clock Event 1**

  Clock event 1 appears after the desired phase shifts are applied to the DCM. In this example, the shifts are positive shifts. CLK0 and CLK2X are no longer aligned to CLKIN. However, CLK0, and CLK2X are aligned to each other, while CLK90 and CLK180 remain as 90° and 180° versions of CLK0. The LOCK signal is also asserted once the clock outputs are ready.

Variable-Phase Shifting

In Figure 2-22, the CLK0 output is phase-shifted using the dynamic phase-shift adjustments in the synchronous user interface. The PSDONE signal is asserted for one cycle when the DCM completes one phase adjustment. After PSDONE is deasserted, PSEN can be asserted again, allowing an additional phase shift to occur.

As shown in Figure 2-22, all the variable-phase shift control and status signals are synchronous to the rising edge of PSCLK.

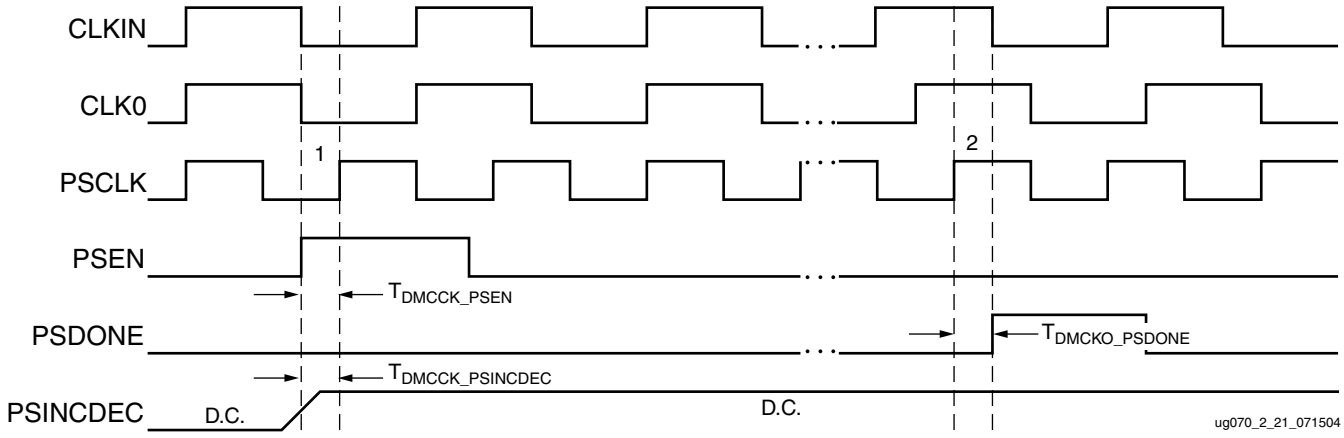


Figure 2-22: **Phase Shift Example: Variable**

- **Clock Event 1**

  At T_{DMCCK_PSEN}, before clock event 1, PSEN is asserted. PSEN must be active for exactly one clock period; otherwise, a single increment/decrement of phase shift is not guaranteed. Also, the PSINCDEC value at $T_{DMCCK\_PSINCDEC}$, before clock event 1, determines whether it is an increment (logic High) or a decrement (logic Low).

- **Clock Event 2**

  At T_{DMCKO_PSDONE}, after clock event 2, PSDONE is asserted to indicate one increment or decrement of the DCM outputs. PSDONE is High for exactly one clock period when the phase shift is complete. The time required for a complete phase shift will vary. As a result, PSDONE must be monitored for phase-shift status.

Status Flags

The example in Figure 2-23 shows the behavior of the status flags in the event of a phase-shift overflow and CLKIN/CLKFB/CLKFX failure.

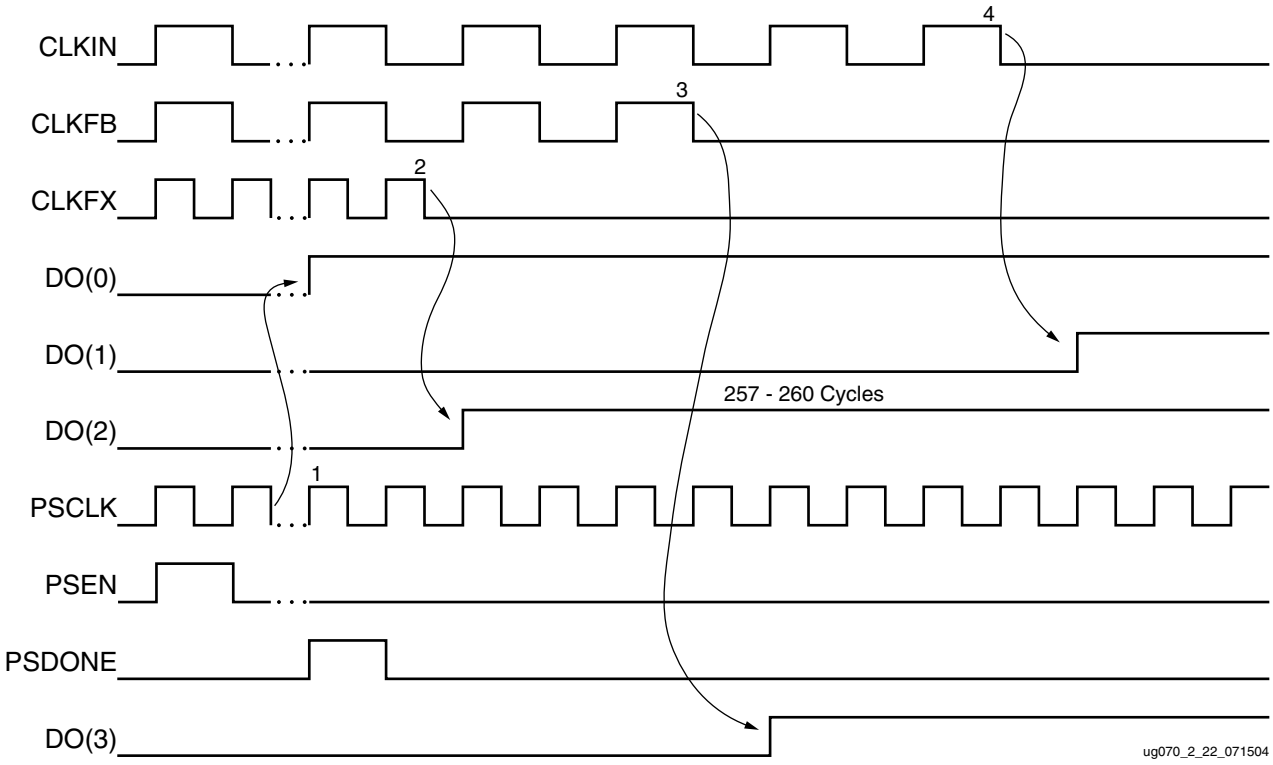


Figure 2-23: **Status Flags Example**

- **Clock Event 1**

  Prior to the beginning of this timing diagram, CLK0 (not shown) is already phase-shifted at its maximum value. At clock event 1, PSDONE is asserted. However, since the DCM has reached its maximum phase-shift capability no phase adjustment is performed. Instead, the phase-shift overflow status pin DO[0] is asserted to indicate this condition.

- **Clock Event 2**

  The CLKFX output stops toggling. Within 257 to 260 clock cycles after this event, the CLKFX stopped status DO[2] is asserted to indicate that the CLKFX output stops toggling.

- **Clock Event 3**

  The CLKFB input stops toggling. Within 257 to 260 clock cycles after this event, the CLKFB stopped status DO[3] is asserted to indicate that the CLKFB output stops toggling.

- **Clock Event 4**

  The CLKIN input stops toggling. Within 9 clock cycles after this event, DO[1] is asserted to indicate that the CLKIN output stops toggling.

Legacy Support

The Virtex-4 device supports the Virtex-II and Virtex-II Pro family DCM primitives. The mapping of Virtex-II or Virtex-II Pro FPGA DCM components to Virtex-4 FPGA DCM_ADV components are as follows:

- CLKIN, CLKFB, PSCLK, PSINDEC, PSEN, RST, CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKFX, CLKFX180, CLKDV, PSDONE, LOCKED of Virtex-4 FPGA primitives (DCM_BASE/DCM_PS/DCM_ADV) map to the same corresponding pins of a Virtex-II or Virtex-II Pro FPGA DCM.
- Dynamic reconfiguration pins of Virtex-4 FPGA DCM_ADV are not accessible when a Virtex-II or Virtex-II Pro FPGA DCM component is used, except for DO[15:0].
- DO[7:0] pins of Virtex-4 FPGA DCM_ADV/DCM_PS components map to Status[7:0] of the Virtex-II or Virtex-II Pro FPGA DCMs. DO[15:8] of DCM_ADV/DCM_PS components are not available when using Virtex-II or Virtex-II Pro FPGA DCM components.

XILINX®

Chapter 3

Phase-Matched Clock Dividers (PMCDs)

PMCD Summary

The Phase-Matched Clock Dividers (PMCDs) are one of the clock resources available in the Virtex-4 architecture. PMCDs provide the following clock management features:

- Phase-Matched Divided Clocks

  The PMCDs create up to four frequency-divided and phase-matched versions of an input clock, CLKA. The output clocks are a function of the input clock frequency: divided-by-1 (CLKA1), divided-by-2 (CLKA1D2), divided-by-4 (CLKA1D4), and divided-by-8 (CLKA1D8). CLKA1, CLKA1D2, CLKA1D4, and CLKA1D8 output clocks are rising-edge aligned to each other but not to the input (CLKA).

- Phase-Matched Delay Clocks

  PMCDs preserve edge alignments, phase relations, or skews between the input clock CLKA and other PMCD input clocks. Three additional inputs (CLKB, CLKC, and CLKD) and three corresponding delayed outputs (CLKB1, CLKC1, and CLKD1) are available. The same delay is inserted to CLKA, CLKB, CLKC, and CLKD; thus, the delayed CLKA1, CLKB1, CLKC1, and CLKD1 outputs maintain edge alignments, phase relations, and the skews of the respective inputs.

  A PMCD can be used with other clock resources including global buffers and DCMs. Together, these clock resources provide flexibility in managing complex clock networks in designs.

  In Virtex-4 devices, the PMCDs are located in the center column. Figure 3-1 shows a simplified view of the Virtex-4 FPGA center column resources. The PMCDs are grouped, with two PMCDs in one tile. The PMCDs in each tile have special characteristics to support applications requiring multiple PMCDs. Table 3-1 summarizes the availability of PMCDs in each Virtex-4 device.

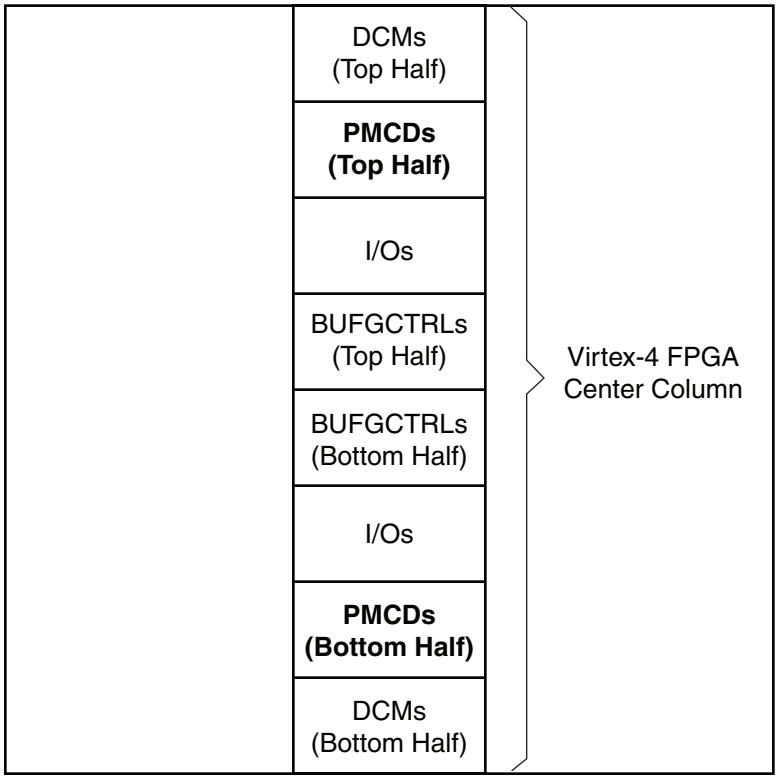


Figure 3-1: **PMCD Location in the Virtex-4 Device**

Table 3-1: **Available PMCD Resources**

Device	Available PMCDs	Site Names
XC4VLX15 XC4VSX25 XC4VFX12, XC4VFX20	0	No PMCDs in these devices
XC4VLX25, XC4VLX40, XC4VLX60 XC4VSX35, XC4VSX55 XC4VFX40	4	Bottom Half: PMCD_X0Y0, PMCD_X0Y1 (one tile) Top Half: PMCD_X0Y2, PMCD_X0Y3 (one tile)
XC4VLX80, XC4VLX100, XC4VLX160, XC4VLX200 XC4VFX60, XC4VFX100, XC4VFX140	8	Bottom Half: PMCD_X0Y0, PMCD_X0Y1 (one tile) PMCD_X0Y2, PMCD_X0Y3 (one tile) Top Half: PMCD_X0Y4, PMCD_X0Y5 (one tile) PMCD_X0Y6, PMCD_X0Y7 (one tile)

PMCD Primitives, Ports, and Attributes

Figure 3-2 illustrates the PMCD primitive. The VHDL and Verilog template section includes an example of a PMCD instantiation template.

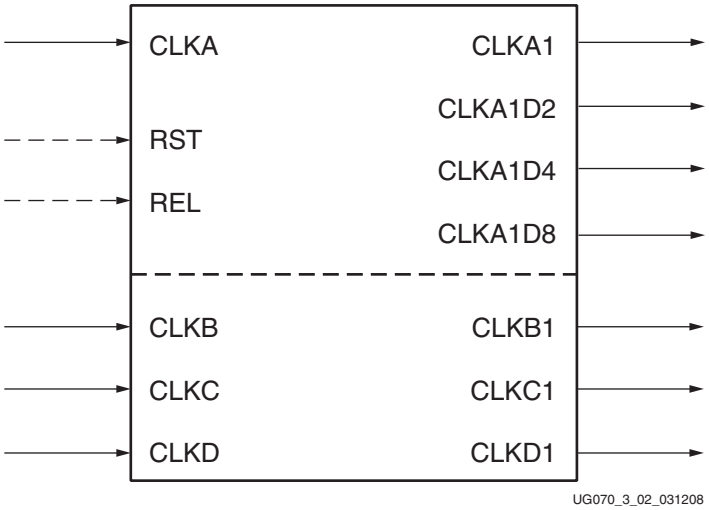


Figure 3-2: **PMCD Primitive**

Table 3-2 lists the port names and description of the ports.

Table 3-2: **PMCD Port Description**

Port Name	Direction	Description
CLKA	Input	CLKA is a clock input to the PMCD. The CLKA frequency can be divided by 1, 2, 4, and 8.
CLKB CLKC CLKD	Input	CLKB, CLKC, and CLKD are clock inputs to the PMCD. These clock are not divided by the PMCD; however, they are delayed by the PMCD to maintain the phase alignment and phase relationship at the input clocks.
RST	Input	RST is the reset input to the PMCD. Asserting the RST signal asynchronously forces all outputs Low. Deasserting RST synchronously allows all outputs to toggle.
REL	Input	REL is the release input to the PMCD. Asserting the REL signal releases the divided output synchronous to CLKA.
CLKA1	Output	The CLKA1 output has the same frequency as the CLKA input. It is a delayed version of CLKA.
CLKA1D2	Output	The CLKA1D2 output has the frequency of CLKA divided by two. CLKA1D2 is rising-edge aligned to CLKA1.
CLKA1D4	Output	The CLKA1D4 output has the frequency of CLKA divided by four. CLKA1D4 is rising-edge aligned to CLKA1.
CLKA1D8	Output	The CLKA1D8 output has the frequency of CLKA divided by eight, CLKA1D8 is rising-edge aligned to CLKA1.
CLKB1 CLKC1 CLKD1	Output	The CLKB1 output has the same frequency as the CLKB input, a delayed version of CLKB. The skew between CLKB1 and CLKA1 is the same as the skew between CLKB and CLKA inputs. Similarly, CLKC1 is a delayed version of CLKC, and CLKD1 is a delayed version of CLKD.

Table 3-3 lists the PMCD attributes.

Table 3-3: **PMCD Attributes**

PMCD Attribute Name	Description	Values	Default Value
RST_DEASSERT_CLK	This attribute allows the deassertion of the RST signal to be synchronous to a selected PMCD input clock.	String: CLKA, CLKB, CLKC, or CLKD	CLKA
EN_REL	This attribute allows for CLKA1D2, CLKA1D4, and CLKA1D8 outputs to be released at REL signal assertion. Note: REL is synchronous to CLKA input.	Boolean: FALSE, TRUE	FALSE

PMCD Usage and Design Guidelines

This section provides guidelines for using the Virtex-4 FPGA PMCD.

Phase-Matched Divided Clocks

A PMCD produces binary-divided clocks that are rising-edge aligned to each other. From a clock input CLKA, the PMCD derives four output clocks: a clock with the same frequency as the original CLKA, ½, ¼, and $^1/_8$ the frequency. Figure 3-3 illustrates the input CLKA and the derived clocks (CLKA1, CLKA1D2, CLKA1D4, and CLKA1D8). CLKA1 is a delayed CLKA; thus, CLKA and CLKA1 are not deskewed. CLKA1D2, CLKA1D4, and CLKA1D8 are rising-edge aligned to CLKA1. CLKA1 reflects the duty cycle of CLKA. However, the divided clocks (CLKA1D2, CLKA1D4, and CLKA1D8) will have a 50/50 duty cycle regardless of the CLKA duty cycle.

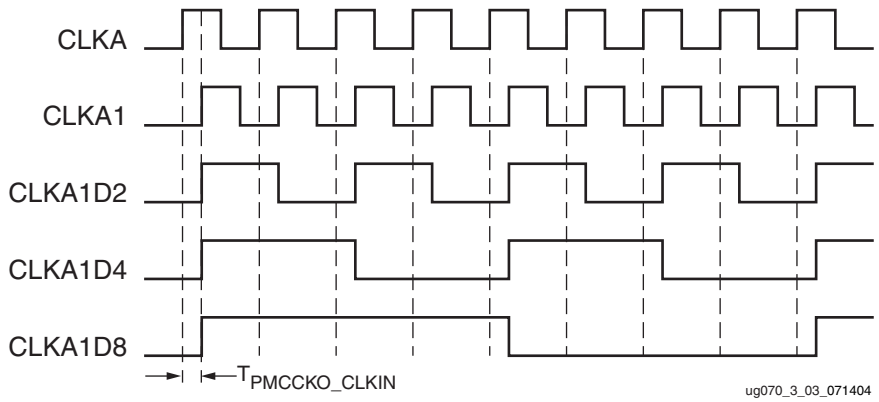


Figure 3-3: **PMCD Frequency Divider**

Matched Clock Phase

A PMCD allows three additional input clocks (CLKB, CLKC, CLKD) to pass through the same delay as CLKA. Thus, the corresponding clock outputs CLKB1, CLKC1, and CLKD1 maintain the same phase relation to each other as well as the CLKA outputs (CLKA1, CLKA1D2, CLKA1D4, CLKA1D6, and CLKA1D8) as their input. By matching the delay inserted to all inputs, a PMCD preserves the phase relation of its divided clock to other clocks in the design. Figure 3-4 illustrates CLKA, CLKB, CLKC, and CLKD with a 90° phase difference and the resulting PMCD outputs. CLKA1, CLKB1, CLKC1, and CLKD1 reflect the duty cycle of their corresponding input.

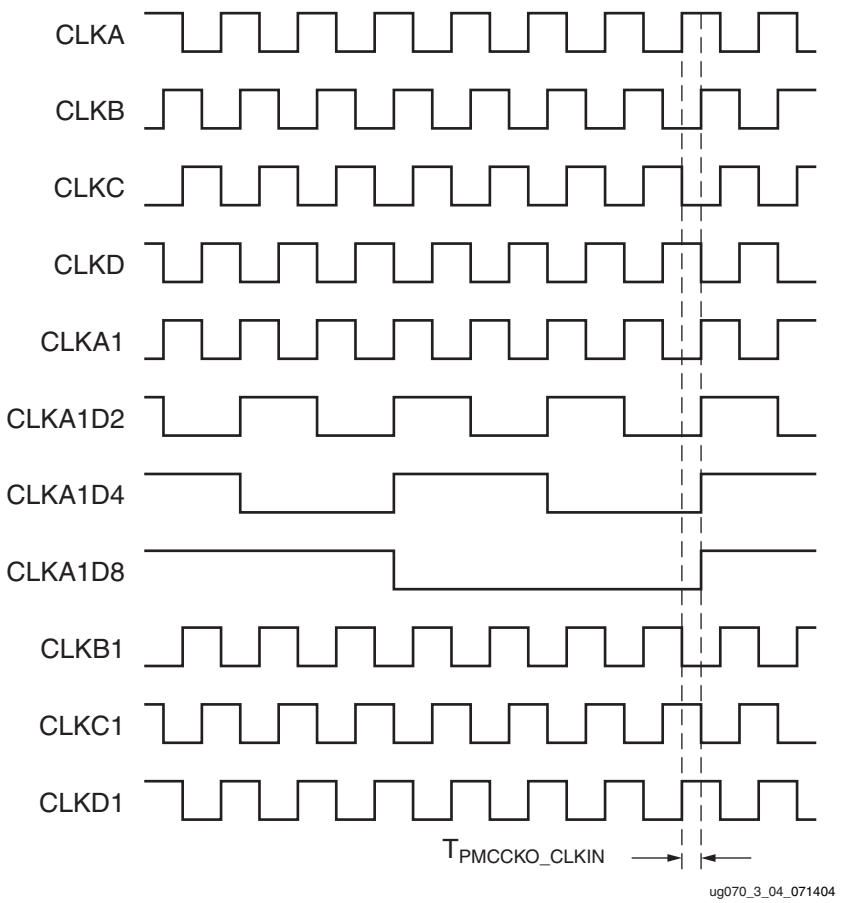


Figure 3-4: **Matched Clock Phase**

Reset (RST) and Release (REL) Control Signals

RST and REL are the control signals for the PMCD. The interaction between RST, REL, and the PMCD input clocks help manage the starting and stopping of PMCD outputs.

The reset (RST) signal affects the PMCD clock outputs in the following manner:

- Asserting RST asynchronously forces all outputs Low.
- Deasserting RST synchronously allows all outputs to toggle:
  - The delayed outputs begin toggling one cycle after RST is deasserted and is registered.
  - If EN_REL = FALSE (default), the divided outputs will also begin toggling one cycle after RST is deasserted and is registered.
  - If EN_REL = TRUE, then a positive edge on REL starts the divided outputs toggling on the next positive edge of CLKA.
- By setting the RST_DEASSERT_CLK attribute, deasserting RST can be synchronized to any of the four input clocks. The default value of RST_DEASSERT_CLK is CLKA (see Table 3-3).

Figure 3-5 illustrates an RST waveform when EN_REL = FALSE.

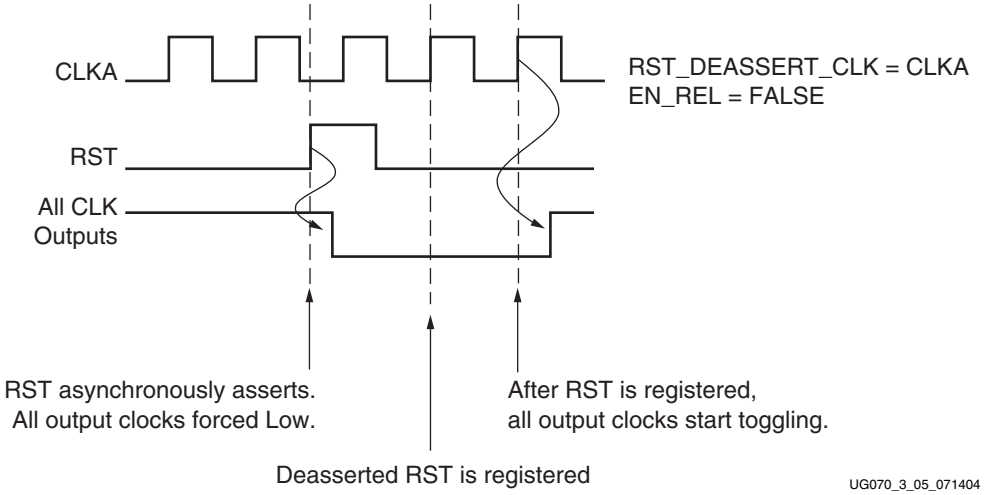


Figure 3-5: **RST Waveform Example**

The release (REL) signal affects PMCD outputs in the following manner:

- Asserting REL *synchronously* starts the divided outputs toggling. REL is synchronous to CLKA. Asserting REL must meet the setup time to CLKA.
- REL assertion does not affect the delayed clock outputs.
- REL is necessary when multiple PMCDs are used together and all PMCDs divided outputs should toggle in phase.
- REL is enabled with the EN_REL attribute. The default value of this attribute is FALSE.

  Set to TRUE only if multiple PMCDs are used together, or if other external synchronization is needed.
- RST must be Low before REL can have any effect.
- The REL input is positive edge sensitive.
- Once REL is asserted, the input has no further effect until another reset.

Figure 3-6 illustrates the interaction between the RST and REL signals.

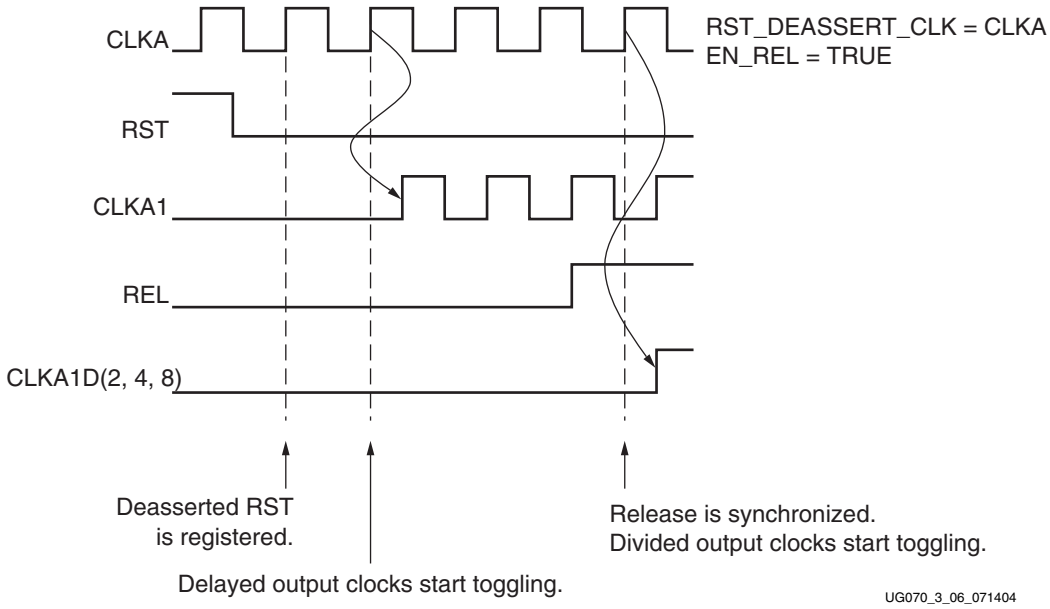


Figure 3-6: **REL Waveform Example**

Connecting PMCD to other Clock Resources

In most applications, the PMCD is used with other clock resources including dedicated clock I/O (IBUFG), clock buffers (BUFGCTRLs), DCMs, and an MGT clock. Additionally, PMCD inputs and outputs can be connected to the general interconnects. This section provides guidelines on connecting a PMCD to clock resources using dedicated routing.

IBUFG to PMCD

Virtex-4 devices contain 16 or 32 global clock I/Os. These clock I/Os are accessible by instantiating the IBUFG component. Each top and bottom half of the center column contains eight or 16 IBUFGs. Any of the IBUFGs in the top or bottom half can drive the clock input pins (CLKA, CLKB, CLKC, or CLKD) of a PMCD located in the same top/bottom half. The routing from multiple IBUFGs to PMCD inputs are not matched.

DCM to PMCD

Any DCM clock output can drive any PMCD input in the same top/bottom half of the device. A DCM can drive parallel PMCDs in the same group of two. It is not advisable to drive parallel PMCDs with DCMs in different groups of two (on the same top/bottom half) because there can be significant skew between PMCD outputs. This skew is caused by the skew between inputs of PMCDs in different groups.

BUFGCTRL to PMCD

Any BUFGCTRL can drive any Virtex-4 FPGA PMCD. However, only up to eight dedicated global clock routing resources exist in a particular clock region. Therefore, access to PMCD inputs via a BUFGCTRL is limited to eight unique signals. Other resources in the clock region will compete for the eight global clock tracks.

PMCD to BUFGCTRL

A PMCD can drive any BUFGCTRL in the same top/bottom half of the chip.

PMCD to PMCD

A dedicated local connection exists from the CLKA1D8 output of each PMCD to the CLKA input of any other PMCD within the same tile (group of two).

Application Examples

The Virtex-4 FPGA PMCD can be used in a variety of creative and useful applications. The following examples show some of the common applications.

DCM and a Single PMCD

A PMCD can be connected to a DCM to further divide a DCM clock. Figure 3-7 illustrates this example. Note the following guidelines:

- The DCM feedback (CLKFB) must be driven by the same frequency as CLKIN for 1X feedback. Therefore, the PMCD output corresponding to CLK0 must be used to drive the CLKFB pin.
- The RST_DEASSERT_CLK attribute must be set to the PMCD input driven by CLK0.
  - When a DCM is connected to a PMCD, all output clocks, except CLK0 and CLK2X, are held Low until LOCKED is High. Therefore, setting RST_DEASSERT_CLK to the corresponding DCM feedback clock ensures a completed feedback loop. Note: CLK2X feedback is not supported.

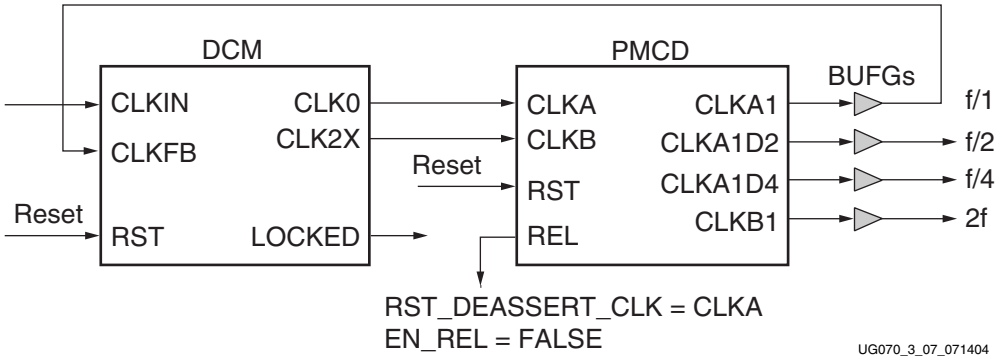


Figure 3-7: **DCM and a Single PMCD**

DCM and Parallel PMCDs

A DCM can be connected to parallel PMCDs. Figure 3-8 illustrates this example. Note the following guidelines:

- The DCM feedback (CLKFB) must be driven by the same frequency as CLKIN for 1X feedback. Therefore, the PMCD output corresponding to CLK0 must be used to drive the CLKFB pin.
- The RST_DEASSERT_CLK attribute must be set to the PMCD input driven by CLK0.
  - When a DCM is connected to a PMCD, all output clocks, except CLK0 and CLK2X, are held Low until LOCKED is High. Thus, setting RST_DEASSERT_CLK

to the corresponding DCM feedback clock ensures all PMCD outputs will start synchronously. Note: CLK2X feedback is not supported.

- The REL signals of the parallel PMCDs must be driven directly from the DCM LOCKED output.
  - Dedicated, timing-matched routes for both CLK signals and LOCKED signals exist from the DCMs to the PMCDs on the same top/bottom half of the device.
- To match output skews between two PMCDs, a DCM must connect to two PMCDs in the same tile (group of two).

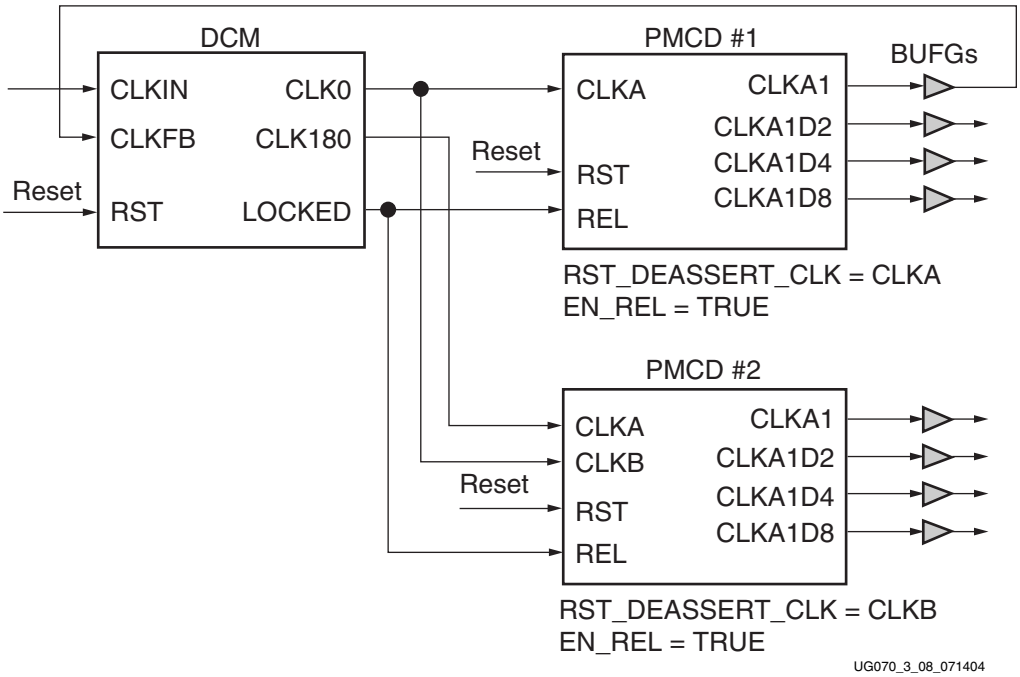


Figure 3-8: **DCM and Parallel PMCDs**

IBUFG, BUFG, and PMCD

When deskewed clocks are not required, a PMCD can be used without a DCM. Figure 3-9 and Figure 3-10 illustrate these examples.

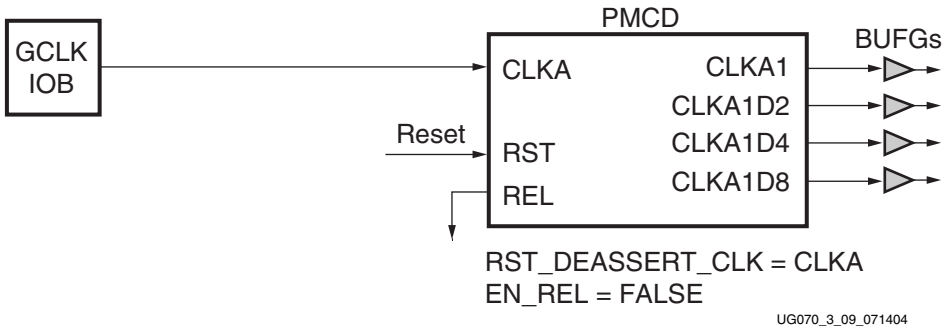


Figure 3-9: **PMCD Driven by IBUFG (GCLK IOB)**

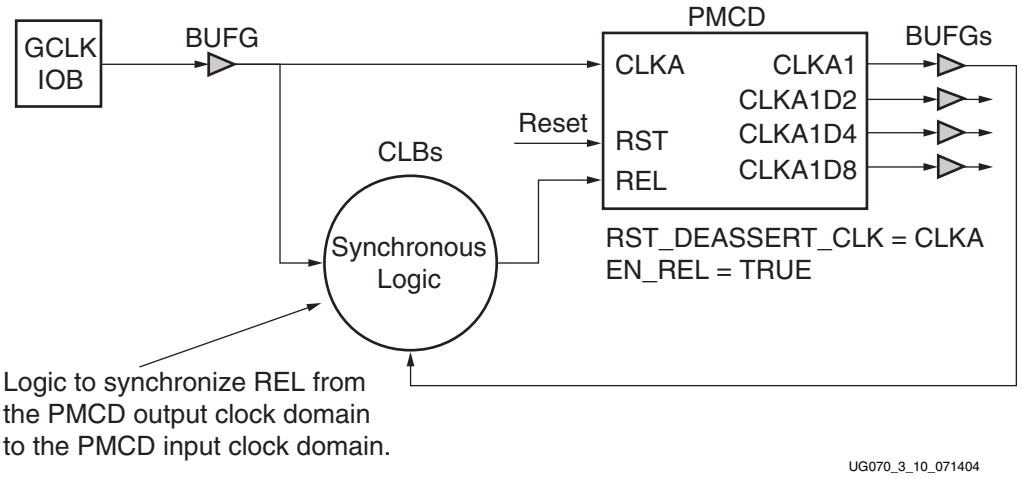


Figure 3-10: **PMCD Driven by BUFG and Synchronous Logic**

PMCD for Further Division of Clock Frequencies

PMCDs can be used to further divide clock frequencies. A dedicated local connection exists from the CLKA1D8 output of each PMCD to the CLKA input of the other PMCD within the same tile (group of two). Thus, only CLKA1D8 can directly connect two PMCDs in series.

Figure 3-11 illustrates an example of dividing clock frequencies using a DCM and a PMCD. Note the following guidelines:

- The CLKDV output is connected to CLKA of PMCD to allow further frequency division.
- The CLK0 feedback clock is connected to CLKB, and the RST_DEASSERT_CLK attribute is set to CLKB. These connections and settings ensure synchronous PMCD outputs.

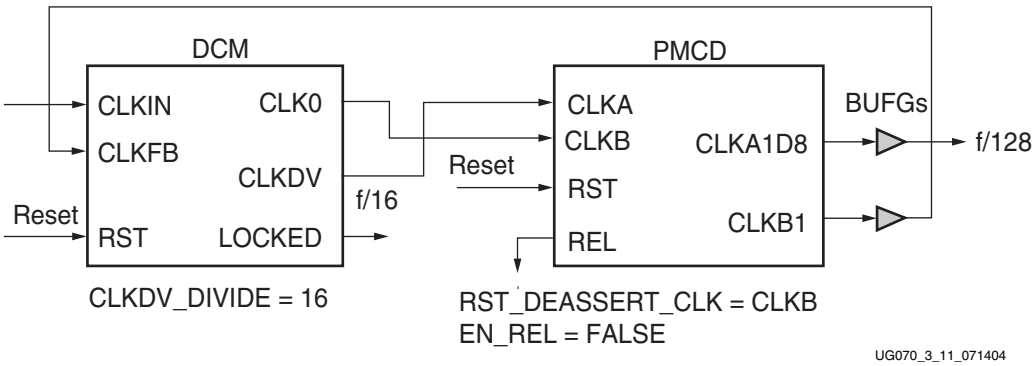


Figure 3-11: **DCM to PMCD for Clock Frequency Division**

Figure 3-12 illustrates an example of dividing clock frequencies using two PMCDs in series. Note the following guidelines:

- A dedicated local connection exists from the CLKA1D8 output of each PMCD to the CLKA and CLKB inputs of the other PMCD within the same tile (group of two). Thus, only CLKA1D8 can directly connect two PMCDs in series.

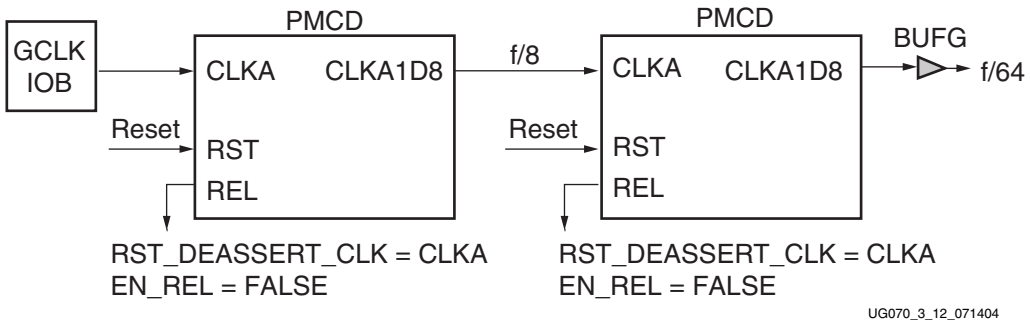


Figure 3-12: **PMCD to PMCD for Clock Frequency Division**

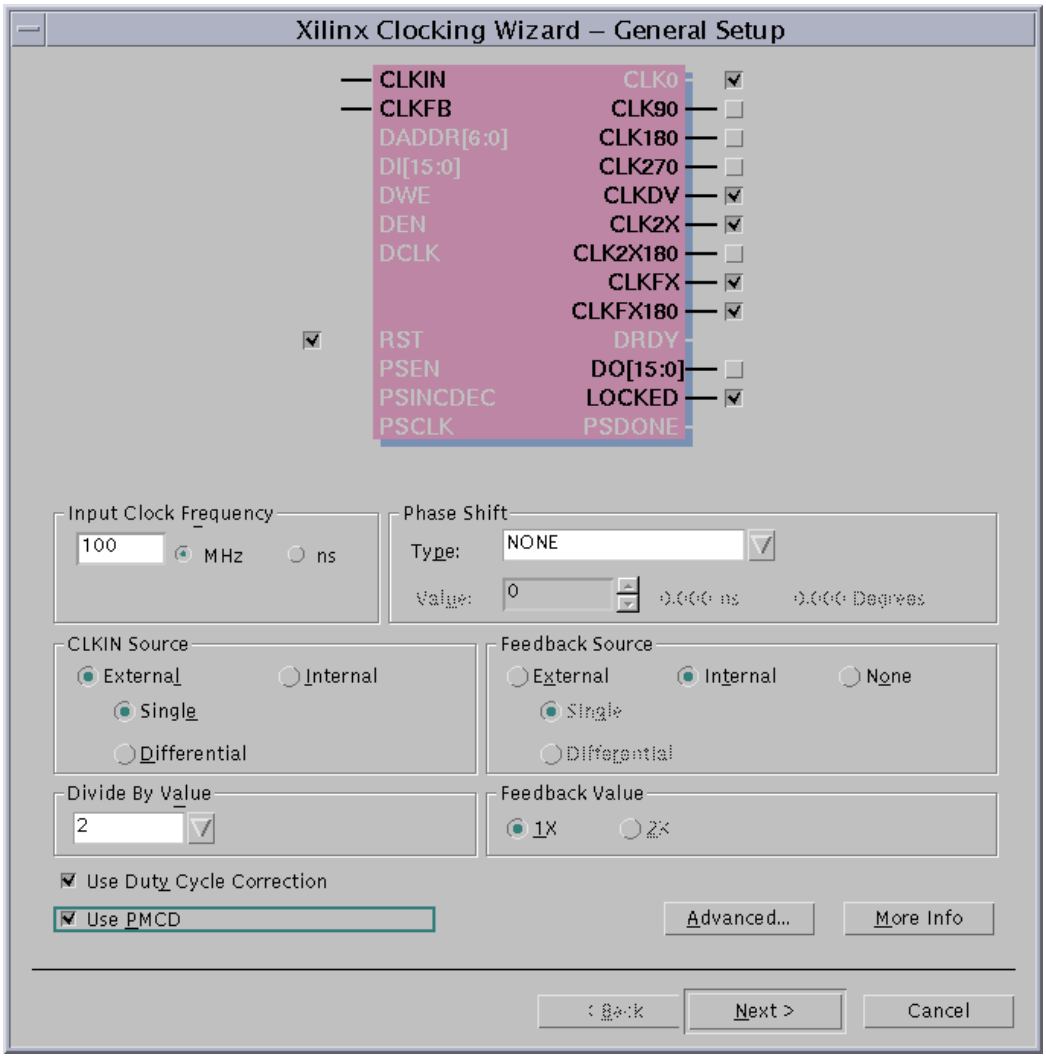
VHDL and Verilog Templates, and the Clocking Wizard

The "VHDL Template," page 111 and "Verilog Template," page 112 are also available in the Libraries Guide for all primitives. In addition, VHDL and Verilog files are generated by the Clocking Wizard in the ISE® software. The Clocking Wizard sets appropriate DCM and single/parallel PMCD configurations.

The Clocking Wizard is accessed using the ISE software, in the Project Navigator. Refer to the Xilinx® Software Manuals for more information on ISE software.

1. From the Project Navigator menu, select Project -> New Source. The *New Source* window appears.
2. Enter a file name and select IP (CoreGen and Architecture Wizard).
3. Click Next. The *Select Core Type* window appears.
4. Select Clocking -> Single DCM_ADV, click next. The *New Source Information* window appears.
5. Click Finish.
6. The Clocking Wizard starts.

Figure 3-13 and Figure 3-14 show the settings in the Clocking Wizard for using the DCM with the PMCD. To access further information on available settings, choose the *More Info* button in each window.



ug070_3_13_071204

Figure 3-13: **Xilinx Clocking Wizard - General Setup (PMCD)**

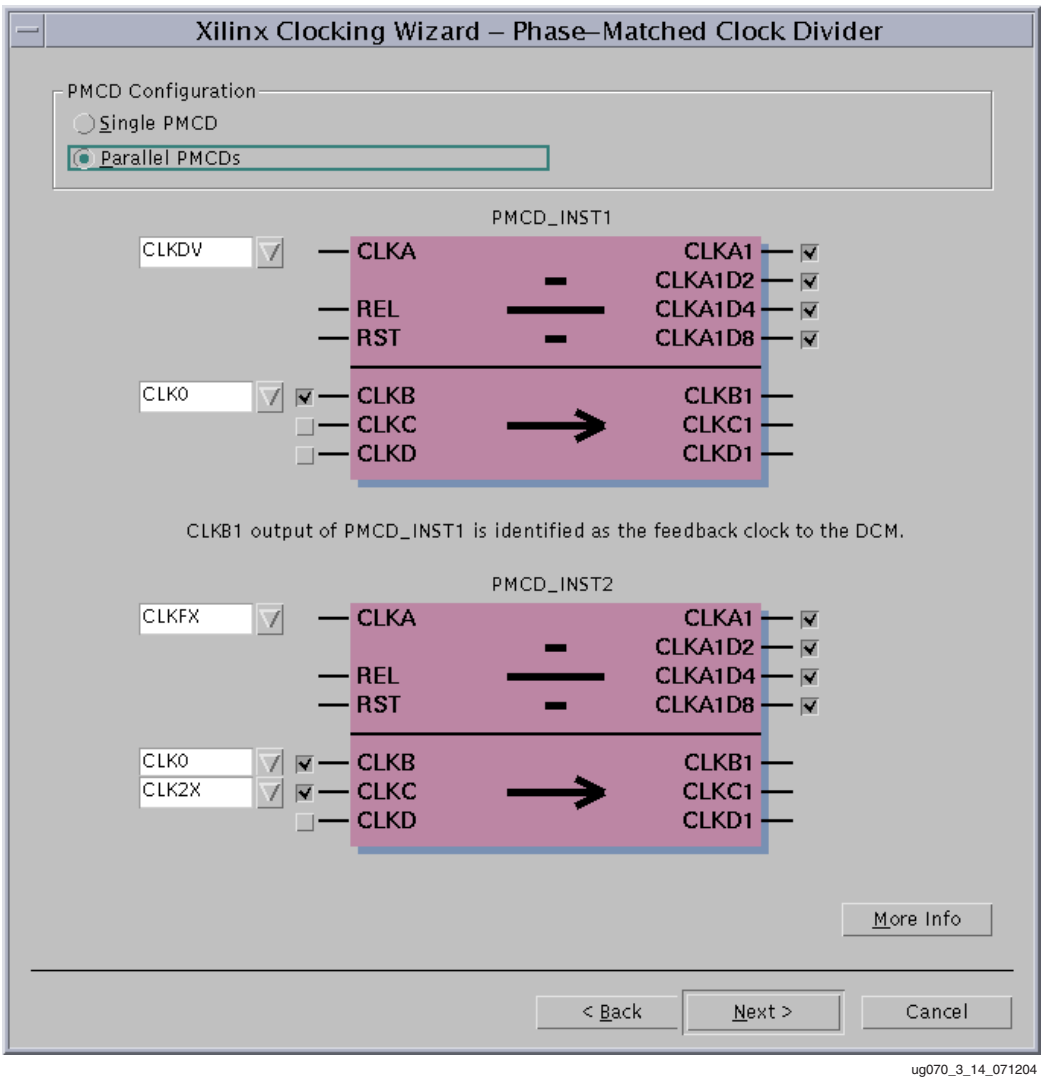


Figure 3-14: **Xilinx Clocking Wizard - Phase-Matched Clock Divider (PMCD)**

VHDL Template

```
-- Example PMCD Component Declaration

component PMCD
  generic(
      EN_REL           : boolean := FALSE;
      RST_DEASSERT_CLK : string  := "CLKA";
 );
  port(
      CLKA1      : out std_ulogic;
      CLKA1D2    : out std_ulogic;
      CLKA1D4    : out std_ulogic;
      CLKA1D8    : out std_ulogic;
      CLKB1      : out std_ulogic;
      CLKC1      : out std_ulogic;
      CLKD1      : out std_ulogic;
```

```
      CLKA       : in  std_ulogic;
      CLKB       : in  std_ulogic;
      CLKC       : in  std_ulogic;
      CLKD       : in  std_ulogic;
      REL        : in  std_ulogic;
      RST        : in  std_ulogic
    );
  end component;

--Example PMCD instantiation
U_PMCD : PMCD
Port map (
CLKA1 => user_clka1,
      CLKA1D2 => user_clka1d2,
      CLKA1D4 => user_clka1d4,
      CLKA1D8 => user_clka1d8,
      CLKB1   => user_clkb1,
      CLKC1   => user_clkc1,
      CLKD1   => user_clkd1,
      CLKA    => user_clka,
      CLKB    => user_clkb,
      CLKC    => user_clkc,
      CLKD    => user_clkd,
      REL     => user_rel,
      RST     => user_rst
);
```

Verilog Template

```
// Example PMCD module declaration

module PMCD (CLKA1, CLKA1D2, CLKA1D4, CLKA1D8, CLKB1, CLKC1, CLKD1,
CLKA, CLKB, CLKC, CLKD, REL, RST);

    output CLKA1;
    output CLKA1D2;
    output CLKA1D4;
    output CLKA1D8;
    output CLKB1;
    output CLKC1;
    output CLKD1;

    input CLKA;
    input CLKB;
    input CLKC;
    input CLKD;
    input REL;
    input RST;

    parameter EN_REL = "FALSE";
    parameter RST_DEASSERT_CLK = "CLKA";

endmodule;

//Example PMCD instantiation
PMCD U_PMCD (
.CLKA1(user_clka1),
.CLKA1D2(user_clka1d2),
```

```
.CLKA1D4(user_clka1d4),
.CLKA1D8(user_clka1d8),
.CLKB1(user_clkb1),
.CLKC1(user_clkc1),
.CLKD1(user_clkd1),
.CLKA(user_clka),
.CLKB(user_clkb),
.CLKC(user_clkc),
.CLKD(user_clkd),
.REL(user_rel),
.RST(user_rst)
);
```

XILINX®

Chapter 4

Block RAM

Block RAM Summary

The Virtex®-4 FPGA block RAMs are similar to the Virtex-II and Spartan™-3 FPGA block RAMs. Each block RAM stores 18 Kbits of data. Write and Read are synchronous operations; the two ports are symmetrical and totally independent, sharing only the stored data. Each port can be configured in any "aspect ratio" from 16Kx1, 8Kx2, to 512x36, and the two ports are independent even in this regard. The memory content can be defined or cleared by the configuration bitstream. During a write operation the data output can either reflect the new data being written, or the previous data now being overwritten, or the output can remain unchanged.

Virtex-4 FPGA enhancements of the basic block RAM include:

- The user can invoke a pipeline register at the data read output, still inside the block RAM. This allows a higher clock rate, at the cost of one additional clock period latency.
- Two adjacent block RAMs can be combined to one deeper 32Kx1 memory without any external logic or speed loss.
- Ports 18 or 36 bits wide can have individual write enable per byte. This feature is used for interfacing to an on-chip (PPC405) microprocessor.
- Each block RAM contains optional address sequencing and control circuitry to operate as a built-in Multi-rate FIFO memory. The FIFO can be 4K deep and 4 bits wide, or 2Kx9, 1Kx18, or 512x36. Write and read ports have identical width. The two free-running clocks can have completely unrelated frequencies (asynchronous relative to each other). Operation is controlled by the read and write enable inputs. FULL and EMPTY outputs signal the extreme conditions, without a possibility of errors or glitches. Programmable ALMOSTFULL and ALMOSTEMPTY outputs can be used for warning to simplify the external control of the write and read operation, especially the maximum clock rate.

Additional Virtex-4 FPGA block RAM features include:

- All output ports are latched. The state of the output port does not change until the port executes another read or write operation.
- All inputs are registered with the port clock and have a setup-to-clock timing specification.
- All outputs have a read function or a read-during-write function, depending on the state of the WE pin. The outputs are available after the clock-to-out timing interval. The read-during-write outputs have one of three operating modes: WRITE_FIRST, READ_FIRST, and NO_CHANGE.
- A write operation requires one clock edge.

- A read operation requires one clock edge.
- DO has an optional internal pipeline register.
- Data input and output signals are always described as buses; that is, in a 1-bit width configuration, the data input signal is DI[0] and the data output signal is DO[0].

Block RAM Introduction

In addition to distributed RAM, Virtex-4 devices feature a large number of 18 Kb block RAM memories. True Dual-Port™ RAM offers fast blocks of memory in the device. Block RAMs are placed in columns, and the total number of block RAM memory depends on the size of the Virtex-4 device. The 18 Kb blocks are cascadable to enable a deeper and wider memory implementation, with a minimal timing penalty.

Embedded dual- or single-port RAM modules, ROM modules, synchronous FIFOs, and data width converters are easily implemented using the Xilinx® CORE Generator™ software memory modules. Asynchronous FIFOs can be generated using the CORE Generator tool FIFO Generator module. The synchronous or asynchronous FIFO implementation does not require additional CLB resources for the FIFO control logic since it uses dedicated hardware resources.

Synchronous Dual-Port and Single-Port RAMs

Data Flow

The 18 Kb block RAM dual-port memory consists of an 18 Kb storage area and two completely independent access ports, A and B. The structure is fully symmetrical, and both ports are interchangeable. Figure 4-1 illustrates the dual-port data flow. Table 4-1 lists the port names and descriptions.

Data can be written to either or both ports and can be read from either or both ports. Each write operation is synchronous, each port has its own address, data in, data out, clock, clock enable, and write enable. The read operation is synchronous and requires a clock edge.

There is no dedicated monitor to arbitrate the effect of identical addresses on both ports. It is up to the user to time the two clocks appropriately. However, conflicting simultaneous writes to the same location never cause any physical damage.

When a block RAM port is enabled, all address transitions must meet the setup/hold time of the ADDR inputs with respect to the port clock, as listed in the *Virtex-4 Data Sheet*. The requirement must be met even when the read data output is of no interest and ignored by the user.

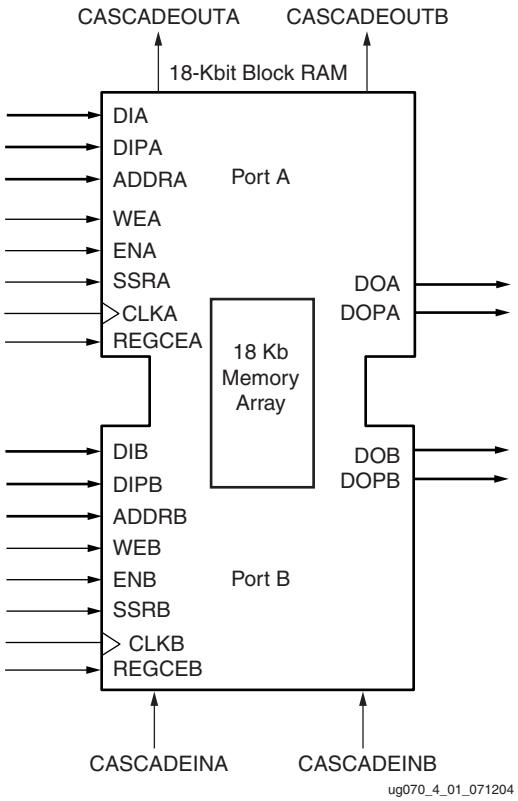


Figure 4-1: **Dual-Port Data Flows**

Table 4-1: **Dual-Port Names and Descriptions**

Port Name	Description
DI[A | B]	Data Input Bus
DIP[A | B](1)	Data Input Parity Bus
ADDR[A | B]	Address Bus
WE[A | B]	Write Enable
EN[A | B]	When inactive no data is written to the block RAM and the output bus remains in its previous state.
SSR[A | B]	Set/Reset
CLK[A | B]	Clock Input
DO[A | B]	Data Output Bus
DOP[A | B](1)	Data Output Parity Bus
REGCE[A | B]	Output Register Enable
CASCADEIN[A | B]	Cascade input pin for 32K x 1 mode
CASCADEOUT[A | B]	Cascade output pin for 32K x 1 mode

Notes:

1. The "Data Parity Buses - DIP[A/B] and DOP[A/B]" section has more information on Data Parity pins.

Read Operation

The read operation uses one clock edge. The read address is registered on the read port, and the stored data is loaded into the output latches after the RAM access time.

Write Operation

A write operation is a single clock-edge operation. The write address is registered on the write port, and the data input is stored in memory.

Operating Modes

There are three modes of a write operation. The "read during write" mode offers the flexibility of using the data output bus during a write operation on the same port. Output mode is set during device configuration. These choices increase the efficiency of block RAM memory at each clock cycle.

Three different modes are used to determine data available on the output latches after a write clock edge: WRITE_FIRST, READ_FIRST, and NO_CHANGE.

Mode selection is set by configuration. One of these three modes is set individually for each port by an attribute. The default mode is WRITE_FIRST.

WRITE_FIRST or Transparent Mode (Default)

In WRITE_FIRST mode, the input data is simultaneously written into memory and stored in the data output (transparent write), as shown in Figure 4-2.

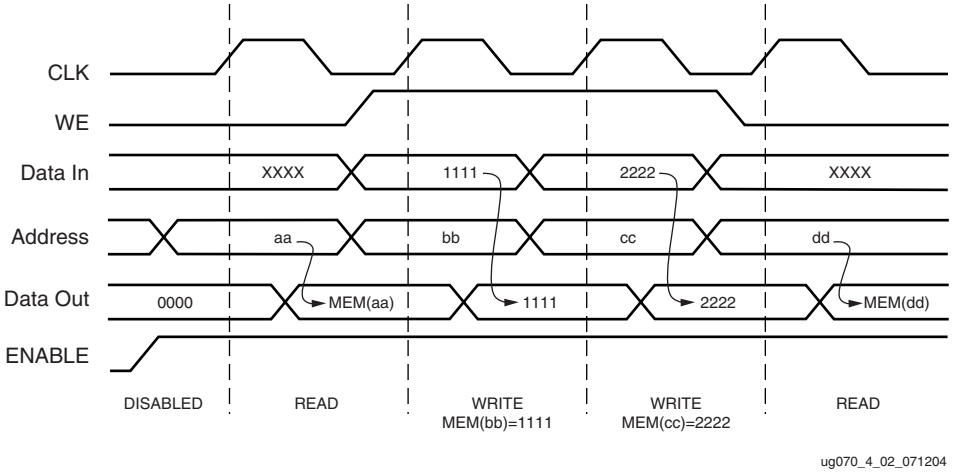


Figure 4-2: **WRITE_FIRST Mode Waveforms**

READ_FIRST or READ-BEFORE-WRITE Mode

In READ_FIRST mode, data previously stored at the write address appears on the output latches, while the input data is being stored in memory (read before write). See Figure 4-3.

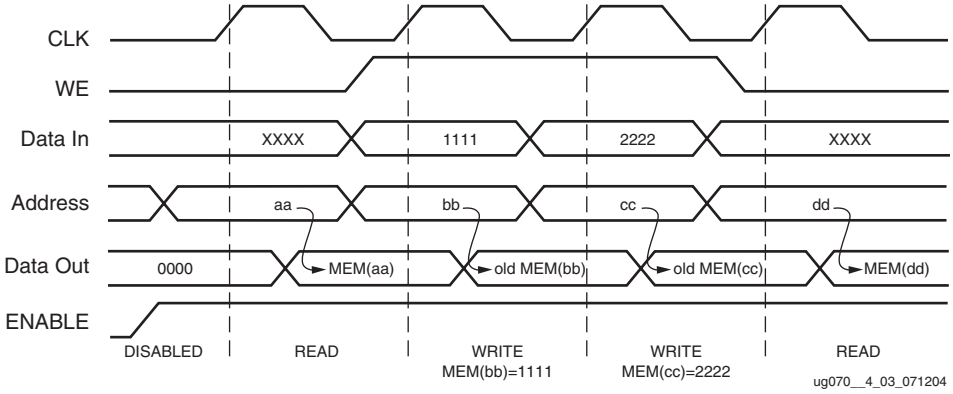


Figure 4-3: **READ_FIRST Mode Waveforms**

NO_CHANGE Mode

In NO_CHANGE mode, the output latches remain unchanged during a write operation. As shown in Figure 4-4, data output is still the last read data and is unaffected by a write operation on the same port. NO_CHANGE mode is not supported in 32K x 1 RAM configuration.

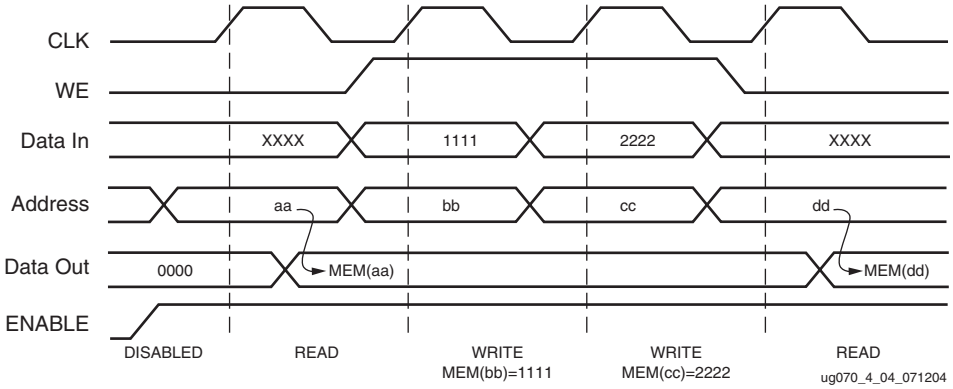


Figure 4-4: **NO_CHANGE Mode Waveforms**

Conflict Avoidance

Virtex-4 FPGA block RAM is a true dual-port RAM, where both ports can access any memory location at any time. When accessing the same memory location from both ports, the user must, however, observe certain restrictions, specified by the clock-to-clock setup time window. There are two fundamentally different situations: The two ports either have a common clock (“synchronous clocking”), or the clock frequency or phase is different for the two ports (“asynchronous clocking”).

Asynchronous Clocking

Asynchronous clocking is the more general case, where the active edges of both clocks do not occur simultaneously:

- There are no timing constraints when both ports perform a read operation.
- When one port performs a write operation, the other port must not read- or write-access the same memory location by using a clock edge that falls within the specified forbidden clock-to-clock setup time window. If this restriction is ignored, a read operation could read unreliable data, perhaps a mixture of old and new data in this location; a write operation could result in wrong data stored in this location. There is, however, no risk of physical damage to the device. If a read and write operation is performed, the write will store valid data at the write location.

The clock-to-clock setup timing parameter is specified together with other block RAM switching characteristics in the *Virtex-4 Data Sheet*.

Synchronous Clocking

Synchronous clocking is the special case, where the active edges of both port clocks occur simultaneously:

- There are no timing constraints when both ports perform a read operation.
- When one port performs a write operation, the other port must not write into the same location, unless both ports write identical data.
- When one port performs a write operation, the write operation succeeds; the other port can reliably read data from the same location if the write port is in READ_FIRST mode. DATA_OUT will then reflect the previously stored data.

  If the write port is in either WRITE_FIRST or in NO_CHANGE mode, then the DATA-OUT on the read port would become invalid (unreliable). Obviously, the mode setting of the read-port does not affect this operation.

Additional Block RAM Features in Virtex-4 Devices

Optional Output Registers

The optional output registers improve design performance by eliminating routing delay to the CLB flip-flops for pipelined operation. These output registers have programmable clock inversion as in CLB flip-flops. An independent clock enable input is provided for these output registers. As a result the output data registers hold the value independent of the input register operation. Figure 4-5 shows the optional output register.

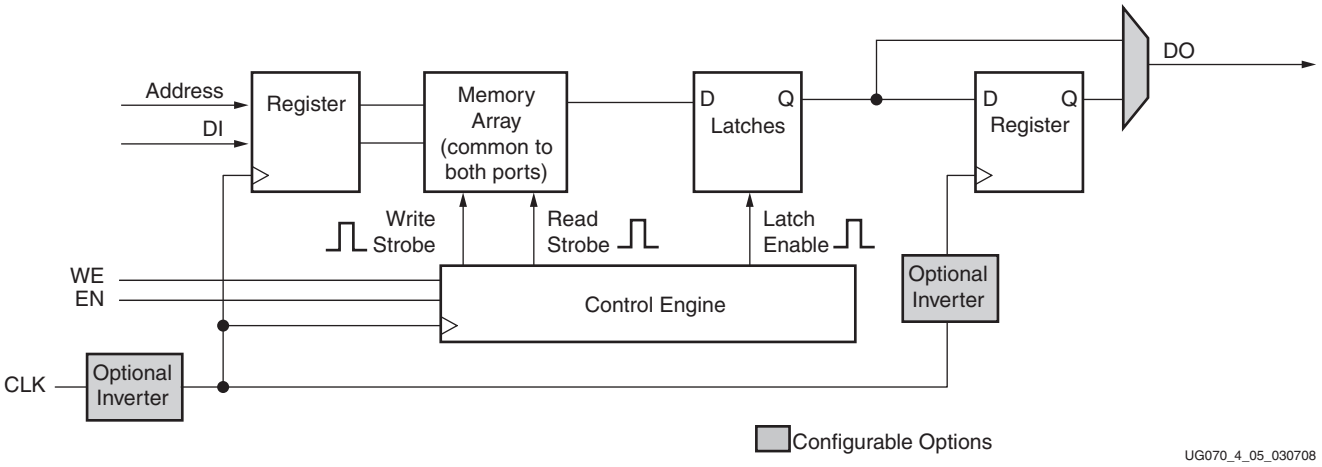


Figure 4-5: **Block RAM Logic Diagram (One Port Shown)**

Independent Read and Write Port Width Selection

All block RAM ports have control over data width and address depth (aspect ratio). Virtex-4 devices extend this flexibility to each individual port where Read and Write can be configured with different data widths. See "Block RAM Attributes," page 127.

If the Read port width differs from the Write port width, and is configured in WRITE_FIRST mode, then DO shows valid new data only if all the write bytes are enabled.

Independent Read and Write port width selection increases the efficiency of implementing a content addressable memory (CAM) in block RAM. Excluding the built-in FIFO, this option is available for all RAM port sizes and modes.

Cascadable Block RAM

Combining two 16K x 1 RAMs to form one 32K x 1 RAM is possible in the Virtex-4 block RAM architecture without using local interconnect or additional CLB logic resources. NO_CHANGE mode is not supported in 32K x 1 RAM configuration. Any two adjacent block RAMs can be cascaded to generate a 32K x 1 block RAM. Increasing the depth of the block RAM by cascading two block RAMs is available only in the 32K x 1 mode. Further information on cascadable block RAM is described in the "Additional RAMB16 Primitive Design Considerations" section. For other wider and/or deeper sizes, consult the Creating Larger RAM Structures section. Figure 4-6 shows the block RAM with the appropriate ports connected in the Cascadable mode. The "Additional Block RAM Features in Virtex-4 Devices" section includes further information on cascadable block RAMs.

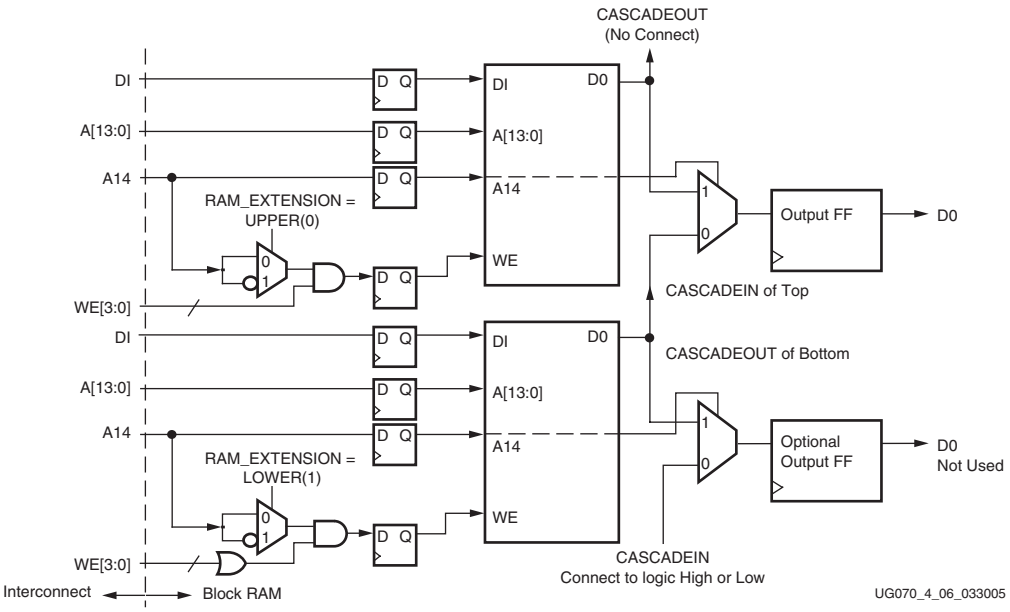


Figure 4-6: **Cascadable Block RAM**

FIFO Support

The block RAM can be configured as an asynchronous FIFO (different clock on read and write ports) or a synchronous FIFO. In the FIFO mode, the block RAM Port A is the FIFO read port, while the block RAM Port B is the FIFO write port. The supported configurations are: 4K x 4, 2K x 9, 1K x 18, and 512 x 36. Figure 4-7 shows the block RAM I/Os used for the FIFO implementation. The "Built-in FIFO Support" section contains further details.

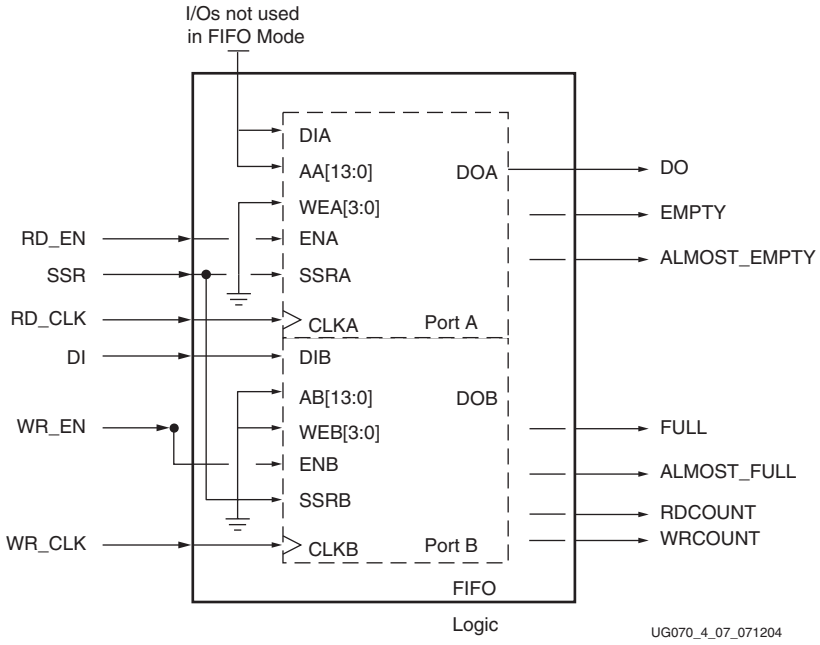


Figure 4-7: **Block RAM Implemented as a FIFO**

Byte-Wide Write Enable

The byte-wide write enable feature of the block RAM gives the capability to write eight bit (one byte) portions of incoming data. There are four independent byte-write enable inputs. Each byte-write enable is associated with one byte of input data and one parity bit. All four byte-write enable inputs must be driven in all data width configurations. This feature is useful when using block RAM to interface with the PPC405. Byte-write enable is not available in the Multi-rate FIFO. Byte-write enable is further described in the "Additional RAMB16 Primitive Design Considerations" section. Figure 4-8 shows the byte-wide write-enable logic.

When configured for a 36-bit or 18-bit wide data path, any port can restrict writing to specified byte locations within the data word. If configured in READ_FIRST mode, the DO bus shows the previous content of the whole addressed word. In WRITE_FIRST mode, with identical Read and Write port widths, DO shows only the enabled newly written byte(s). The other byte values must be ignored. In WRITE_FIRST mode with different widths for Read and Write ports, all data on DO must be ignored.

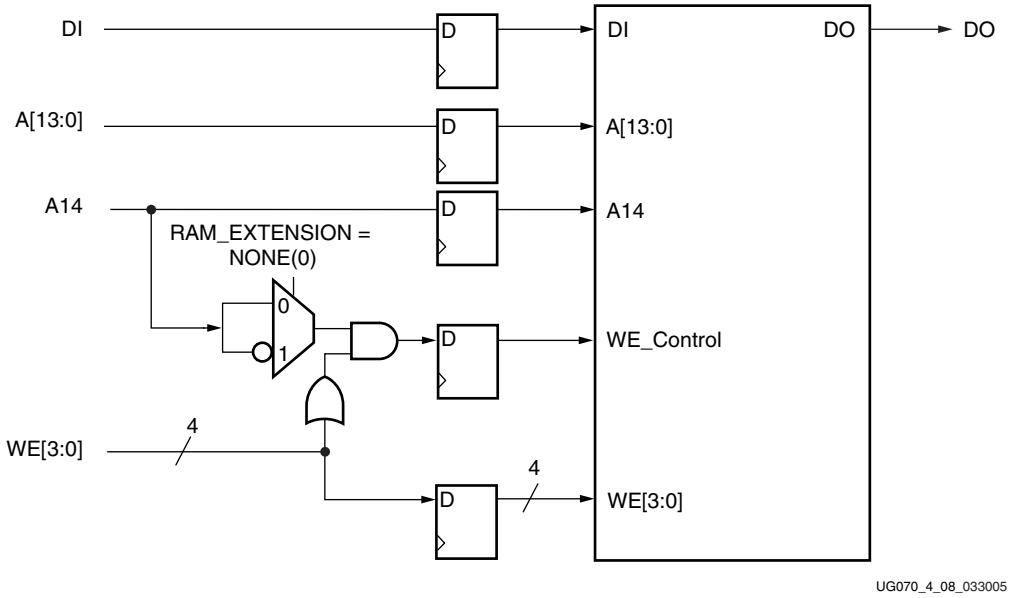


Figure 4-8: **Byte-Wide Write Enable In Block RAM**

Block RAM Library Primitives

RAMB16 is the block RAM library primitive. It is the basic building block for all block RAM configurations. Other block RAM primitives and macros are based on this primitive. Some block RAM attributes can only be configured using this primitive (e.g., pipeline register, cascade). See "Block RAM Attributes," page 127.

Figure 4-9 illustrates all the I/O ports of the block RAM primitive (RAMB16).

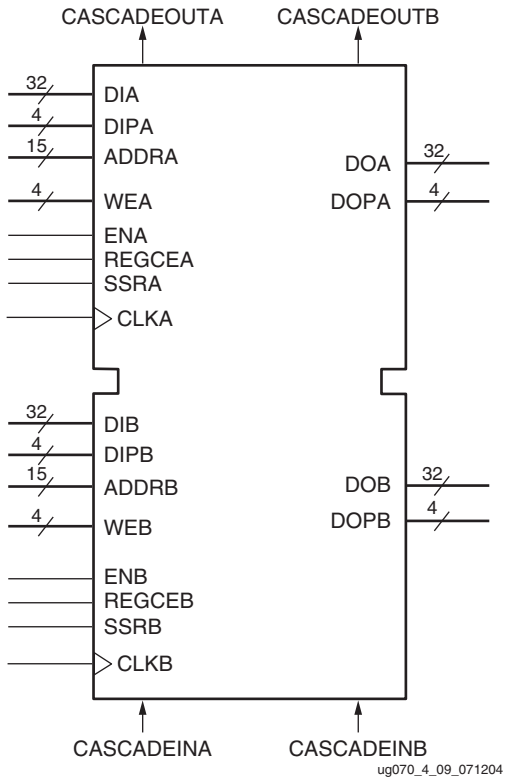


Figure 4-9: **Block RAM Port Signals (RAMB16)**

Block RAM Port Signals

Each block RAM port operates independently of the other while accessing the same set of 18 Kbit memory cells.

Clock - CLK[A|B]

Each port is fully synchronous with independent clock pins. All port input pins have setup time referenced to the port CLK pin. The output data bus has a clock-to-out time referenced to the CLK pin. Clock polarity is configurable (rising edge by default).

Enable - EN[A|B]

The enable pin affects the read, write, and set/reset functionality of the port. Ports with an inactive enable pin keep the output pins in the previous state and do not write data to the memory cells. Enable polarity is configurable (active High by default).

Write Enable - WE[A|B]

To write the content of the data input bus into the addressed memory location, both EN and WE must be active within a setup time before the active clock edge. The output latches are loaded or not loaded according to the write configuration (WRITE_FIRST, READ_FIRST, NO_CHANGE). When inactive, a read operation occurs, and the contents of the memory cells referenced by the address bus reflect on the data-out bus, regardless of the write mode attribute. Write enable polarity is configurable (active High by default).

Register Enable - REGCE[A|B]

The register enable pin (REGCE) controls the optional output register. When the RAM is in register mode, REGCE = 1 registers the output into a register at a clock edge. The polarity of REGCE is configurable (active High by default).

Set/Reset - SSR[A|B]

The SSR pin forces the data output latches to contain the value "SRVAL" (see "Block RAM Attributes," page 127). The data output latches are synchronously asserted to 0 or 1, including the parity bit. In a 36-bit width configuration, each port has an independent SRVAL[A | B] attribute of 36 bits. This operation does not affect RAM cells and does not disturb write operations on the other port. Similar to the read and write operation, the set/reset function is active only when the enable pin of the port is active. Set/reset polarity is configurable (active High by default). This pin is not available when optional output registers are used.

Address Bus - ADDR[A|B]<14:#>

The address bus selects the memory cells for read or write. The width of the port determines the required address bus width for a single RAMB16, as shown in Table 4-2.

Table 4-2: **Port Aspect Ratio**

Port Data Width	Port Address Width	Depth	ADDR Bus	DI Bus / DO Bus	DIP Bus / DOP Bus
1	14	16,384	<13:0>	<0>	NA
2	13	8,192	<13:1>	<1:0>	NA
4	12	4,096	<13:2>	<3:0>	NA
9	11	2,048	<13:3>	<7:0>	<0>
18	10	1,024	<13:4>	<15:0>	<1:0>
36	9	512	<13:5>	<31:0>	<3:0>

For cascadable block RAM, the data width is one bit, however, the address bus is 15 bits <14:0>. The address bit 15 is only used in cascadable block RAM.

Data and address pin mapping is further described in the "Additional RAMB16 Primitive Design Considerations"section.

Data-In Buses - DI[A|B]<#:0> & DIP[A|B]<#:0>

Data-in buses provide the new data value to be written into RAM. The regular data-in bus (DI), and the parity data-in bus (DIP) when available, have a total width equal to the port

width. For example the 36-bit port data width is represented by DI<31:0> and DIP<3:0>, as shown in Table 4-2.

Data-Out Buses - DO[A|B]<#:0> and DOP[A|B]<#:0>

Data-out buses reflect the contents of memory cells referenced by the address bus at the last active clock edge during a read operation. During a write operation (WRITE_FIRST or READ_FIRST configuration), the data-out buses reflect either the data-in buses or the stored value before write. During a write operation in NO_CHANGE mode, data-out buses are not affected. The regular data-out bus (DO) and the parity data-out bus (DOP) (when available) have a total width equal to the port width, as shown in Table 4-2.

Cascade - CASCADEIN[A|B]

The CASCADEIN pins are used to connect two block RAMs to form the 32K x 1 mode. This pin is used when the block RAM is the UPPER block RAM, and is connected to the CASCADEOUT pins of the LOWER block RAM. When cascade mode is not used, this pin does not need to be connected. Refer to the "Cascadable Block RAM" for further information.

Cascade - CASCADEOUT[A|B]

The CASCADEOUT pins are used to connect two block RAMs to form the 32K x 1 mode. This pin is used when the block RAM is the LOWER block RAM, and is connected to the CASCADEIN pins of the UPPER block RAM. When cascade mode is not used, this pin does not need to be connected. Refer to the "Cascadable Block RAM" for further information.

Inverting Control Pins

For each port, the five control pins (CLK, EN, WE, REGCE, and SSR) each have an individual inversion option. Any control signal can be configured as active High or Low, and the clock can be active on a rising or falling edge (active High on rising edge by default) without requiring other logic resources.

GSR

The global set/reset (GSR) signal of a Virtex-4 device is an asynchronous global signal that is active at the end of device configuration. The GSR can also restore the initial Virtex-4 FPGA state at any time. The GSR signal initializes the output latches to the INIT, or to the INIT_A and INIT_B value (see "Block RAM Attributes"). A GSR signal has no impact on internal memory contents. Because it is a global signal, the GSR has no input pin at the functional level (block RAM primitive).

Unused Inputs

Unused Data and/or address inputs should be tied High.

Block RAM Address Mapping

Each port accesses the same set of 18,432 memory cells using an addressing scheme dependent on the width of the port. The physical RAM locations addressed for a particular width are determined using the following formula (of interest only when the two ports use different aspect ratios):

```
END = ((ADDR + 1) * Width) - 1
START= ADDR * Width
```

Table 4-3 shows low-order address mapping for each port width.

Table 4-3: **Port Address Mapping**

Port Width	Parity Locations	Data Locations
1	N.A.	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
2		15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
4		7 6 5 4 3 2 1 0
8 + 1	3 2 1 0	3 2 1 0
16 + 2	1 0	1 0
32 + 4	0	0

Block RAM Attributes

All attribute code examples are shown in the "Block RAM VHDL and Verilog Templates" section. Further information on using these attributes is available in the "Additional RAMB16 Primitive Design Considerations" section.

Content Initialization - INIT_xx

INIT_xx attributes define the initial memory contents. By default block RAM memory is initialized with all zeros during the device configuration sequence. The 64 initialization attributes from INIT_00 through INIT_3F represent the regular memory contents. Each INIT_xx is a 64-digit hex-encoded bit vector. The memory contents can be partially initialized and are automatically completed with zeros.

The following formula is used for determining the bit positions for each INIT_xx attribute.

Given yy = conversion hex-encoded to decimal (xx), INIT_xx corresponds to the memory cells as follows:

- from [(yy + 1) * 256] – 1
- to (yy) * 256

For example, for the attribute INIT_1F, the conversion is as follows:

- yy = conversion hex-encoded to decimal X"1F" = 31
- from [(31+1) * 256] – 1 = 8191
- to 31 * 256 = 7936

More examples are given in Table 4-4.

Table 4-4: **Block RAM Initialization Attributes**

Attribute	Memory Location	
	From	To
INIT_00	255	0
INIT_01	511	256
INIT_02	767	512
…	…	…
INIT_0E	3839	3584
INIT_0F	4095	3840
INIT_10	4351	4096
…	…	…
INIT_1F	8191	7936
INIT_20	8447	8192
…	…	…
INIT_2F	12287	12032
INIT_30	12543	12288
..	…	…
INIT_3F	16383	16128

Content Initialization - INITP_xx

INITP_xx attributes define the initial contents of the memory cells corresponding to DIP/DOP buses (parity bits). By default these memory cells are also initialized to all zeros. The eight initialization attributes from INITP_00 through INITP_07 represent the memory contents of parity bits. Each INITP_xx is a 64-digit hex-encoded bit vector with a regular INIT_xx attribute behavior. The same formula can be used to calculate the bit positions initialized by a particular INITP_xx attribute.

Output Latches Initialization - INIT (INIT_A & INIT_B)

The INIT (single-port) or INIT_A and INIT_B (dual-port) attributes define the output latches values after configuration. The width of the INIT (INIT_A & INIT_B) attribute is the port width, as shown in Table 4-5. These attributes are hex-encoded bit vectors, and the default value is 0.

Output Latches Synchronous Set/Reset - SRVAL (SRVAL_A & SRVAL_B)

The SRVAL (single-port) or SRVAL_A and SRVAL_B (dual-port) attributes define output latch values when the SSR input is asserted. The width of the SRVAL (SRVAL_A and SRVAL_B) attribute is the port width, as shown in Table 4-5. These attributes are hex-encoded bit vectors, and the default value is 0. This attribute is not available when the optional output register attribute is set.

Table 4-5: **Port Width Values**

Port Data Width	DOP Bus	DO Bus	INIT / SRVAL
1	NA	<0>	1
2	NA	<1:0>	2
4	NA	<3:0>	4
9	<0>	<7:0>	(1 + 8) = 9
18	<1:0>	<15:0>	(2 + 16) = 18
36	<3:0>	<31:0>	(4 + 32) = 36

Optional Output Register On/Off Switch - DO[A|B]_REG

This attribute sets the number of pipeline register at A/B output of RAMB16. The valid values are 0 (default) or 1.

Clock Inversion at Output Register Switch - INVERT_CLK_DO[A|B]_REG

When set to TRUE, the clock input to the pipeline register at A/B output of RAMB16 is inverted. The default value is FALSE.

Extended Mode Address Determinant - RAM_EXTENSION_[A|B]

This attribute determines whether the block RAM of interest has its A/B port as UPPER/LOWER address when using the cascade mode. Refer to the "Cascadable Block RAM" section. When the block RAM is not used in cascade mode, the default value is NONE.

Read Width - READ_WIDTH_[A|B]

This attribute determines the A/B read port width of the block RAM. The valid values are: 0 (default), 1, 2, 4, 9, 18, and 36.

Write Width - WRITE_WIDTH_[A|B]

This attribute determines the A/B write port width of the block RAM. The valid values are: 0 (default), 1, 2, 4, 9, 18, and 36.

Write Mode - WRITE_MODE_[A|B]

This attribute determines the write mode of the A/B input ports. The possible values are WRITE_FIRST (default), READ_FIRST, and NO_CHANGE. Additional information on the write modes is in the "Operating Modes" section.

Block RAM Location Constraints

Block RAM instances can have LOC properties attached to them to constrain placement. Block RAM placement locations differ from the convention used for naming CLB locations, allowing LOC properties to transfer easily from array to array.

The LOC properties use the following form:

```
LOC = RAMB16_X#Y#
```

The RAMB16_X0Y0 is the bottom-left block RAM location on the device. If RAMB16 is constrained to RAMB16_X#Y#, the FIFO cannot be constrained to FIFO16_X#Y# since they share a location. An example location constraint is shown in the "Block RAM VHDL and Verilog Templates" section.

Block RAM Initialization in VHDL or Verilog Code

Block RAM memory attributes and content can be initialized in VHDL or Verilog code for both synthesis and simulation by using generic maps (VHDL) or defparams (Verilog) within the instantiated component. Modifying the values of the generic map or defparam affects both the simulation behavior and the implemented synthesis results.

Block RAM VHDL and Verilog Templates

The following template is a RAMB16 example in both VHDL and Verilog. This primitive is the building block for all different sizes of block RAM.

RAMB16 VHDL Template

```
--    RAMB16     : To incorporate this function into the design,
--     VHDL      : following instance declaration needs to be placed in
--   instance    : the architecture body of the design code. The
--  declaration  : (RAMB16_inst) and/or the port declarations
--     code      : after the "=>" assignment can be changed to properly
--               : reference and connect this function to the design.
--               : All inputs and outputs must be connected.
--    Library    : In addition to adding the instance declaration, a use
--  declaration  : statement for the UNISIM.v components library needs
--     for       : to be added before the entity declaration. This library
--    Xilinx     : contains the component declarations for all Xilinx
--   primitives  : primitives and points to the models that will be used
--               : for simulation.

--  Copy the following two statements and paste them before the
--  Entity declaration, unless they already exist.

Library UNISIM;
use UNISIM.vcomponents.all;

--  <--Cut code below this line and paste into the architecture body-->

   -- RAMB16: Virtex-4 16k+2k Parity Paramatizable Block RAM
   -- Virtex-4 FPGA User Guide

   RAMB16_inst : RAMB16
   generic map (
      DOA_REG => 0, -- Optional output registers on the A port (0 or 1)
      DOB_REG => 0, -- Optional output registers on the B port (0 or 1)
      INIT_A => X"000000000", --  Initial values on A output port
      INIT_B => X"000000000", --  Initial values on B output port
      INVERT_CLK_DOA_REG => FALSE, -- Invert clock on A port output
      registers (TRUE or FALSE)
```

```
      INVERT_CLK_DOB_REG => FALSE, -- Invert clock on B port output
      registers (TRUE or FALSE)
      RAM_EXTENSION_A => "NONE", -- "UPPER", "LOWER" or "NONE" when
      cascaded
      RAM_EXTENSION_B => "NONE", -- "UPPER", "LOWER" or "NONE" when
      cascaded
      READ_WIDTH_A => 0, -- Valid values are 1,2,4,9,18 or 36
      READ_WIDTH_B => 0, -- Valid values are 1,2,4,9,18 or 36
      SRVAL_A => X"000000000", --  Port A ouput value upon SSR assertion
     SRVAL_B => X"000000000", --  Port B ouput value upon SSR assertion
     WRITE_MODE_A => "WRITE_FIRST", --  "WRITE_FIRST", "READ_FIRST" or
      "NO_CHANGE"
     WRITE_MODE_B => "WRITE_FIRST", --  "WRITE_FIRST", "READ_FIRST" or
      "NO_CHANGE"
      WRITE_WIDTH_A => 2, -- Valid values are 1,2,4,9,18 or 36
      WRITE_WIDTH_B => 0, -- Valid values are 1,2,4,9,18 or 36
      INIT_00 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_01 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_02 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_03 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_04 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_05 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_06 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_07 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_08 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_09 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_0A =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_0B =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_0C =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_0D =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_0E =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_0F =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_10 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_11 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_12 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_13 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_14 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
```

```
      INIT_15 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_16 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_17 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_18 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_19 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_1A =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_1B =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_1C =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_1D =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_1E =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_1F =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_20 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_21 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_22 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_23 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_24 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_25 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_26 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_27 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_28 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_29 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_2A =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_2B =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_2C =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_2D =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_2E =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_2F =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_30 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_31 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
```

```
      INIT_32 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_33 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_34 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_35 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_36 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_37 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_38 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_39 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_3A =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_3B =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_3C =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_3D =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_3E =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INIT_3F =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INITP_00 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INITP_01 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INITP_02 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INITP_03 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INITP_04 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INITP_05 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INITP_06 =>
X"0000000000000000000000000000000000000000000000000000000000000000",
      INITP_07 =>
X"0000000000000000000000000000000000000000000000000000000000000000")
   port map (
      CASCADEOUTA => CASCADEOUTA, -- 1-bit cascade output
      CASCADEOUTB => CASCADEOUTB, -- 1-bit cascade output
      DOA => DOA,       -- 32-bit A port Data Output
      DOB => DOB,       -- 32-bit B port Data Output
      DOPA => DOPA,     -- 4-bit  A port Parity Output
      DOPB => DOPB,     -- 4-bit  B port Parity Output
      ADDRA => ADDRA,   -- 15-bit A port Address Input
      ADDRB => ADDRB,   -- 15-bit B port Address Input
      CASCADEINA => CASCADEINA, -- 1-bit cascade A input
      CASCADEINB => CASCADEINB, -- 1-bit cascade B input
      CLKA => CLKA,     -- Port A Clock
      CLKB => CLKB,     -- Port B Clock
      DIA => DIA,       -- 32-bit A port Data Input
      DIB => DIB,       -- 32-bit B port Data Input
```

```
      DIPA => DIPA,     -- 4-bit  A port parity Input
      DIPB => DIPB,     -- 4-bit  B port parity Input
      ENA => ENA,       -- 1-bit  A port Enable Input
      ENB => ENB,       -- 1-bit  B port Enable Input
      REGCEA => REGCEA, -- 1-bit A port register enable input
      REGCEB => REGCEB, -- 1-bit B port register enable input
      SSRA => SSRA,     -- 1-bit  A port Synchronous Set/Reset Input
      SSRB => SSRB,     -- 1-bit  B port Synchronous Set/Reset Input
      WEA => WEA,       -- 4-bit  A port Write Enable Input
      WEB => WEB        -- 4-bit  B port Write Enable Input
   );

   -- End of RAMB16_inst instantiation
```

RAMB16 Verilog Template

```
//     RAMB16    : To incorporate this function into the design,
//    Verilog    : the following instance declaration needs to be placed
//   instance    : in the body of the design code. The instance name
//  declaration  : (RAMB_inst) and/or the port declarations within the
//     code      : parenthesis can be changed to properly reference and
//               : connect this function to the design. All inputs
//               : and outputs must be connected.

//  <-----Cut code below this line---->

   // RAMB16: Virtex-4 16k+2k Parity Paramatizable Block RAM
   // Virtex-4 FPGA User Guide

   RAMB16 #(
     .DOA_REG(0),  // Optional output registers on A port (0 or 1)
     .DOB_REG(0),  // Optional output registers on B port (0 or 1)
     .INIT_A(36'h000000000),  // Initial values on A output port
     .INIT_B(36'h000000000),  // Initial values on B output port
     .INVERT_CLK_DOA_REG("FALSE"),// Invert clock on A port output
registers
       ("TRUE" or "FALSE")
     .INVERT_CLK_DOB_REG("FALSE"),// Invert clock on A port output
registers
       ("TRUE" or "FALSE")
     .RAM_EXTENSION_A("NONE"),  // "UPPER", "LOWER" or "NONE" when
cascaded
     .RAM_EXTENSION_B("NONE"),  // "UPPER", "LOWER" or "NONE" when
cascaded
     .READ_WIDTH_A(0),  // Valid values are 1, 2, 4, 9, 18, or 36
     .READ_WIDTH_B(0),  // Valid values are 1, 2, 4, 9, 18, or 36
     .SRVAL_A(36'h000000000), // Set/Reset value for A port output
     .SRVAL_B(36'h000000000),  // Set/Reset value for B port output
     .WRITE_MODE_A("WRITE_FIRST"),  // "WRITE_FIRST", "READ_FIRST", or
"NO_CHANGE"
     .WRITE_MODE_B("WRITE_FIRST"),  // "WRITE_FIRST", "READ_FIRST", or
"NO_CHANGE"
     .WRITE_WIDTH_A(0),  // Valid values are 1, 2, 4, 9, 18, or 36
     .WRITE_WIDTH_B(0),  // Valid values are 1, 2, 4, 9, 18, or 36

     // The following INIT_xx declarations specify the initial contents
of the RAM
```

```
.INIT_00(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_01(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_02(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_03(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_04(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_05(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_06(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_08(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_09(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_0A(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_0B(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_0C(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_0D(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_0E(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_0F(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_10(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_11(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_12(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),
```

```
.INIT_13(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_14(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_15(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_16(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_17(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_18(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_19(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_1A(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_1B(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_1C(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_1D(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_1E(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_1F(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_20(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_21(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_22(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_23(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_24(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_25(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),
```

```
.INIT_26(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_27(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_28(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_29(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_2A(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_2B(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_2C(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_2D(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_2E(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_2F(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_30(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_31(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_32(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_33(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_34(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_35(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_36(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_37(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_38(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),
```

```
.INIT_39(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_3A(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_3B(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_3C(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_3D(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_3E(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

.INIT_3F(256'h0000000000000000000000000000000000000000000000000000000000000000
00000000),

      // The next set of INITP_xx are for the parity bits

.INITP_00(256'h0000000000000000000000000000000000000000000000000000000000000000
000000000),

.INITP_01(256'h0000000000000000000000000000000000000000000000000000000000000000
000000000),

.INITP_02(256'h0000000000000000000000000000000000000000000000000000000000000000
000000000),

.INITP_03(256'h0000000000000000000000000000000000000000000000000000000000000000
000000000),

.INITP_04(256'h0000000000000000000000000000000000000000000000000000000000000000
000000000),

.INITP_05(256'h0000000000000000000000000000000000000000000000000000000000000000
000000000),

.INITP_06(256'h0000000000000000000000000000000000000000000000000000000000000000
000000000),

.INITP_07(256'h0000000000000000000000000000000000000000000000000000000000000000
000000000)
   ) RAMB16_inst (
      .CASCADEOUTA(CASCADEOUTA),  // 1-bit cascade output
      .CASCADEOUTB(CASCADEOUTB),  // 1-bit cascade output
      .DOA(DOA),      // 32-bit A port data output
      .DOB(DOB),      // 32-bit B port data output
      .DOPA(DOPA),    // 4-bit A port parity data output
      .DOPB(DOPB),    // 4-bit B port parity data output
      .ADDRA(ADDRA),  // 15-bit A port address input
      .ADDRB(ADDRB),  // 15-bit B port address input
      .CASCADEINA(CASCADEINA), // 1-bit cascade A input
      .CASCADEINB(CASCADEINB), // 1-bit cascade B input
      .CLKA(CLKA),     // 1-bit A port clock input
```

```
    .CLKB(CLKB),      // 1-bit B port clock input
    .DIA(DIA),        // 32-bit A port data input
    .DIB(DIB),        // 32-bit B port data input
    .DIPA(DIPA),      // 4-bit A port parity data input
    .DIPB(DIPB),      // 4-bit B port parity data input
    .ENA(ENA),        // 1-bit A port enable input
    .ENB(ENB),        // 1-bit B port enable input
    .REGCEA(REGCEA),  // 1-bit A port register enable input
    .REGCEB(REGCEB),  // 1-bit B port register enable input
    .SSRA(SSRA),      // 1-bit A port set/reset input
    .SSRB(SSRB),      // 1-bit B port set/reset input
    .WEA(WEA),        // 4-bit A port write enable input
    .WEB(WEB)         // 4-bit B port write enable input
);

// End of RAMB16_inst instantiation
```

Additional RAMB16 Primitive Design Considerations

The RAMB16 primitive is part of the Virtex-4 FPGA block RAM solution.

Data Parity Buses - DIP[A/B] and DOP[A/B]

The data parity buses are additional pins used for data parity with incoming data into the block RAM. The block RAM does not generate the parity bits for incoming data. These are supplied by the user. If not supplying parity bits, the pins can be used for incoming data.

Optional Output Registers

Optional output registers can be used at either or both A/B output ports of RAMB16. The choice is made using the DO[A/B]_REG attribute. There is also an option to invert the clocks for either or both of the A/B output registers using the INVERT_CLK_DO[A/B]_REG attribute. The two independent clock enable pins are REGCE[A/B]. When using the optional output registers at port [A|B], the synchronous set/reset (SSR) pin of ports [A|B] can not be used. Figure 4-5 shows a optional output register.

Independent Read and Write Port Width

To specify the port widths, designers must use the READ_WIDTH_[A/B] and WRITE_WIDTH_[A/B] attributes. The following rules should be considered:

- Designing a single port block RAM requires the port pair widths of one write and one read to be set (e.g., READ_WIDTH_A and WRITE_WIDTH_A).
- Designing a dual-port block RAM requires all port widths to be set.
- When using these attributes, if both write ports or both read ports are set to 0, the ISE® tools will not implement the design.

RAMB16 Port Mapping Design Rules

The Virtex-4 FPGA block RAM can be configurable to various port widths and sizes. Depending on the configuration, some data pins and address pins are not used. Table 4-2, page 125 shows the pins used in various configurations. In addition to the information in Table 4-2, the following rules are useful to determine port connections:

1. If the DI[A | B] pins are less than 32 bits wide, concatenate (32 – DI_BIT_WIDTH) logic zeros to the front of DI[A | B].
2. If the DIP[A | B] pins are less than 4 bits wide, concatenate (4 – DIP_BIT_WIDTH) logic zeros to the front of DIP[A | B]. DIP[A | B] is unconnected when not in use.
3. DO[A | B] pins must be 32 bits wide. However, valid data are only found on pins 0 to DO_BIT_WIDTH.
4. DOP[A | B] pins must be 4 bits wide. However, valid data are only found on pins 0 to DO_BIT_WIDTH. DOP[A | B] is unconnected when not in use.
5. ADDR[A | B] pins must be 15 bits wide. However, valid addresses for non-cascadable block RAM are only found on pins 13 to (14 – address width). The remaining pins, including pin 14, should be tied High.

Cascadable Block RAM

To use the cascadable block RAM feature:

1. Two RAMB16 primitives must be instantiated.
2. Set the RAM_EXTENSION_A and RAM_EXTENSION_B attribute for one RAMB16 to UPPER, and another to LOWER.
3. Connect the upper RAMB16's CASCADEINA and CASCADEINB ports to the CASCADEOUTA and CASCADEOUTB ports of the lower RAMB16. The CASCADEOUT ports for the upper RAMB16 do not require a connection. Connect the CASCADEIN ports for the lower RAMB16 to either logic High or Low.
4. The data output ports of the lower RAMB16 are not used. These pins are unconnected.
5. If placing location constraints on the two RAMB16s, they must be adjacent. If no location constraint is specified, the ISE software will automatically manage the RAMB16 locations.
6. The address pins ADDR[A | B] must be 15 bits wide. Both read and write ports must be one bit wide.

Figure 4-6 shows the cascadable block RAM.

Byte-Write Enable

The following rules should be considered when the following when using the byte-write enable feature:

- In x36 mode, WE[3:0] is connected to the four user WE inputs.
- In x18 mode, WE[0] and WE[2] are connected and driven by the user WE[0], while WE[1], and WE[3] are driven by the user WE[1].
- In x9, x4, x2, x1, WE[3:0] are all connected to a single user WE.

Figure 4-8 shows a byte-write enabled block RAM.

Additional Block RAM Primitives

In addition to RAMB16, some added block RAM primitives are available for Virtex-4 FPGA designers allowing the implementation of various block RAM sizes with preset configurations.

The input and output data buses are represented by two buses for 9-bit width (8+1), 18-bit width (16+2), and 36-bit width (32+4) configurations. The ninth bit associated with each byte can store parity or error correction bits. No specific function is performed on this bit.

The separate bus for parity bits facilitates some designs. However, other designs safely use a 9-bit, 18-bit, or 36-bit bus by merging the regular data bus with the parity bus. Read/write and storage operations are identical for all bits, including the parity bits.

Some block RAM attributes can only be configured using the RAMB16 primitive (e.g., pipeline register, cascade, etc.). See the "Block RAM Attributes" section.

Figure 4-10 shows the generic dual-port block RAM primitive. DIA, DIPA, ADDRA, DOA, DOPA, and the corresponding signals on port B are buses.

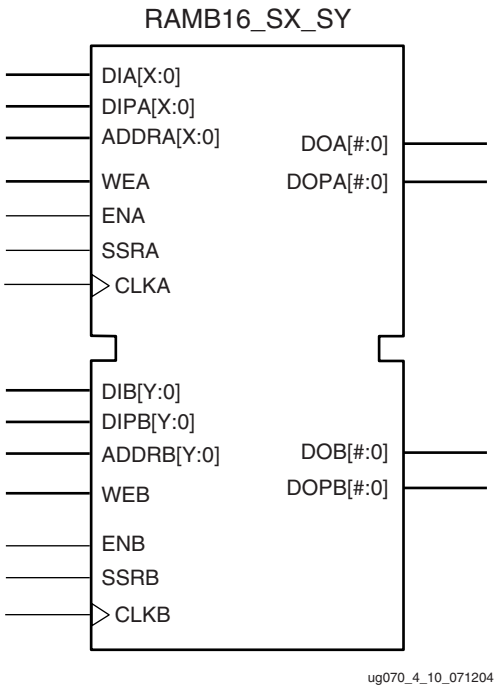


Figure 4-10: **Dual-Port Block RAM Primitive**

Table 4-6 lists the available dual-port primitives for synthesis and simulation.

Table 4-6: **Dual-Port Block RAM Primitives**

Primitive	Port A Width	Port B Width
RAMB16_S1_S1	1	1
RAMB16_S1_S2		2
RAMB16_S1_S4		4
RAMB16_S1_S9		(8+1)
RAMB16_S1_S18		(16+2)
RAMB16_S1_S36		(32+4)
RAMB16_S2_S2	2	2
RAMB16_S2_S4		4
RAMB16_S2_S9		(8+1)
RAMB16_S2_S18		(16+2)
RAMB16_S2_S36		(32+4)
RAMB16_S4_S4	4	4
RAMB16_S4_S9		(8+1)
RAMB16_S4_S18		(16+2)
RAMB16_S4_S36		(32+4)
RAMB16_S9_S9	(8+1)	(8+1)
RAMB16_S9_S18		(16+2)
RAMB16_S9_S36		(32+4)
RAMB16_S18_S18	(16+2)	(16+2)
RAMB16_S18_S36		(32+4)
RAMB16_S36_S36	(32+4)	(32+4)

Figure 4-11 shows the generic single-port block RAM primitive. DI, DIP, ADDR, DO, and DOP are buses.

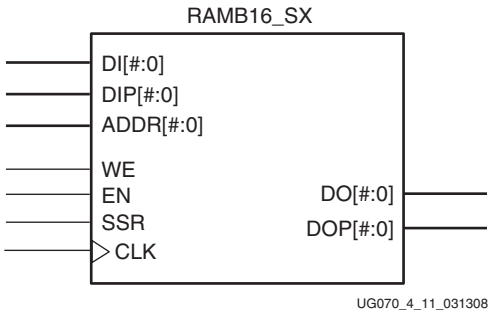


Figure 4-11: **Single-Port Block RAM Primitive**

Table 4-7 lists all of the available single-port primitives for synthesis and simulation.

Table 4-7: **Single-Port Block RAM Primitives**

Primitive	Port Width
RAMB16_S1	1
RAMB16_S2	2
RAMB16_S4	4
RAMB16_S9	(8+1)
RAMB16_S18	(16+2)
RAMB16_S36	(32+4)

Instantiation of Additional Block RAM Primitives

The RAM_Ax templates (with x = 1, 2, 4, 9, 18, or 36) are single-port modules and instantiate the corresponding RAMB16_Sx module.

RAM_Ax_By templates (with x = 1, 2, 4, 9, 18, or 36 and y = = 1, 2, 4, 9, 18, or 36) are dual-port modules and instantiate the corresponding RAMB16_Sx_Sy module.

Block RAM Applications

Creating Larger RAM Structures

Block RAM columns have special routing to create wider/deeper blocks with minimal routing delays. Wider or deeper RAM structures are achieved with a smaller timing penalty than is encountered when using normal routing resources.

The CORE Generator software offers the designer an easy way to generate wider and deeper memory structures using multiple block RAM instances. This program outputs VHDL or Verilog instantiation templates and simulation models, along with an EDIF file for inclusion in a design.

Block RAM Timing Model

This section describes the timing parameters associated with the block RAM in Virtex-4 devices (illustrated in Figure 4-12). The switching characteristics section in the *Virtex-4 Data Sheet* and the Timing Analyzer (TRCE) report from Xilinx software are also available for reference.

Block RAM Timing Parameters

Table 4-8 shows the Virtex-4 FPGA block RAM timing parameters.

Table 4-8: **Block RAM Timing Parameters**

Parameter	Function	Control Signal	Description
Setup and Hold Relative to Clock (CLK)			
T_{RxCK_x} = Setup time (before clock edge) and T_{RCKx_x} = Hold time (after clock edge)			
T_{RCCK_ADDR}	Address inputs	ADDR	Time before the clock that address signals must be stable at the ADDR inputs of the block RAM.(1)
T_{RCKC_ADDR}			Time after the clock that address signals must be stable at the ADDR inputs of the block RAM.(1)
T_{RDCK_DI}	Data inputs	DI	Time before the clock that data must be stable at the DI inputs of the block RAM.
T_{RCKD_DI}			Time after the clock that data must be stable at the DI inputs of the block RAM.
T_{RCCK_EN}	Enable	EN	Time before the clock that the enable signal must be stable at the EN input of the block RAM.
T_{RCKC_EN}			Time after the clock that the enable signal must be stable at the EN input of the block RAM.
T_{RCCK_SSR}	Synchronous Set/Reset	SSR	Time before the clock that the synchronous set/reset signal must be stable at the SSR input of the block RAM.
T_{RCKC_SSR}			Time after the clock that the synchronous set/reset signal must be stable at the SSR input of the block RAM.
T_{RCCK_WEN}	Write Enable	WEN	Time before the clock that the write enable signal must be stable at the WEN input of the block RAM.
T_{RCKC_WEN}			Time after the clock that the write enable signal must be stable at the WEN input of the block RAM.
T_{RCCK_REGCE}	Optional Output Register Enable	REGCE	Time before the clock that the register enable signal must be stable at the REGCE input of the block RAM.
T_{RCKC_REGCE}			Time after the clock that the register enable signal must be stable at the REGCE input of the block RAM.
Sequential Delays			
T_{RCKO_DO} (Max)	Clock to Output	CLK to DO	Time after the clock that the output data is stable at the DO outputs of the block RAM (without output register).
T_{RCKO_DO} (Min)	Clock to Output	CLK to DO	Time after the clock that the output data is stable at the DO outputs of the block RAM (with output register).

Notes:

1. While EN is active, ADDR inputs must be stable during the entire setup/hold time window, even if WEN is inactive. Violating this requirement can result in block RAM data corruption. If ADDR timing could violate the specified requirements, EN must be inactive (disabled).

Block RAM Timing Characteristics

The timing diagram in Figure 4-12 describes a single-port block RAM in write-first mode without the optional output register. The timing for read-first and no-change modes are similar. For timing using the optional output register, an additional clock latency appears at the DO pin.

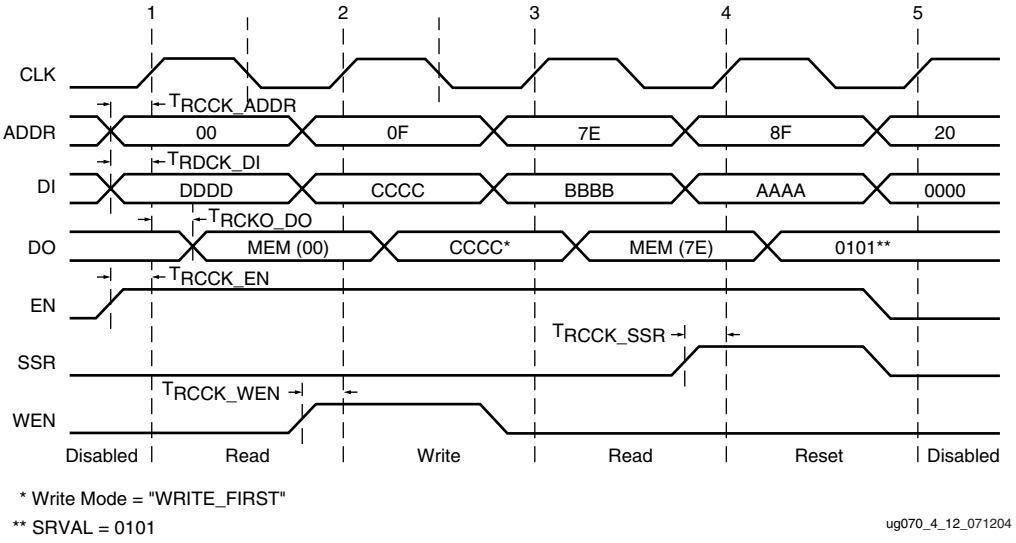


Figure 4-12: **Block RAM Timing Diagram**

At time 0, the block RAM is disabled; EN (enable) is Low.

Clock Event 1

Read Operation

During a read operation, the contents of the memory at the address on the ADDR inputs are unchanged.

- T_{RCCK_ADDR} before clock event 1, address 00 becomes valid at the ADDR inputs of the block RAM.
- At time T_{RCCK_EN} before clock event 1, enable is asserted High at the EN input of the block RAM, enabling the memory for the READ operation that follows.
- At time T_{RCKO_DO} after clock event 1, the contents of the memory at address 00 become stable at the DO pins of the block RAM.

Clock Event 2

Write Operation

During a write operation, the content of the memory at the location specified by the address on the ADDR inputs is replaced by the value on the DI pins and is immediately reflected on the output latches (in WRITE-FIRST mode); EN (enable) is High.

- At time T_{RCCK_ADDR} before clock event 2, address 0F becomes valid at the ADDR inputs of the block RAM.
- At time T_{RDCK_DI} before clock event 2, data CCCC becomes valid at the DI inputs of the block RAM.

- At time T_{RCCK_WEN} before clock event 2, write enable becomes valid at the WEN following the block RAM.
- At time T_{RCKO_DO} after clock event 2, data CCCC becomes valid at the DO outputs of the block RAM.

Clock Event 4

SSR (Synchronous Set/Reset) Operation

During an SSR operation, initialization parameter value SRVAL is loaded into the output latches of the block RAM. The SSR operation does NOT change the contents of the memory and is independent of the ADDR and DI inputs.

- At time T_{RCCK_SSR} before clock event 4, the synchronous set/reset signal becomes valid (High) at the SSR input of the block RAM.
- At time T_{RCKO_DO} after clock event 4, the SRVAL 0101 becomes valid at the DO outputs of the block RAM.

Clock Event 5

Disable Operation

Deasserting the enable signal EN disables any write, read, or SSR operation. The disable operation does NOT change the contents of the memory or the values of the output latches.

- At time T_{RCCK_EN} before clock event 5, the enable signal becomes valid (Low) at the EN input of the block RAM.
- After clock event 5, the data on the DO outputs of the block RAM is unchanged.

Block RAM Timing Model

Figure 4-13 illustrates the delay paths associated with the implementation of block RAM. This example takes the simplest paths on and off chip (these paths can vary greatly depending on the design). This timing model demonstrates how and where the block RAM timing parameters are used.

- **NET** = Varying interconnect delays
- $\mathbf{T_{IOPI}}$ = Pad to I-output of IOB delay
- $\mathbf{T_{IOOP}}$ = O-input of IOB to pad delay
- $\mathbf{T_{BCCKO\_O}}$ = BUFGCTRL delay

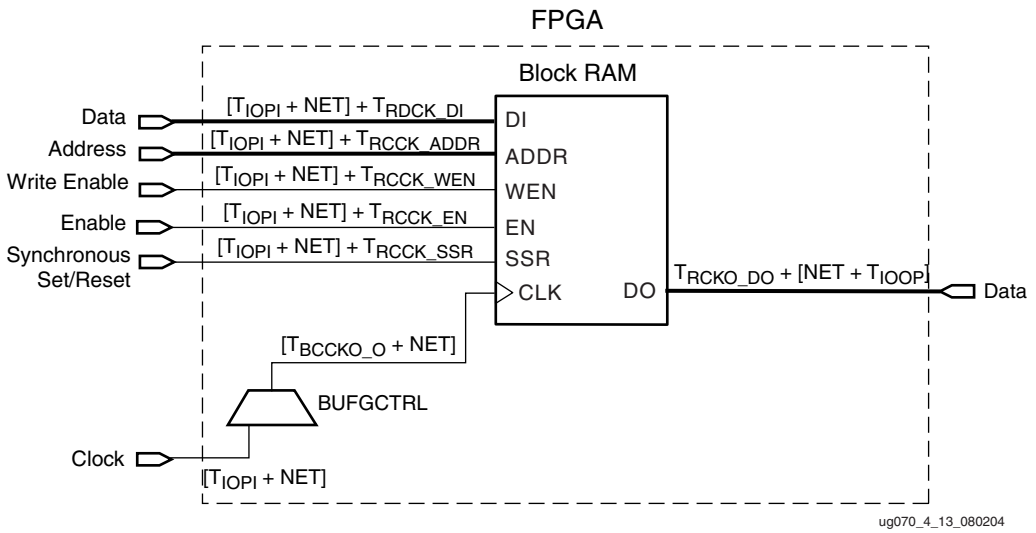


Figure 4-13: **Block RAM Timing Model**

Built-in FIFO Support

A large percentage of FPGA designs use block RAMs to implement FIFOs. In the Virtex-4 architecture, dedicated logic in the block RAM enables users to easily implement synchronous or asynchronous FIFOs. This eliminates the need for additional CLB logic for counter, comparator, or status flag generation, and uses just one block RAM resource per FIFO. Both standard and first-word fall-through (FWFT) modes are supported.

The supported configurations are 4K x 4, 2K x 9, 1K x 18, and 512 x 36.

The block RAM can be configured as first-in/first-out (FIFO) memory with common or independent read and write clocks. Port A of the block RAM is used as a FIFO read port, and Port B is a FIFO write port. Data is read from the FIFO on the rising edge of read clock and written to the FIFO on the rising edge of write clock. Independent read and write port width selection is not supported in FIFO mode without the aid of external CLB logic.

The FIFO offers a very simple user interface. The design relies on free-running write and read clocks, of identical or different frequencies up to the specified maximum frequency limit. The design avoids any ambiguity, glitch, or metastable problems, even when the two frequencies are completely unrelated.

The write operation is synchronous, writing the data word available at DI into the FIFO whenever WREN is active a setup time before the rising WRCLK edge.

The read operation is also synchronous, presenting the next data word at DO whenever the RDEN is active one setup time before the rising RDCLK edge.

Data flow control is automatic; the user need not be concerned about the block RAM addressing sequence, although WRCOUNT and RDCOUNT are also brought out, if needed for unusual applications.

The user must, however, observe the FULL and EMPTY flags, and stop writing when FULL is High, and stop reading when EMPTY is High. If these rules are violated, an active WREN while FULL is High will activate the WRERR flag, and an active RDEN while EMPTY is High will activate the RDERR flag. In either violation, the FIFO content will, however, be preserved, and the address counters will stay valid.

Programmable ALMOSTFULL and ALMOSTEMPTY flags are brought out to give the user an early warning when the FIFO is approaching its limits. Both these flag values can be set by configuration to (almost) anywhere in the FIFO address range.

Two operating modes affect the reading of the first word after the FIFO was empty:

- In Standard mode, the first word written into an empty FIFO will appear at DO after the user has activated RDEN. The user must "pull" the data out of the FIFO.
- In FWFT mode, the first word written into an empty FIFO will automatically appear at DO without the user activating RDEN. The FIFO "pushes" the data onto DO. The next RDEN will then "pull" the subsequent data word onto DO.

EMPTY Latency

The rising edge of EMPTY is fast, and inherently synchronous with RDCLK. The empty condition can only be terminated by WRCLK, asynchronous to RDCLK. The falling edge of EMPTY must, therefore, artificially be moved onto the RDCLK time domain. Since the two clocks have an unknown phase relationship, it takes several cascaded flip-flops to guarantee that such a move does not cause glitches or metastable problems. The falling edge of EMPTY is thus delayed by several RDCLK periods after the first write into the previously empty FIFO. This delay guarantees proper operation under all circumstances, and causes an insignificant loss of performance after the FIFO had gone empty.

Table 4-9 shows the FIFO capacity in the two modes.

Table 4-9: **FIFO Capacity**

Standard Mode	FWFT Mode
4k+1 entries by 4 bits	4k+2 entries by 4 bits
2k+1 entries by 9 bits	2k+2 entries by 9 bits
1k+1 entries by 18 bits	1k+2 entries by 18 bits
512+1 entries by 36 bits	512+2 entries by 36 bits

Top-Level View of FIFO Architecture

Figure 4-14 shows a top-level view of the Virtex-4 FIFO architecture. The read pointer, write pointer, and status flag logic are dedicated for FIFO use only.

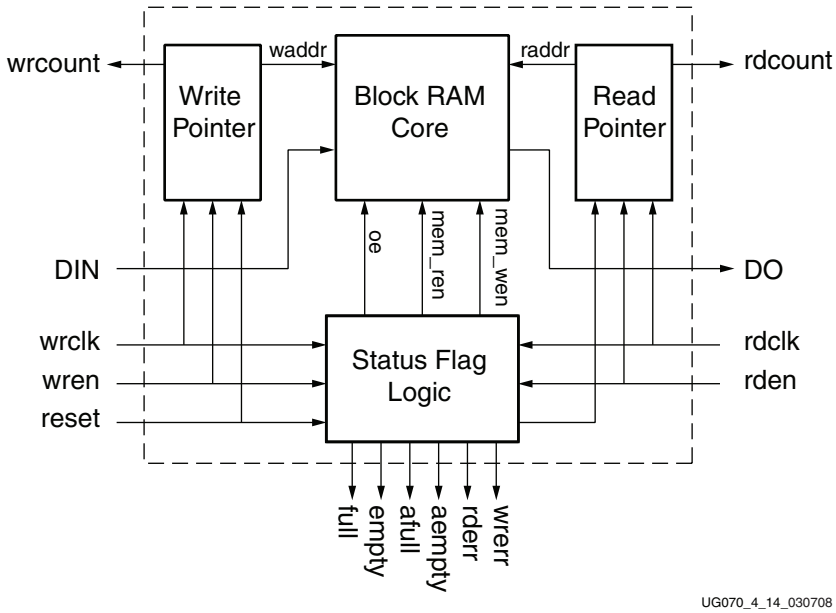


Figure 4-14: **Top-Level View of FIFO in Block RAM**

FIFO Primitive

Figure 4-15 shows the FIFO16 primitive.

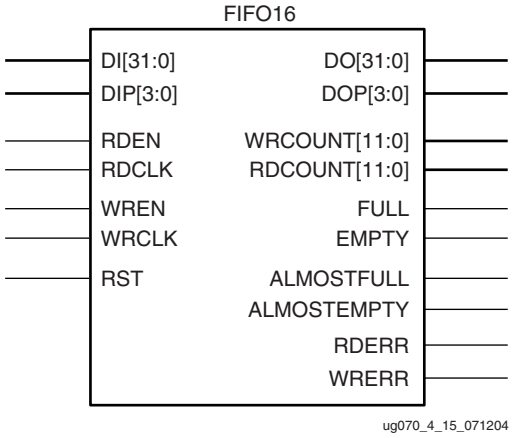


Figure 4-15: **FIFO16 Primitive**

FIFO Port Descriptions

Table 4-10 lists the FIFO I/O port names and descriptions.

Table 4-10: **FIFO I/O Port Names and Descriptions**

Port Name	Direction	Description
DI	Input	Data input.
DIP	Input	Parity-bit input.
WREN	Input	Write enable. When WREN = 1, data will be written to memory. When WREN = 0, write is disabled.
WRCLK	Input	Clock for write domain operation.
RDEN	Input	Read enable. When RDEN = 1, data will be read to output register. When RDEN = 0, read is disabled.
RDCLK	Input	Clock for read domain operation.
RESET	Input	Asynchronous reset of all FIFO functions, flags, and pointers.
DO	Output	Data output, synchronous to RDCLK.
DOP	Output	Parity-bit output, synchronous to RDCLK.
FULL	Output	All entries in FIFO memory are filled. No additional write enable is performed. Synchronous to WRCLK.
ALMOSTFULL	Output	Almost all entries in FIFO memory have been filled. Synchronous to WRCLK. The offset for this flag is user configurable.
EMPTY	Output	FIFO is empty. No additional read can be performed. Synchronous to RDCLK.
ALMOSTEMPTY	Output	Almost all valid entries in FIFO have been read. Synchronous with RDCLK. The offset for this flag is user configurable.
RDCOUNT	Output	The FIFO data read pointer. It is synchronous with RDCLK. The value will wrap around if the maximum read pointer value has been reached.
WRCOUNT	Output	The FIFO data write pointer. It is synchronous with WRCLK. The value will wrap around if the maximum write pointer value has been reached.
WRERR	Output	When the FIFO is full, any additional write operation generates an error flag. Synchronous with WRCLK.
RDERR	Output	When the FIFO is empty, any additional read operation generates an error flag. Synchronous with RDCLK.

FIFO Operations

Reset

Reset is an asynchronous signal to reset all read and write address counters, and must be asserted to initialize flags after power up. Reset does not clear the memory, nor does it clear the output register. When reset is asserted High, EMPTY and ALMOST_EMPTY will be set to 1, FULL and ALMOST_FULL will be reset to 0. The reset signal must be High for at least three read clock and write clock cycles to ensure all internal states are reset to the correct values. During RESET, RDEN and WREN must be held Low.

Operating Mode

There are two operating modes in FIFO functions. They differ only in output behavior after the first word is written to a previously empty FIFO.

Standard Mode

After the first word is written into an empty FIFO, the Empty flag deasserts synchronously with RDCLK. After Empty is deasserted Low and RDEN is asserted, the first word will appear at DO on the rising edge of RDCLK.

First Word Fall Through (FWFT) Mode

After the first word is written into an empty FIFO, it automatically appears at DO without asserting RDEN. Subsequent Read operations require Empty to be Low and RDEN to be High. Figure 4-16 illustrates the difference between standard mode and FWFT mode.

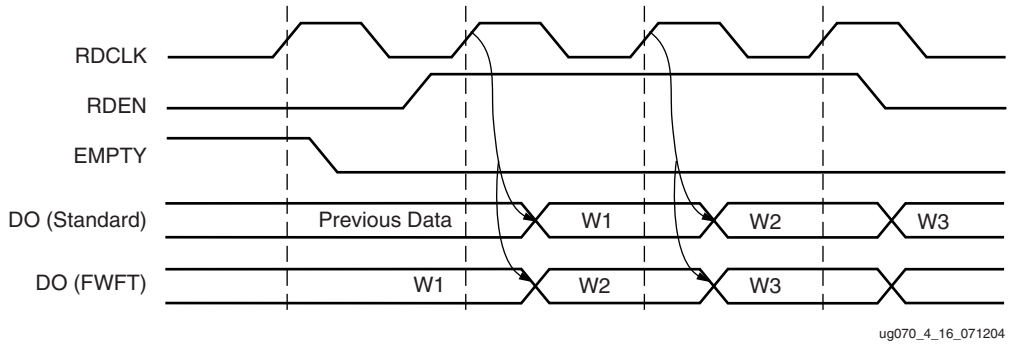


Figure 4-16: **Read Cycle Timing (Standard and FWFT Modes)**

Status Flags

Empty Flag

The Empty flag is synchronous with RDCLK, and is asserted when the last entry in the FIFO is read. When there are no more valid entries in the FIFO queue, the read pointer will be frozen. The Empty flag is deasserted at three (in standard mode) or four (in FWFT mode) read clocks after new data is written into the FIFO.

ALMOSTEMPTY Flag

The ALMOSTEMPTY flag is set when the FIFO contains the number of entries specified by the ALMOST_EMPTY_OFFSET value (or fewer), warning the user to stop reading. The ALMOSTEMPTY flag deasserts three clock cycles after the number of entries in the FIFO becomes greater than the ALMOST_EMPTY_OFFSET value. It is synchronous to RDCLK.

Read Error Flag

Once the Empty flag has been asserted, any further read attempts will not increment the read address pointer but will trigger the Read Error flag. The Read Error flag is deasserted when Read Enable or Empty is deasserted Low. The Read Error flag is synchronous to RDCLK.

Full Flag

The Full flag is synchronous with WRCLK, and is asserted one WRCLK after there are no more available entries in the FIFO queue. Because of this latency, it is recommended to use the ALMOST_FULL signal to stop further writing. When the FIFO is full, the write pointer will be frozen. The Full flag is deasserted three write clock cycles after any read operation.

Write Error Flag

Once the Full flag has been asserted, any further write attempts will not increment the write address pointer but will trigger the Write Error flag. The Write Error flag is deasserted when Write Enable or Full is deasserted Low. This signal is synchronous to WRCLK.

ALMOSTFULL Flag

The ALMOSTFULL flag is set when the FIFO has the number of available empty spaces specified by the ALMOST_FULL_OFFSET value or fewer. The ALMOSTFULL flag warns the user to stop writing. It deasserts when the number of empty spaces in the FIFO is greater than the ALMOST_FULL_OFFSET value, and is synchronous to WRCLK.

Table 4-11 shows the number of clock cycles to assert or deassert each flag.

Table 4-11: **Clock Cycle Latency for Flag Assertion and Deassertion**

Clock Cycle Latency	Assertion		Deassertion	
	Standard	FWFT	Standard	FWFT
EMPTY	0	0	3	4
FULL	1	1	3	3
ALMOST EMPTY(1)	1	1	3	3
ALMOST FULL(1)	1	1	3	3
READ ERROR	0	0	0	0
WRITE ERROR	0	0	0	0

Notes:

1. Depending on the time between read and write clock edges, the ALMOSTEMPTY and ALMOSTFULL flags can deassert one cycle later.

FIFO Attributes

Table 4-12 lists the FIFO16 attributes. The size of the asynchronous FIFO can be configured by setting the DATA_WIDTH attribute. The "FIFO VHDL and Verilog Templates" section has examples for setting the attributes.

Table 4-12: **FIFO16 Attributes**

Attribute Name	Type	Values	Default	Notes
ALMOST_FULL_OFFSET	12-bit HEX	See Table 4-13		Setting determines ALMOST_FULL condition. Must be set using hexadecimal notation.
ALMOST_EMPTY_OFFSET	12-bit HEX	See Table 4-13		Setting determine ALMOST_EMPTY condition. Must be set using hexadecimal notation.
FIRST_WORD_FALL_THROUGH	Boolean	FALSE, TRUE	FALSE	If TRUE, during a write of the 1st word the word appears at the FIFO output without RDEN asserted.
DATA_WIDTH	Integer	4, 9, 18, 36	36	
LOC	String	Valid FIFO16 location		Sets the location of the FIFO16.

Notes:

1. If FIFO16 is constrained to FIFO16_X#Y#, then RAMB16 can not be constrained to RAMB16_X#Y# since the same location would be used.

FIFO ALMOSTEMPTY / ALMOSTFULL Flag Offset Range

The offset ranges for ALMOSTEMPTY and ALMOSTFULL are listed in Table 4-13.

Table 4-13: **FIFO ALMOSTFULL / EMPTY Flag Offset Range**

Configuration	ALMOST_EMPTY_OFFSET		ALMOST_FULL_OFFSET
	Standard	FWFT	
4k x 4	5 to 4092	6 to 4093	4 to 4091
2k x 9	5 to 2044	6 to 2045	4 to 2043
1k x 18	5 to 1020	6 to 1021	4 to 1019
512 x 36	5 to 508	6 to 509	4 to 507

Notes:

1. ALMOST_EMPTY_OFFSET and ALMOST_FULL_OFFSET for any design must be less than the FIFO depth.

The ALMOSTFULL and ALMOSTEMPTY offsets are usually set to a small value of less than 10 to provide a warning that the FIFO is about to reach its limits. Since the full capacity of any FIFO is normally not critical, most applications use the ALMOST_FULL flag not only as a warning but also as a signal to stop writing.

Similarly, the ALMOST_EMPTY flag can be used to stop reading. However, this would make it impossible to read the very last entries remaining in the FIFO. The user can ignore the ALMOSTEMPTY signal and continue to read until EMPTY is asserted.

The ALMOSTFULL and ALMOSTEMPTY offsets can also be used in unstoppable block transfer applications to signal that a new block of data can be written or read.

When setting the offset ranges in the design tools, use hexadecimal notation.

FIFO VHDL and Verilog Templates

VHDL and Verilog templates are available in the Libraries Guide. Also see section "FIFO16 Error Condition and Work-Arounds," page 165.

FIFO VHDL Template

```
--    FIFO16     : To incorporate this function into the design, the
--     VHDL      : following instance declaration needs to be placed in
--   instance    : the architecture body of the design code. The instance
--  declaration  : name (FIFO16_inst) and/or the port declarations
--     code      : after the "=>" assignment can be changed to properly
--               : connect this function to the design. All inputs and
--               : outputs must be connected.

--    Library    : In addition to adding the instance declaration, a use
--  declaration  : statement for the UNISIM.v components library needs
--      for      : to be added before the entity declaration. This
--    Xilinx     : library contains the component declarations for all
--   primitives  : Xilinx primitives and points to the models that will
--               : be used for simulation.

--  Copy the following two statements and paste them before the
--  Entity declaration, unless they already exists.

Library UNISIM;
use UNISIM.vcomponents.all;

--  <--Cut code below this line and paste into the architecture body-->

   -- FIFO16: Virtex-4 Block RAM Asynchronous FIFO
   -- Virtex-4 FPGA User Guide

   FIFO16_inst : FIFO16
   generic map (
      ALMOST_FULL_OFFSET => X"000",  -- Sets almost full threshold
      ALMOST_EMPTY_OFFSET => X"000", -- Sets the almost empty threshold
      DATA_WIDTH => 36, -- Sets data width to 4, 9, 18, or 36
      FIRST_WORD_FALL_THROUGH => FALSE) -- Sets the FIFO FWFT to TRUE or FALSE
   port map (
      ALMOSTEMPTY => ALMOSTEMPTY, -- 1-bit almost empty output flag
      ALMOSTFULL => ALMOSTFULL,   -- 1-bit almost full output flag
      DO => DO,                   -- 32-bit data output
      DOP => DOP,                 -- 4-bit parity data output
      EMPTY => EMPTY,             -- 1-bit empty output flag
      FULL => FULL,               -- 1-bit full output flag
      RDCOUNT => RDCOUNT,         -- 12-bit read count output
      RDERR => RDERR,             -- 1-bit read error output
      WRCOUNT => WRCOUNT,         -- 12-bit write count output
      WRERR => WRERR,             -- 1-bit write error
      DI => DI,                   -- 32-bit data input
      DIP => DIP,                 -- 4-bit partity input
```

```
      RDCLK => RDCLK,               -- 1-bit read clock input
      RDEN => RDEN,                 -- 1-bit read enable input
      RST => RST,                   -- 1-bit reset input
      WRCLK => WRCLK,               -- 1-bit write clock input
      WREN => WREN                  -- 1-bit write enable input
   );

   -- End of FIFO16_inst instantiation
```

FIFO Verilog Template

```
//     FIFO16    : To incorporate this function into the design, the
//    Verilog    : following instance declaration needs to be placed in
//   instance    : the body of the design code. The instance name
//  declaration  : (FIFO16_1kx18_inst) and/or the port declarations
//     code      : within the parenthesis can be changed to properly
//               : reference and connect this function to the design.
//               : All inputs and outputs must be connected.

//  <-----Cut code below this line---->

// FIFO16: Virtex-4 Block RAM Asynchronous FIFO configured for 1k deep x
// 18 wide
// Virtex-4 FPGA User Guide

   FIFO16 #(
      .ALMOST_FULL_OFFSET(12'h000),     // Sets almost full threshold
      .ALMOST_EMPTY_OFFSET(12'h000),    // Sets the almost empty
threshold
      .DATA_WIDTH(36),                 // Sets data width to 4, 9, 18,
or 36
     .FIRST_WORD_FALL_THROUGH("FALSE") // Sets the FIFO FWFT to "TRUE"
or "FALSE"
   ) FIFO16_inst (
      .ALMOSTEMPTY(ALMOSTEMPTY), // 1-bit almost empty output flag
      .ALMOSTFULL(ALMOSTFULL),   // 1-bit almost full output flag
      .DO(DO),                   // 32-bit data output
      .DOP(DOP),                 // 4-bit parity data output
      .EMPTY(EMPTY),             // 1-bit empty output flag
      .FULL(FULL),               // 1-bit full output flag
      .RDCOUNT(RDCOUNT),         // 12-bit read count output
      .RDERR(RDERR),             // 1-bit read error output
      .WRCOUNT(WRCOUNT),         // 12-bit write count output
      .WRERR(WRERR),             // 1-bit write error
      .DI(DI),                   // 32-bit data input
      .DIP(DIP),                 // 4-bit partity input
      .RDCLK(RDCLK),             // 1-bit read clock input
      .RDEN(RDEN),               // 1-bit read enable input
      .RST(RST),                 // 1-bit reset input
      .WRCLK(WRCLK),             // 1-bit write clock input
      .WREN(WREN)                // 1-bit write enable input
   );

   // End of FIFO16_1kx18_inst instantiation
```

FIFO Timing Models and Parameters

Table 4-14 shows the FIFO parameters.

Table 4-14: **FIFO Timing Parameters**

Parameter	Function	Control Signal	Description
Setup and Hold Relative to Clock (CLK)			
T_{FXCK} = Setup time (before clock edge) T_{FCKX} = Hold time (after clock edge)		The following descriptions are for setup times only.	
T_{FDCK_DI}/ T_{FCKD_DI}(4)	Data inputs	DI	Time before WRCLK that data must be stable at the DI inputs of the FIFO.
T_{FCCK_RDEN}/ T_{FCKC_RDEN}(5)	Read enable	RDEN	Time before RDCLK that Read Enable must be stable at the RDEN inputs of the FIFO.
T_{FCCK_WREN}/ T_{FCKC_WREN}(5)	Write enable	WREN	Time before WRCLK that write enable must be stable at the WREN inputs of the FIFO.
Sequential Delays			
T_{FCKO_DO}(1)	Clock to data output	DO	Time after RDCLK that the output data is stable at the DO outputs of the FIFO.
T_{FCKO_AEMPTY}(2)	Clock to ALMOSTEMPTY output	AEMPTY	Time after RDCLK that the ALMOSTEMPTY signal is stable at the ALMOSTEMPTY outputs of the FIFO.
T_{FCKO_AFULL}(2)	Clock to ALMOSTFULL output	AFULL	Time after WRCLK that the ALMOSTFULL signal is stable at the ALMOSTFULL outputs of the FIFO.
T_{FCKO_EMPTY}(2)	Clock to EMPTY output	EMPTY	Time after RDCLK that the Empty signal is stable at the EMPTY outputs of the FIFO.
T_{FCKO_FULL}(2)	Clock to FULL output	FULL	Time after WRCLK that the FULL signal is stable at the FULL outputs of the FIFO.
T_{FCKO_RDERR}(2)	Clock to read error output	RDERR	Time after RDCLK that the Read Error signal is stable at the RDERR outputs of the FIFO.
T_{FCKO_WRERR}(2)	Clock to write error output	WRERR	Time after WRCLK that the Write Error signal is stable at the WRERR outputs of the FIFO.
$T_{FCKO\_RDCOUNT}$(3)	Clock to read pointer output	RDCOUNT	Time after RDCLK that the Read pointer signal is stable at the RDCOUNT outputs of the FIFO.
$T_{FCKO\_WRCOUNT}$(3)	Clock to write pointer output	WRCOUNT	Time after WRCLK that the Write pointer signal is stable at the WRCOUNT outputs of the FIFO.
Reset to Out			
T_{FCO_AEMPTY}	Reset to ALMOSTEMPTY output	AEMPTY	Time after reset that the ALMOSTEMPTY signal is stable at the ALMOSTEMPTY outputs of the FIFO.
T_{FCO_AFULL}	Reset to ALMOSTFULL output	AFULL	Time after reset that the ALMOSTFULL signal is stable at the ALMOSTFULL outputs of the FIFO.
T_{FCO_EMPTY}	Reset to EMPTY output	EMPTY	Time after reset that the Empty signal is stable at the EMPTY outputs of the FIFO.

Table 4-14: **FIFO Timing Parameters *(Continued)***

Parameter	Function	Control Signal	Description
T_{FCO_FULL}	Reset to FULL output	FULL	Time after reset that the FULL signal is stable at the FULL outputs of the FIFO.
T_{FCO_RDERR}	Reset to read error output	RDERR	Time after reset that the Read error signal is stable at the RDERR outputs of the FIFO.
T_{FCO_WRERR}	Reset to write error output	WRERR	Time after reset that the Write error signal is stable at the WRERR outputs of the FIFO.
$T_{FCO\_RDCOUNT}$	Reset to read pointer output	RDCOUNT	Time after reset that the Read pointer signal is stable at the RDCOUNT outputs of the FIFO.
$T_{FCO\_WRCOUNT}$	Reset to write pointer output	WRCOUNT	Time after reset that the Write pointer signal is stable at the WRCOUNT outputs of the FIFO.

Notes:

1. T_{FCKO_DO} includes parity output (T_{FCKO_DOP}).
2. In the *Virtex-4 Data Sheet*, T_{FCKO_AEMPTY}, T_{FCKO_AFULL}, T_{FCKO_EMPTY}, T_{FCKO_FULL}, T_{FCKO_RDERR}, T_{FCKO_WRERR} are combined into T_{FCKO_FLAGS}.
3. In the *Virtex-4 Data Sheet*, $T_{FCKO\_RDCOUNT}$ and $T_{FCKO\_WRCOUNT}$ are combined into $T_{FCKO\_POINTERS}$.
4. T_{FCDCK_DI} includes parity inputs (T_{FCDCK_DIP}).
5. In the *Virtex-4 Data Sheet*, WRITE and READ enables are combined into T_{FCCK_EN}.

FIFO Timing Characteristics

The various timing parameters in the FIFO are described in this section. There is also additional data on FIFO functionality. The timing diagrams describe the behavior in these five cases.

- "Case 1: Writing to an Empty FIFO"
- "Case 2: Writing to a Full or Almost Full FIFO"
- "Case 3: Reading From a Full FIFO"
- "Case 4: Reading From an Empty or Almost Empty FIFO"
- "Case 5: Resetting All Flags"

Case 1: Writing to an Empty FIFO

Prior to the operations performed in Figure 4-17, the FIFO is completely empty.

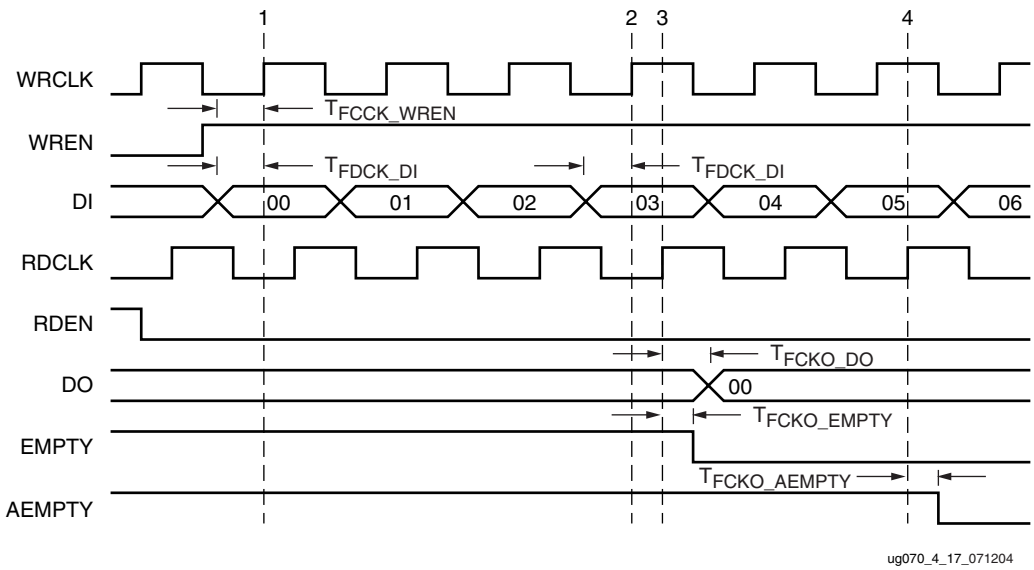


Figure 4-17: **Writing to an Empty FIFO in FWFT Mode**

Clock Event 1 and Clock Event 3: Write Operation and Deassertion of EMPTY Signal

During a write operation to an empty FIFO, the content of the FIFO at the first address is replaced by the data value on the DI pins. Three read-clock cycles later (four read-clock cycles for FWFT mode), the EMPTY pin is deasserted when the FIFO is no longer empty.

For the example in Figure 4-17, the timing diagram is drawn to reflect FWFT mode. Clock event 1 is with respect to the write-clock, while clock event 3 is with respect to the read-clock. Clock event 3 appears four read-clock cycles after clock event 1.

- At time T_{FDCK_DI}, before clock event 1 (WRCLK), data 00 becomes valid at the DI inputs of the FIFO.
- At time T_{FCCK_WREN}, before clock event 1 (WRCLK), write enable becomes valid at the WREN input of the FIFO.
- At time T_{FCKO_DO}, after clock event 3 (RDCLK), data 00 becomes valid at the DO output pins of the FIFO. In the case of standard mode, data 00 does not appear at the DO output pins of the FIFO.
- At time T_{FCKO_EMPTY}, after clock event 3 (RDCLK), EMPTY is deasserted. In the case of standard mode, EMPTY is deasserted one read-clock earlier than clock event 3.

If the rising WRCLK edge is close to the rising RDCLK edge, EMPTY could be deasserted one RDCLK period later.

Clock Event 2 and Clock Event 4: Write Operation and Deassertion of ALMOSTEMPTY Signal

Three read-clock cycles after the fourth data is written into the FIFO, the ALMOSTEMPTY pin is deasserted to signify that the FIFO is not in the ALMOSTEMPTY state.

For the example in Figure 4-17, the timing diagram is drawn to reflect FWFT mode. Clock event 2 is with respect to write-clock, while clock event 4 is with respect to read-clock. Clock event 4 appears three read-clock cycles after clock event 2.

- At time T_{FDCK_DI}, before clock event 2 (WRCLK), data 03 becomes valid at the DI inputs of the FIFO.
- Write enable remains asserted at the WREN input of the FIFO.
- At clock event 4, DO output pins of the FIFO remains at 00 since no read has been performed. In the case of standard mode, data 00 will never appear at the DO output pins of the FIFO.
- At time T_{FCKO_AEMPTY}, after clock event 4 (RDCLK), ALMOSTEMPTY is deasserted at the AEMPTY pin. In the case of standard mode, AEMPTY deasserts in the same way as in FWFT mode.

If the rising WRCLK edge is close to the rising RDCLK edge, AEMPTY could be deasserted one RDCLK period later.

Case 2: Writing to a Full or Almost Full FIFO

Prior to the operations performed in Figure 4-18, the FIFO is almost completely full. In this example, the timing diagram reflects of both standard and FWFT modes.

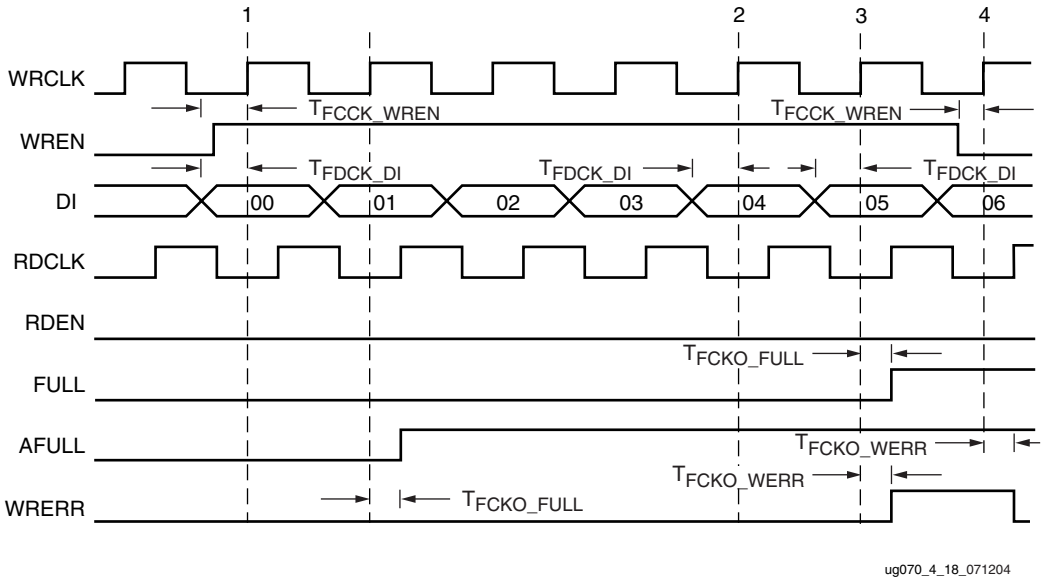


Figure 4-18: **Writing to a Full / Almost Full FIFO**

Clock Event 1: Write Operation and Assertion of ALMOSTFULL Signal

During a write operation to an almost full FIFO, the ALMOSTFULL signal is asserted.

- At time T_{FDCK_DI}, before clock event 1 (WRCLK), data 00 becomes valid at the DI inputs of the FIFO.
- At time T_{FCCK_WREN}, before clock event 1 (WRCLK), write enable becomes valid at the WREN input of the FIFO.
- At time T_{FCKO_AFULL}, one clock cycle after clock event 1 (WRCLK), ALMOSTFULL is asserted at the AFULL output pin of the FIFO.

Clock Event 2: Write Operation, and Assertion of FULL Signal

The FULL signal pin is asserted when the FIFO is full.

- At time T_{FDCK_DI}, before clock event 2 (WRCLK), data 04 becomes valid at the DI inputs of the FIFO.
- Write enable remains asserted at the WREN input of the FIFO.
- At time T_{FCKO_FULL}, one clock cycle after clock event 2 (WRCLK), FULL is asserted at the FULL output pin of the FIFO.

If the FIFO is full and a read followed by a write is performed, the FULL signal remains asserted.

Clock Event 3: Write Operation and Assertion of Write Error Signal

The write error signal pin is asserted when data going into the FIFO is not written because the FIFO is in a FULL state.

- At time T_{FDCK_DI}, before clock event 3 (WRCLK), data 05 becomes valid at the DI inputs of the FIFO.
- Write enable remains asserted at the WREN input of the FIFO.
- At time T_{FCKO_WRERR}, after clock event 3 (WRCLK), a write error is asserted at the WRERR output pin of the FIFO. Data 05 is not written into the FIFO.

Clock Event 4: Write Operation and Deassertion of Write Error Signal

WRERR) is deasserted when a user stops trying to write into a full FIFO.

- At time T_{FCCK_WREN}, before clock event 4 (WRCLK), write enable is deasserted at the WREN input of the FIFO.
- At time T_{FCKO_WRERR}, after clock event 4 (WRCLK), write error is deasserted at the WRERR output pin of the FIFO.

The write error signal is asserted/deasserted at every write-clock positive edge. As long as both the write enable and FULL signals are true, write error will remain asserted.

Case 3: Reading From a Full FIFO

Prior to the operations performed in Figure 4-19, the FIFO is completely full.

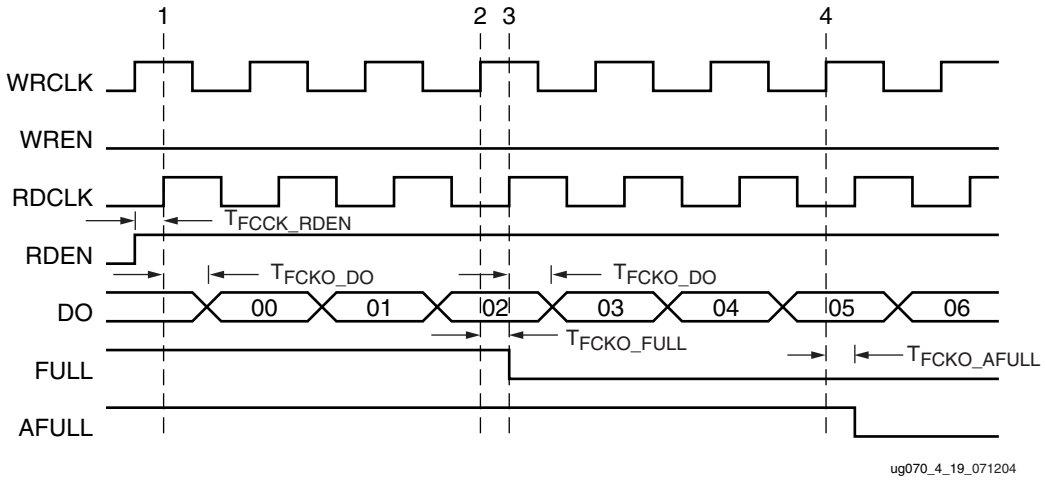


Figure 4-19: **Reading From a Full FIFO**

Clock Event 1 and Clock Event 2: Read Operation and Deassertion of Full Signal

During a read operation on a full FIFO, the content of the FIFO at the first address is asserted at the DO output pins of the FIFO. Three write-clock cycles later, the FULL pin is deasserted when the FIFO is no longer full.

The example in Figure 4-19 reflects both standard and FWFT modes. Clock event 1 is with respect to read-clock, while clock event 2 is with respect to write-clock. Clock event 2 appears three write-clock cycles after clock event 1.

- At time T_{FCCK_RDEN}, before clock event 1 (RDCLK), read enable becomes valid at the RDEN input of the FIFO.
- At time T_{FCKO_DO}, after clock event 1 (RDCLK), data 00 becomes valid at the DO inputs of the FIFO.
- At time T_{FCKO_FULL}, after clock event 2 (WRCLK), FULL is deasserted.

If the rising RDCLK edge is close to the rising WRCLK edge, AFULL could be deasserted one WRCLK period later.

Clock Event 3 and Clock Event 4: Read Operation and Deassertion of ALMOSTFULL Signal

Three write-clock cycles after the fourth data is read from the FIFO, the ALMOSTFULL pin is deasserted to signify that the FIFO is not in the ALMOSTFULL state.

The example in Figure 4-19 reflects both standard and FWFT modes. Clock event 3 is with respect to read-clock, while clock event 4 is with respect to write-clock. Clock event 4 appears three write-clock cycles after clock event 3.

- Read enable remains asserted at the RDEN input of the FIFO.
- At time T_{FCKO_DO}, after clock event 3 (RDCLK), data 03 becomes valid at the DO outputs of the FIFO.
- At time T_{FCKO_AFULL}, after clock event 4 (RDCLK), ALMOSTFULL is deasserted at the AFULL pin.

There is minimum time between a rising read-clock and write-clock edge to guarantee that AFULL will be deasserted. If this minimum is not met, the deassertion of AFULL can take an additional write clock cycle.

Case 4: Reading From an Empty or Almost Empty FIFO

Prior to the operations performed in Figure 4-20, the FIFO is almost completely empty. In this example, the timing diagram reflects standard mode. For FWFT mode, data at DO appears one read-clock cycle earlier.

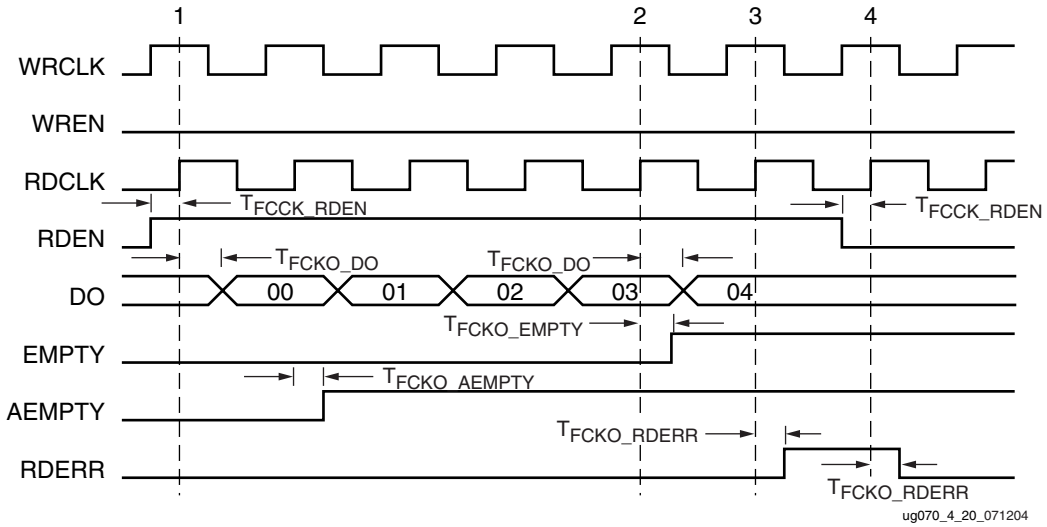


Figure 4-20: **Reading From an Empty / Almost Empty FIFO (Standard Mode)**

Clock Event 1: Read Operation and Assertion of ALMOSTEMPTY Signal

During a read operation to an almost empty FIFO, the ALMOSTEMPTY signal is asserted.

- At time T_{FCCK_RDEN}, before clock event 1 (RDCLK), read enable becomes valid at the RDEN input of the FIFO.
- At time T_{FCKO_DO}, after clock event 1 (RDCLK), data 00 becomes valid at the DO outputs of the FIFO.
- At time T_{FCKO_AEMPTY}, one clock cycle after clock event 1 (RDCLK), ALMOSTEMPTY is asserted at the AEMPTY output pin of the FIFO.

Clock Event 2: Read Operation and Assertion of EMPTY Signal

The EMPTY signal pin is asserted when the FIFO is empty.

- Read enable remains asserted at the RDEN input of the FIFO.
- At time T_{FCKO_DO}, after clock event 2 (RDCLK), data 04 (last data) becomes valid at the DO outputs of the FIFO.
- At time T_{FCKO_EMPTY}, after clock event 2 (RDCLK), Empty is asserted at the EMPTY output pin of the FIFO.

In the event that the FIFO is empty and a write followed by a read is performed, the EMPTY signal remains asserted.

Clock Event 3: Read Operation and Assertion of Read Error Signal

The read error signal pin is asserted when there is no data to be read because the FIFO is in an EMPTY state.

- Read enable remains asserted at the RDEN input of the FIFO.

- At time T_{FCKO_RDERR}, after clock event 3 (RDCLK), read error is asserted at the RDERR output pin of the FIFO.
- Data 04 remains unchanged at the DO outputs of the FIFO.

Clock Event 4: Read Operation and Deassertion of Read Error Signal

The read error signal pin is deasserted when a user stops trying to read from an empty FIFO.

- At time T_{FCCK_RDEN}, before clock event 4 (RDCLK), read enable is deasserted at the RDEN input of the FIFO.
- At time T_{FCKO_RDERR}, after clock event 4 (RDCLK), read error is deasserted at the RDERR output pin of the FIFO.

The read error signal is asserted/deasserted at every read-clock positive edge. As long as both RDEN and EMPTY are true, RDERR will remain asserted.

Case 5: Resetting All Flags

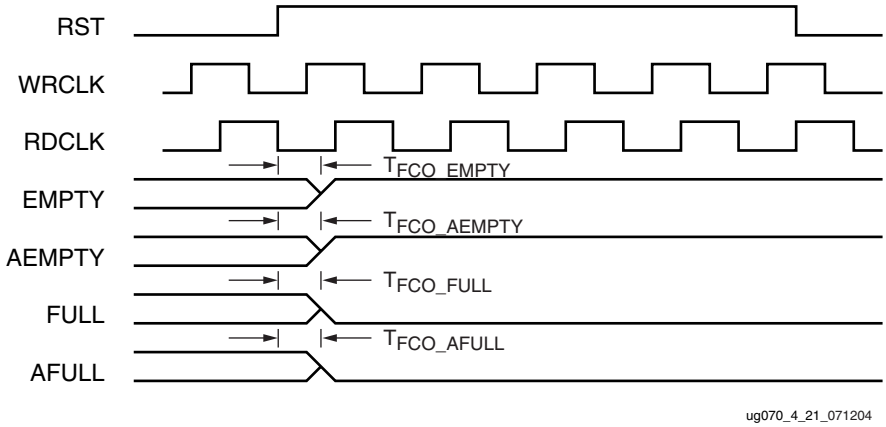


Figure 4-21: **Resetting All Flags**

When the reset signal is asserted, all flags are reset.

- At time T_{FCO_EMPTY}, after reset (RST), EMPTY is asserted at the EMPTY output pin of the FIFO.
- At time T_{FCO_AEMPTY}, after reset (RST), ALMOSTEMPTY is asserted at the AEMPTY output pin of the FIFO.
- At time T_{FCO_FULL}, after reset (RST), full is deasserted at the FULL output pin of the FIFO.
- At time T_{FCO_AFULL}, after reset (RST), ALMOSTFULL is deasserted at the AFULL output pin of the FIFO.

Reset is an asynchronous signal used to reset all flags. Hold the reset signal High for three read and write clock cycles to ensure that all internal states and flags are reset to the correct value.

FIFO Applications

There are various uses for the Virtex-4 FPGA block RAM FIFO:

- Cascading two asynchronous FIFOs to form a deeper FIFO
- Building wider asynchronous FIFO by connecting two FIFOs in parallel.

Cascading FIFOs to Increase Depth

Figure 4-22 shows a way of cascading FIFOs to increase depth. The application sets the first FIFO in FWFT mode, and uses external resources to connect to the second FIFO. The ALMOST_FULL_OFFSET of the second FIFO should be four or more. The data latency of this application can be up to double that of a single FIFO, and the maximum frequency is limited by the feedback path. The NOR gate is implemented using CLB logic.

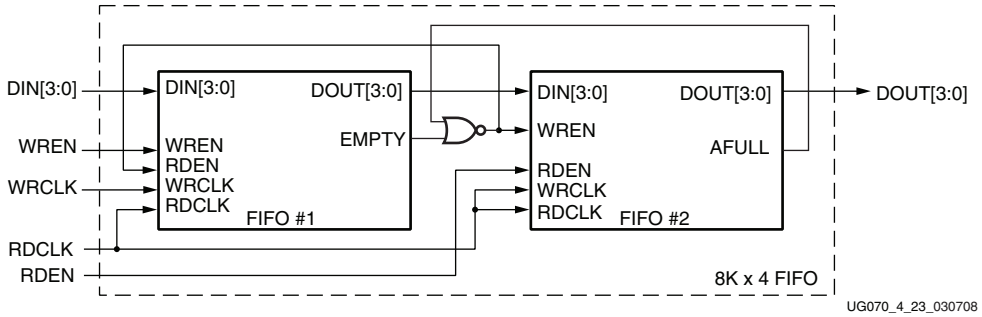


Figure 4-22: **Cascading FIFO**

Cascading FIFOs to Increase Width

As shown in Figure 4-23, the Virtex-4 FPGA FIFO can be cascaded to add width to the design. CLB logic is used to implement the AND/OR gates. The maximum frequency can be limited by the logic gate feedback path.

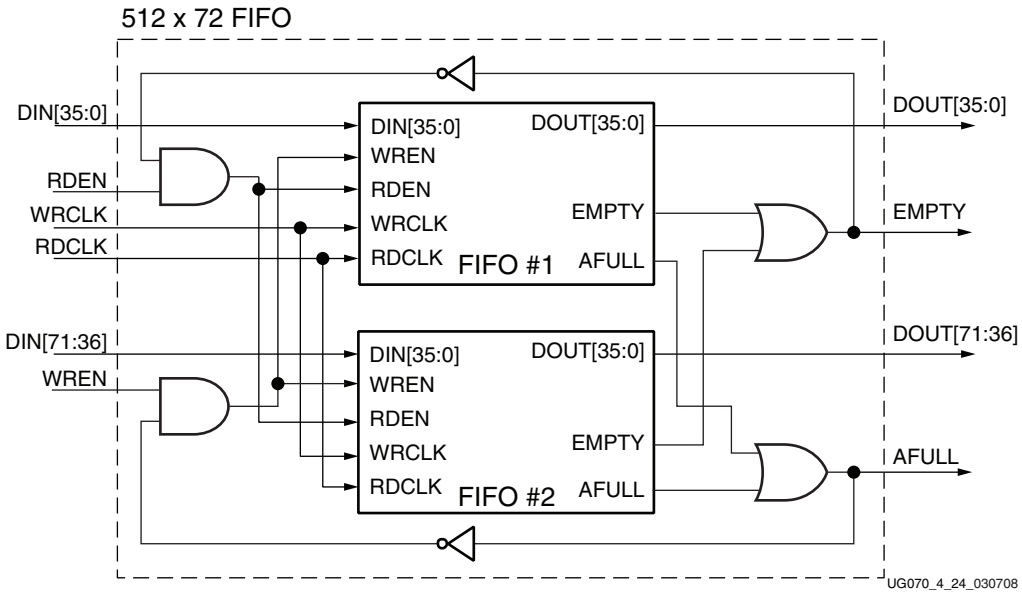


Figure 4-23: **Cascading FIFO by Width**

FIFO16 Error Condition and Work-Arounds

The FIFO16 flags (ALMOSTFULL, FULL, ALMOSTEMPTY, EMPTY), after a very specific sequence of events, transition into a state in which they operate incorrectly. Erroneous settings of the FULL and EMPTY flags can jeopardize even basic FIFO functionality. This section details the error condition and describes synchronous and asynchronous clock work-arounds available to ensure robust operation under all operating conditions. Three different solutions are described in this section. The solution summary section lists the criteria to be used while choosing a particular solution.

FIFO16 Error Condition

The basic Virtex-4 FPGA FIFO16 ceases to correctly generate the ALMOSTEMPTY and EMPTY flags, after the following sequence occurs:

1. A sequence of read and/or write operations makes the number of words in the FIFO equal to the ALMOST_EMPTY_OFFSET threshold (either coming from a higher level as a result of a read operation, or from a lower level as a result of a write operation). This is then followed by either a write or a read operation.
2. If (and only if) the operation immediately following this particular read or write operation is a simultaneous read/write operation, where the enabled read and write active clock edges are coincident or very close (<500 ps) together, the ALMOSTEMPTY flag is incorrect. Since ALMOSTEMPTY is a condition for decoding EMPTY, the EMPTY flag is also wrong.

A similar sequence of operations around the ALMOST_FULL_OFFSET ceases to generate correct ALMOSTFULL and FULL flags.

Solution 1: Synchronous/Asynchronous Clock Work-Arounds

Synchronous Clock Work-Around

In a synchronous design, simultaneous operation can be avoided by offsetting the read and write clocks by about 1 ns. This is easily achieved by using opposite clock edges for the read and write clocks. In most applications, this requires data resynchronization registers to bring read and write back together in the same clock domain. Figure 4-24 illustrates the concept.

This resynchronization must be done on the input side so that the critical EMPTY flag avoids any latency. The FULL flag is eliminated, as it would not be useful with its 2-clock latency; ALMOSTFULL should be used instead. The connections between the input registers and the FIFO16 must be tightly constrained, as this part of the circuit effectively runs at twice the clock rate.

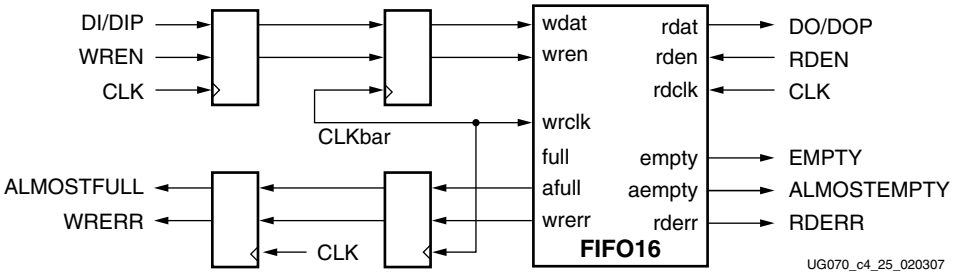


Figure 4-24: **Synchronous Clock Work-Around**

Asynchronous Clock Work-Around

In an asynchronous design, it is inevitable that the two clocks occasionally come very close (<500 ps) to each other, which might cause the problem described above, and no clock delay manipulation can then avoid this problem. For this case, Xilinx has developed a solution that uses additional circuitry to ensure that the FIFO16 never gets into the erred state. This solution operates in a similar manner to the basic FIFO16, and works under all conditions and clock frequencies.

The composite FIFO adds a small LUTFIFO, acting as an asynchronous buffer, that allows the FIFO16 to always operate in synchronous mode. It is necessary to connect the faster clock to the FIFO16 port so that the smaller LUTFIFO never becomes a bottleneck. This constraint leads to two separate designs, as shown in Figure 4-25 and Figure 4-26.

In a case where it is unknown which clock is faster, the "WRCLK faster than RDCLK" design should be used. This design works for any clock frequency combination, including WRCLK faster than RDCLK, WRCLK identical to and/or phase-shifted with respect to RDCLK, and even if the WRCLK and RDCLK relationship is unknown. When this design is used, and RDCLK is faster than WRCLK in the system, it is possible for the EMPTY flag to assert before the ALMOSTEMPTY flag asserts (note that if the two clocks are nominally the same, this does not occur). This is because the intra-FIFO control logic is running off of WRCLK which is designated as the faster clock, but is really the slower clock in the system. This does not cause data corruption or incorrect FIFO behavior in any other manner. If this situation exists and this behavior is not acceptable, the CORE Generator tool FIFO Generator Block RAM work-around described below is recommended.

Some additional logic controls the transfer of data between the two FIFOs for both designs. Resynchronization of specific signals and handshaking between the two FIFOs results in a small uncertainty of the composite FIFO depth and of the ALMOST_FULL_OFFSET and ALMOST_EMPTY_OFFSET. Refer to Table 4-15 for details.

WRCLK Faster than RDCLK Design

In this case (shown in Figure 4-25), the FIFO WRCLK is connected to WRCLKFIFO16. RDCLKFIFO16 and WRCLKLUTFIFO are driven from WRCLKbar, which is a 180-degree phase-shifted version of WRCLK. The FIFO RDCLK is connected to RDCLKLUTFIFO. FIFO16 forms the write interface of the composite FIFO; its read side is clocked by the inverted write clock, which is also used to write into the small LUTFIFO.

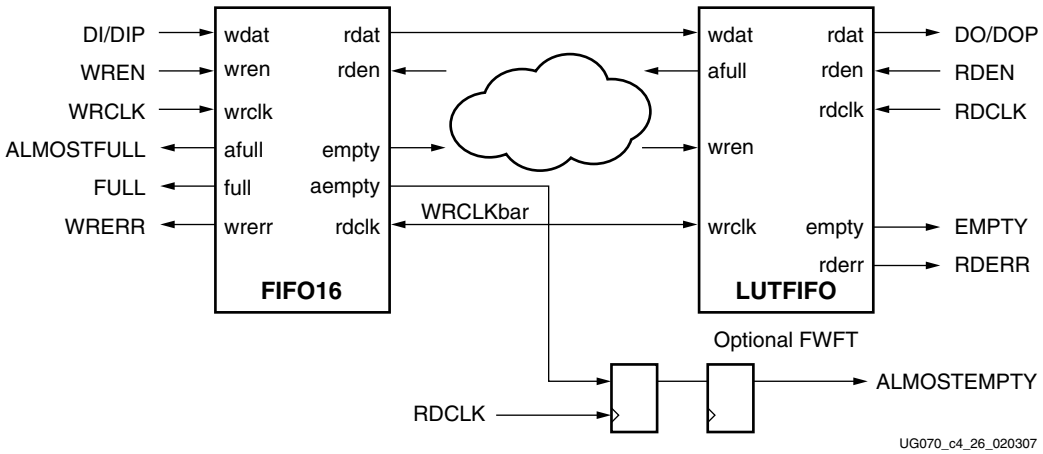


Figure 4-25: **WRCLK Faster than RDCLK Design**

RDCLK Faster than WRCLK Design

In this case (shown in Figure 4-26), the WRCLK of the FIFO is connected to WRCLKLUTFIFO. The RDCLKLUTFIFO and WRCLKFIFO16 are driven from RDCLKbar, which is a 180-degree phase-shifted version of RDCLK. The RDCLK of the FIFO is connected to RDCLKFIFO16. The LUTFIFO forms the write interface of the composite FIFO; its read side is clocked by the inverted read clock, which is also used to write into the FIFO16. LUTFIFO flags are combined and synchronized to the write clock to generate the ALMOSTFULL flag.

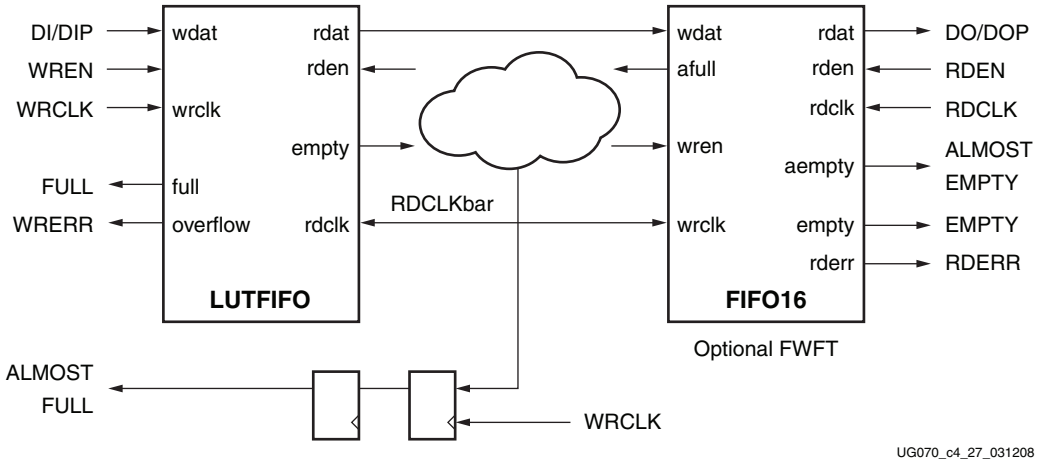


Figure 4-26: **RDCLK Faster than WRCLK Design**

User-Programmable Flag Settings in the Composite FIFO

The offset ranges for user-programmable ALMOSTEMPTY and ALMOSTFULL flags along with the FIFO capacity are listed in Table 4-15. Since the full capacity of any FIFO is normally not critical, most applications use the ALMOSTFULL flag not only as a warning but also as a signal to stop writing. The ALMOSTEMPTY flag can be used as a warning that the FIFO is approaching EMPTY, but to ensure that the very last entries in the FIFO are read out, reading should be continued until EMPTY is asserted.

When setting the offset ranges in the provided Perl script (refer to Design Files below), use decimal notation.

Table 4-15: **FIFO Capacity and Effective ALMOSTFULL/ALMOSTEMPTY Flag Offsets**

FIFO Type	Standard/FWFT	
FIFO Depth	FIFO16(1) + 15	
Clock Style	WRCLK > RDCLK	RDCLK > WRCLK
ALMOST_FULL_OFFSET	AF_{FIFO16}(2) + 15	15
ALMOST_EMPTY_OFFSET	AE_{FIFO16}(3) + 15	AE_{FIFO16}(3)

Notes:

1. FIFO16 = Capacity of FIFO16. Refer to Table 4-9, "FIFO Capacity."
2. AF_{FIFO16} = Set by user in Perl script. Sets the FIFO16 ALMOST_FULL_OFFSET. Refer to Table 4-13.
3. AE_{FIFO16} = Set by user in Perl script. Sets the FIFO16 ALMOST_EMPTY_OFFSET. Refer to Table 4-13.

All values can vary by up to 3 words, depending on the read/write clock rates and the read/write patterns.

Status Flags

Although the functionality of the status flags on the composite FIFO remain the same, the assertion/deassertion latencies for some of the signals have increased. The assertion values for key signals have remained the same as on the FIFO16 (EMPTY, FULL, ALMOSTEMPTY, ALMOSTFULL, RDERR, and WRERR). Table 4-16 lists the latency values for the status flags. Also note that the values have an uncertainty that is affected by the frequency ratios of the read/write clock, as well as the read/write patterns.

Table 4-16: **Clock Cycle Latency for Status Flag Assertion and Deassertion**

FIFO Type	Standard/FWFT			
Clock Style	WRCLK > RDCLK		RDCLK > WRCLK	
Clock Cycle Latency	Assertion	Deassertion	Assertion	Deassertion
EMPTY	0	**10 / 12**(1)	0	**10 / 11**
FULL	1	**9**	0	**9**
ALMOSTEMPTY	**10**	**4**	1	**10**
ALMOSTFULL	1	**9**	**11**	**5**
RDERR	0	0	0	0
WRERR	0	0	0	0

Notes:

1. Latency values in **bold** vary with the ratio between the read/write clock frequencies and read/write pattern. In certain conditions for WRCLK > RDCLK, the ALMOSTEMPTY flag deasserts before the EMPTY flag. This behavior is reflected in simulations, and increasing the ALMOST_EMPTY_OFFSET rectifies the behavior.

Resource Utilization

The design was implemented using the ISE 8.1i software with default settings for MAP, Place, and Route. The approximate LUT count for a x4 design varies from 55 to 70 LUTs. For a x9 design, the LUT count varies from 65 to 80 LUTs, and for a x18 design the LUT count varies from 85 to 100 LUTs. The LUT count for a x36 design varies from 125 to 130 LUTs.

Performance Expressed in Maximum Read and/or Write Clock Frequency

The maximum read and/or write clock rate is >500 MHz for all configurations and modes, except for the 512 x 36 configuration with write clock > read clock, where the max frequency for standard mode is 473 MHz, and for FWFT mode it is 488 MHz.

CORE Generator Tool Implementation

The CORE Generator tool should be used to implement this solution. FIFO Generator (v3.2 and above) automatically implements the work-arounds detailed above. The device utilization is detailed in the core data sheet, which can be accessed from:

http://www.xilinx.com/bvdocs/ipcenter/data_sheet/fifo_generator_ds317.pdf

Both synchronous and asynchronous FIFOs can be implemented using FIFO Generator block RAM FIFOs available from the CORE Generator tool instead of using the FIFO16 primitives. The block RAM-based implementations are slower than FIFO16-based implementations because the FIFO control logic is implemented in the fabric of the device.

The FIFO16 built-in FIFO configurations from the FIFO Generator Core incurs the same issues described above.

Note: When the script is used, RDCOUNT and WRCOUNT might not be an accurate representation of the number of bits read from and written to the FIFO.

Please review the FIFO Generator Data Sheet for more information:

http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?sGlobalNavPick=PRODUCTS&sSecondaryNavPick=Intellectual+Property&key=FIFO_Generator

Software Updates

Starting with ISE 8.1i Service Pack 1 software, the tools automatically detect when a synchronous FIFO16 (RDCLK and WRCLK are connected) has been inserted into a design and issue the following warning:

```
WARNING:PhysDesignRules:1447 - FIFO16 XLXI_1 has been found to have
both RDCLK and WRCLK input pins connected to the same source
XLXN_5_BUFGP. Under certain circumstances, the flag behavior to the
FIFO may be undeterministic. Please consult the Xilinx website for
more details.
```

To remove this warning, use the CORE Generator software FIFO solution or the Synchronous FIFO work-around described above.

Software IP Cores

For information on what software IP cores are affected by this issue, check the following page:

http://www.xilinx.com/ipcenter/coregen/advisories/ip_cores_impacted_by_fifo16_ar22462_issue.htm

Solution 2: Work-Around Using a Third Fast Clock

If the frequencies of WRCLK and RDCLK are low enough, it is possible to synchronize FIFO reads and writes to a third asynchronous fast clock (FASTCLK). The ALMOSTFULL and ALMOSTEMPTY flags are generated in this fast clock domain. These flags are then resynchronized to their respective clocks.

The system described in this solution requires a minimum of 2 and a maximum of 3 fast clock cycles to process a single read or write cycle. To handle back-to-back read or writes, the fast process must complete within one RDCLK or WRCLK period. Thus, the fast clock must be at least three times faster than the faster of WRCLK and RDCLK.

For example, if the fastest RDCLK or WRCLK is 125 MHz, then FASTCLK could be 400 MHz (400/125 = 3.2).

Design Description

The circuit shown in Figure 4-27 is used to generate the "words in FIFO" (WIF) signal. The Up/Down counter must be large enough to hold the maximum number of words in the FIFO; e.g., 10 bits wide if the FIFO depth is 512 words.

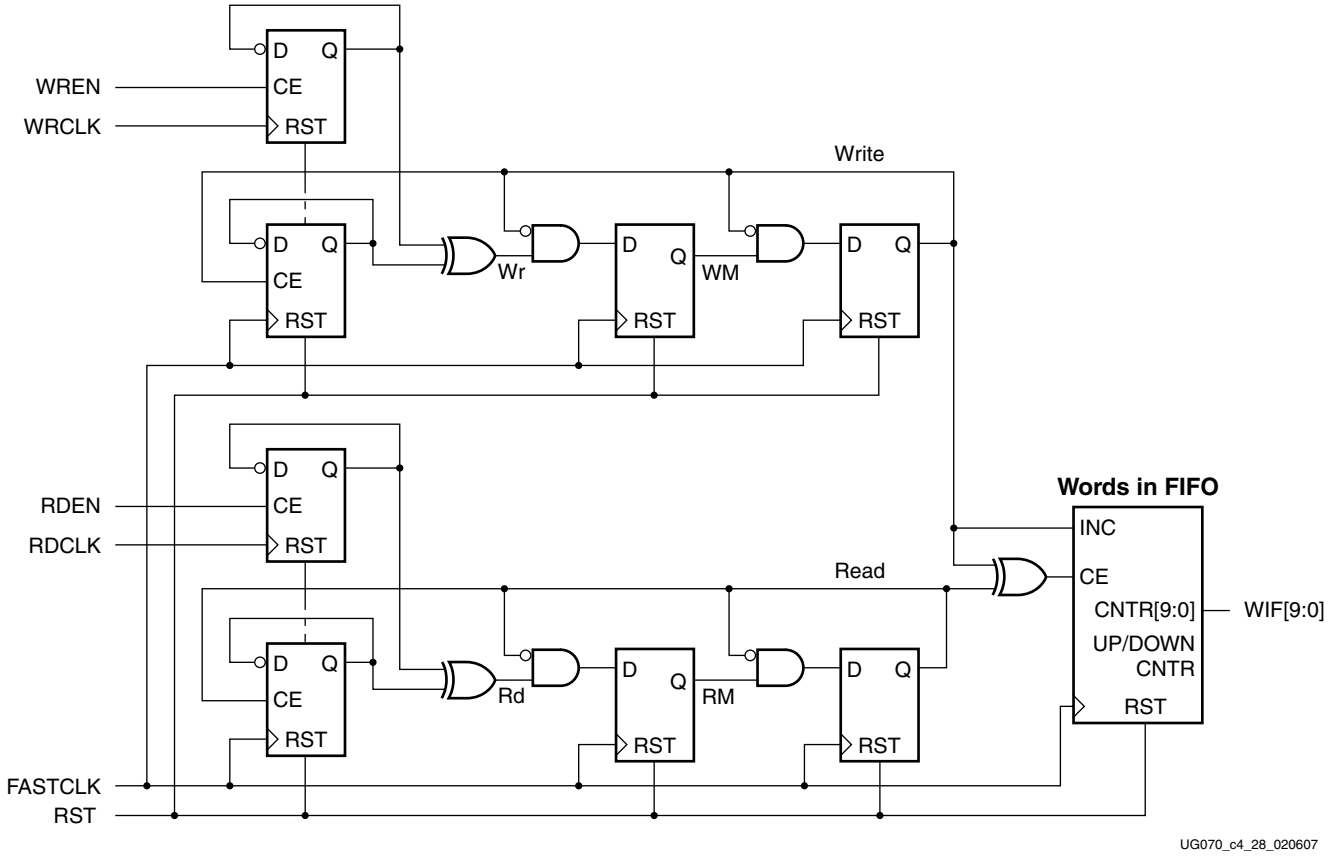


Figure 4-27: **WIF Signal Generation**

The WIF signal is used along with the ALMOST_EMPTY_OFFSET to generate the ALMOST_FULL and ALMOST_EMPTY flags, as shown in Figure 4-28.

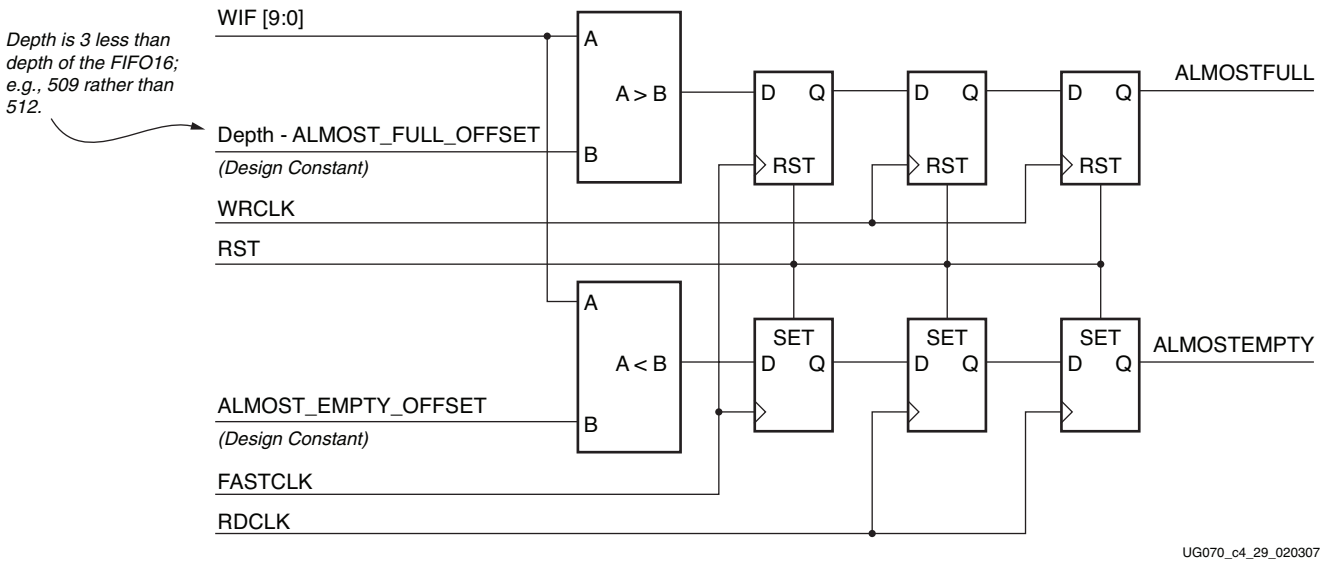


Figure 4-28: **ALMOSTFULL and ALMOSTEMPTY Signal Generation**

For this design to work, the ALMOST_FULL_OFFSET and ALMOST_EMPTY_OFFSET for the FIFO16 instantiations must be fixed values as shown below:

```
defparam fifo16.ALMOST_FULL_OFFSET  = 12'h001; // do not change this line
defparam fifo16.ALMOST_EMPTY_OFFSET= 12'h1FE; // set this to FIFO16 depth - 2
```

The FIFO16 configurations supported are 4K x 4, 2K x 9, 1K x 18, and 512 x 36.

Two DCMs can be used to generate the FASTCLK as shown in Figure 4-29.

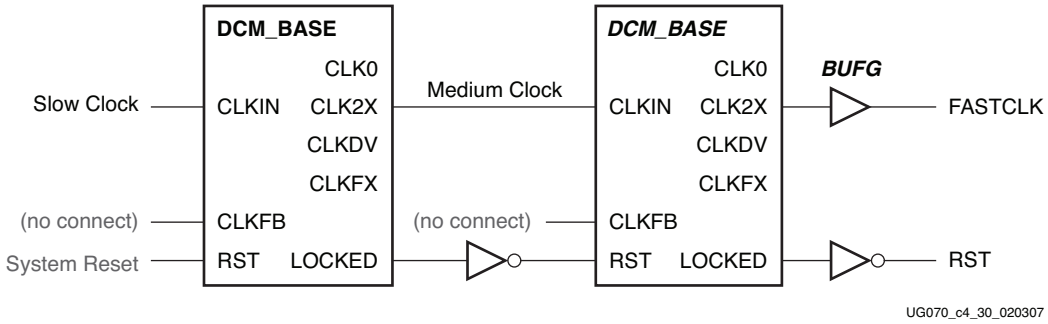


Figure 4-29: **FASTCLK Generator**

In some cases, only one DCM is needed to generate FASTCLK. The FASTCLK signal should be connected to all instances of module `fifo_third_clk_flags` in the design. The output RST signal is connected to all FIFOs and all instances of module `fifo_third_clk_flags`. Clock feedback must be specified as NONE on both DCMs (defparam dcm.CLK_FEEDBACK = "NONE").

Notes:

- The ALMOSTEMPTY flag is delayed from 1 to 2 RDCLK periods after the condition is detected.
- The ALMOSTFULL flag is delayed from 1 to 2 WRCLK periods after the condition is detected.
- The DCM generating the FASTCLK clock must be locked before the FIFOs can be used. (The STARTUP_WAIT attribute can be used to make sure that the DCMs are locked before the configuration is done.)
- The FASTCLK clock must be continuously available when any of the FIFOs in the system are being used. (Monitor the LOCK signals from all the DCMs to make sure that the FASTCLK clock is running. If LOCK goes Low, the DCMs should be reset.)
- For this design to work properly the maximum words in the FIFO16 must never exceed the nominal maximum - 3; e.g., a 512 word FIFO must never contain more than 509 words.
- This work-around does not currently provide a FULL flag. However, the EMPTY flag from the FIFO16 can be used.

Timing Diagram

The timing diagram for the worst-case write condition is shown in Figure 4-30. The diagram depicts two back-to-back FIFO write cycles. This is a "worst-case" diagram, because the rising edge of WRCLK slightly trails the rising edge of FASTCLK when write enable (WREN) is TRUE. Please refer to Figure 4-27 for signals Wr and WM. Signal Wr is asynchronous to FASTCLK and the leading edge of WM might be metastable. FASTCLK and WRCLK depictions are drawn to scale, relative to each other.

The Read timing is similar to the Write timing shown in Figure 4-30.

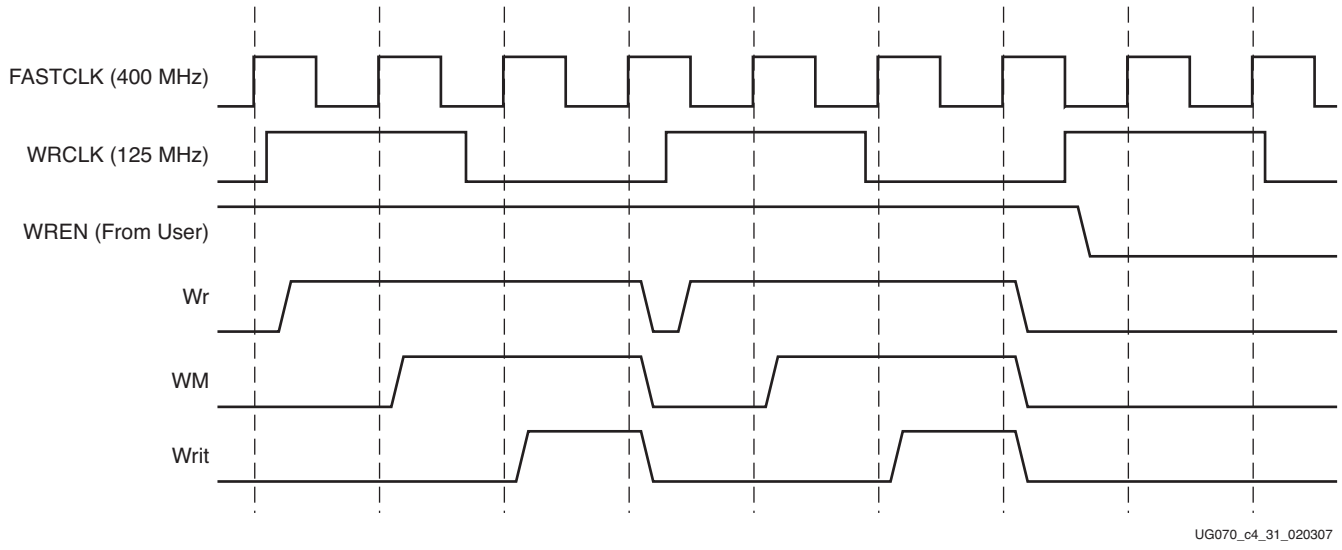


Figure 4-30: **Write Timing Diagram**

Resource Utilization

The resources used in implementing the solution described above with a 400 MHz FASTCLK are as follows.

The design was implemented using the ISE 8.1i software with default settings for MAP, Place, and Route. The approximate resource count was 20 LUTs and 24 flip-flops per FIFO. One DCM is required to generate a 400 MHz clock if a 200 MHz input clock is available. Two DCMs are needed if only a 100 MHz input clock is available. One extra BUFG was used per device.

Performance

The maximum FASTCLK frequency for each speed grade is Global Clock Tree FMAX, as given in the *Virtex-4 Data Sheet*. If any back-to-back reads or writes occur, the maximum RDCLK and WRCLK frequency is 1/3 the FASTCLK frequency. If the system design guarantees that there is at least one clock cycle between all reads and all writes, then the maximum RDCLK and WRCLK frequency is 2/3 the FASTCLK frequency. If the system design guarantees that there are at least two clock cycles between all reads and all writes, then the RDCLK and WRCLK frequency can be equal to the FASTCLK frequency.

Design Files

All the necessary files required for the above design are contained in a ZIP archive downloadable from the Xilinx website at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=30163

Open the ZIP archive and extract `FIFO16_solution2.zip`.

Solution 3: FIFO Flag Generator Using Gray Code

The incorrect operation of the FIFO16 after a specific sequence of events occurs only on the flag signals. Once the flag signals are incorrect, the FIFO operation itself can be affected.

In Solution 3, the FIFO flags are generated outside the FIFO16. The externally generated flags are used in conjunction with the FIFO16 to give the complete FIFO solution. This solution can also be used if the customer design has the read or write clock stopped in between during the FIFO operation.

In the solution described here, the FIFO memory address space is divided into 16 sectors. The number of word in each sector depends on the FIFO depth and is given by FIFO Depth/ 16. The granularity of this solution is equal to the number of words in each sector.

Design Description

Four bits are used to identify the 16 sectors within the memory. The four bits are the four MSB bits of the WRCOUNT signal from the FIFO16.The ALMOSTFULL flag goes true if the current read sector is equal to the current write sector + 1, 2, or 3. Three binary to gray code converters uses the four WRCOUNT MSB bits to convert from binary to gray +1, gray +2, and gray +3 values as shown in Figure 4-31.

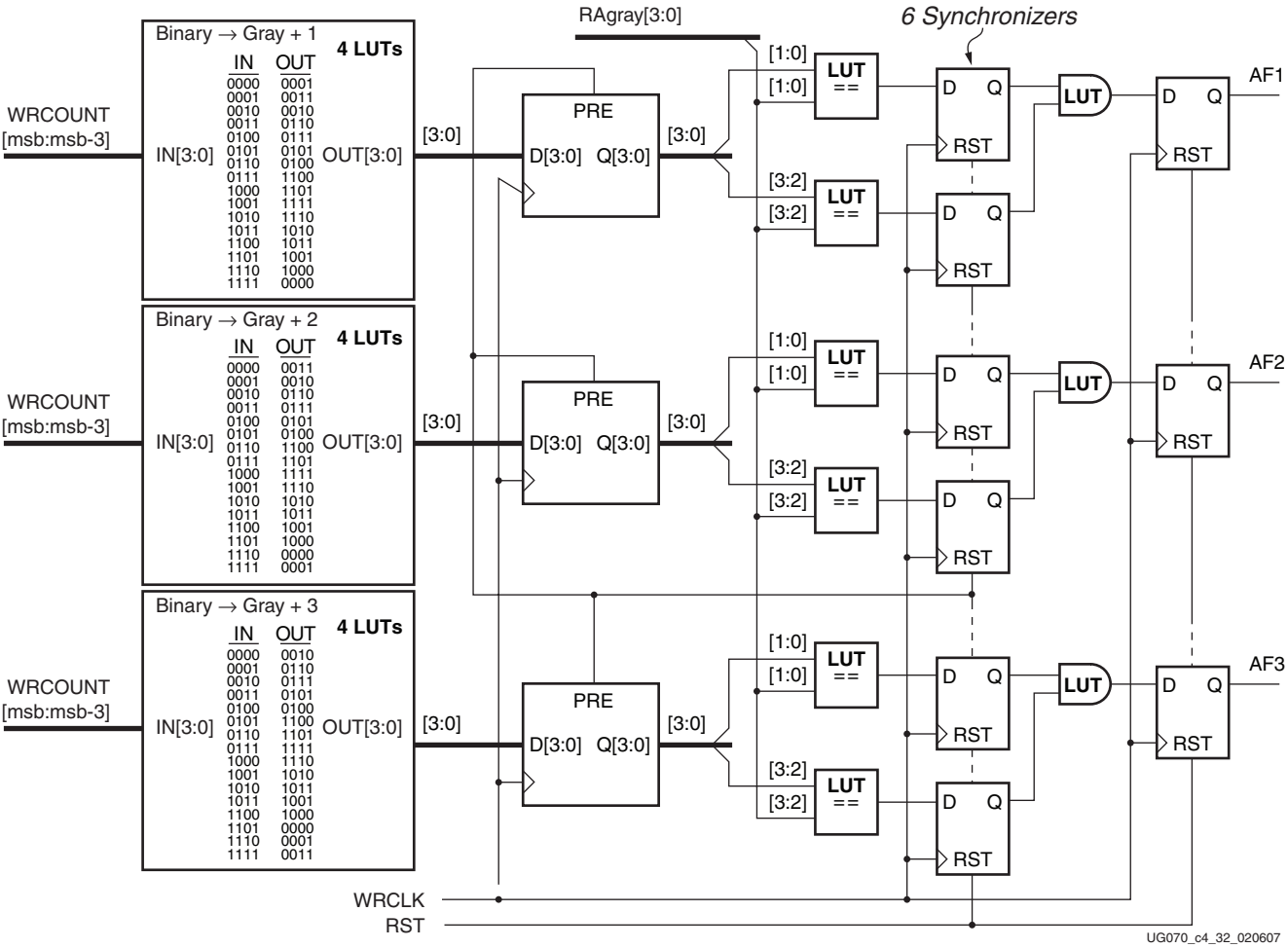


Figure 4-31: **Intermediate Signal Generation for ALMOSTFULL Flag**

Flag AF1, AF2, or AF3 goes High if the read sector (RAgray) is equal to write counter sector one, two, or three respectively. A High on one of these flags sets the ALMOSTFULL flag High as shown in Figure 4-32.

Because the circuit is operating in two clock domains (RDCLK and WRCLK), there is a possibility that the OR of AF1, AF2, and AF3 could spike FALSE as one of these signals transitions FALSE and another transitions TRUE. To prevent ALMOSTFULL from erroneously going FALSE when this occurs, a flip-flop is added to the circuit. This flip-flop adds the requirement that the OR output must be FALSE for two consecutive WRCLK periods for ALMOSTFULL to go FALSE. The ALMOSTEMPTY circuit works in a similar fashion.

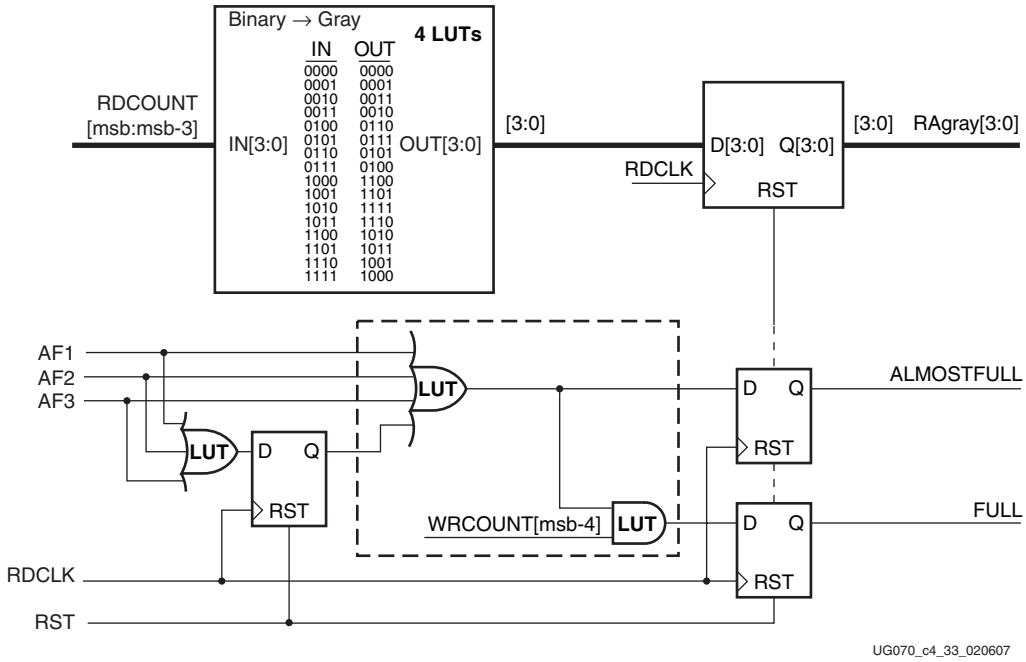


Figure 4-32: **ALMOSTFULL and FULL Flag Generation**

The FULL flag goes true when the above ALMOSTFULL flag conditions are true AND the next most significant bit of WRCOUNT is a one.

If WRCLK is halted, ALMOSTFULL and FULL are frozen in their current states. When WRCLK restarts, these flags can switch, subject to the delays specified above. (Note that this is the behavior of all asynchronous FIFOs, because the ALMOSTFULL and FULL flags are always synchronous to the write clock.)

The ALMOSTEMPTY flag goes true if the current read sector is equal to the current write sector, or if the current read sector is equal to the current write sector – 1. ALMOSTEMPTY flag is generated similar to the ALMOSTFULL flag and is shown in Figure 4-33.

Setting of the ALMOSTEMPTY flag occurs between two and three RDCLK periods after an equality comparison goes true.

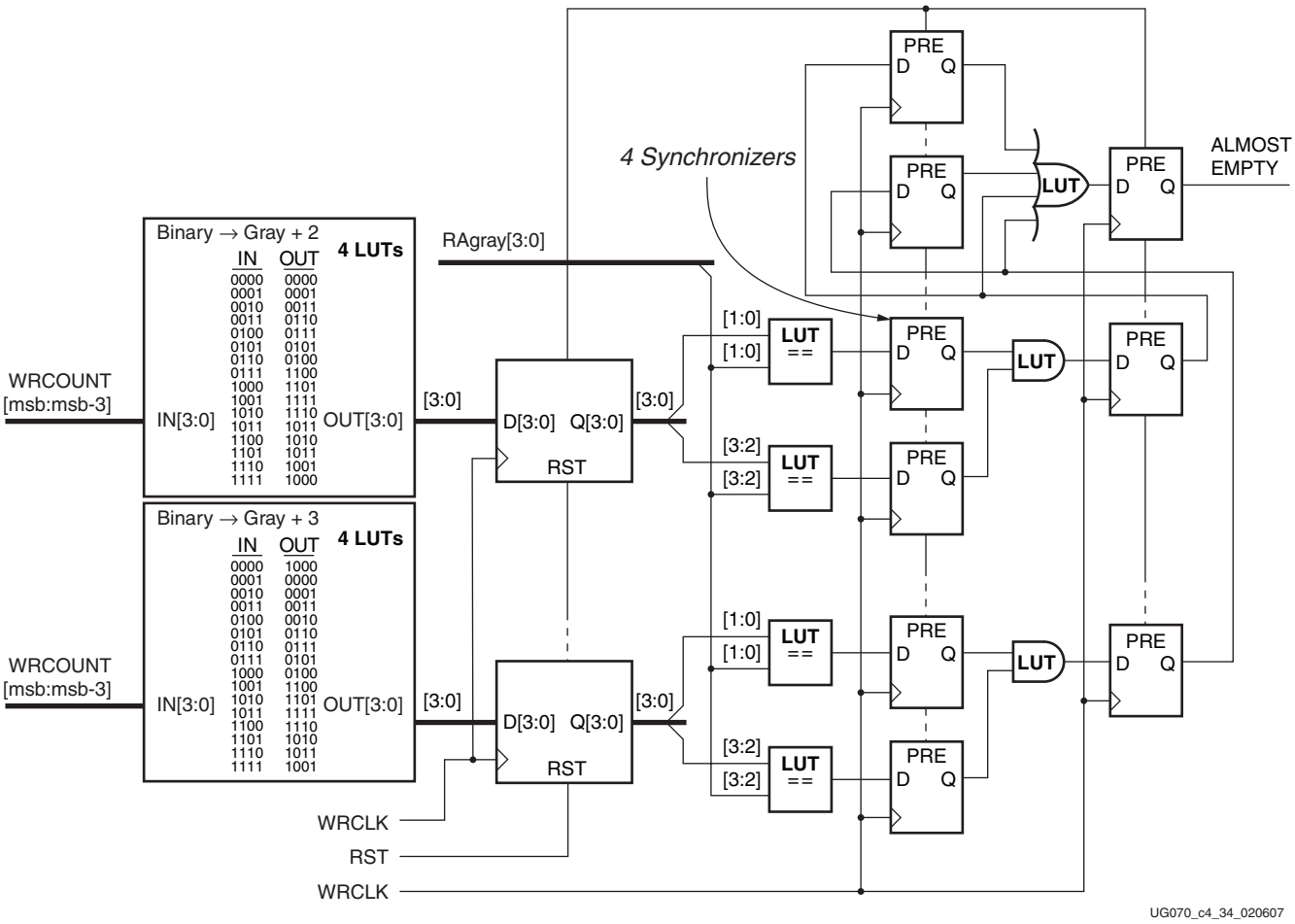


Figure 4-33: **ALMOSTEMPTY Flag Generation**

Notes:

- The FULL flag does not mean that the FIFO is full; it means that half of the sector has been written into after the ALMOSTFULL flag went true.
- Setting and clearing of ALMOSTFULL occurs between two and three WRCLK periods after equality goes true or false, respectively in Figure 4-32.
- Setting and clearing of the FULL flag is delayed as with the ALMOSTFULL flag.
- Clearing of the ALMOSTEMPTY flag occurs between 3 and 4 RDCLK periods after an equality goes false in Figure 4-33.

Resource Utilization

In this design, the combinatorial logic delay between flip-flops is never comprised of more than one LUT. A total of 39 LUTs and 41 flip-flops are used.

Performance

The performance of this logic matches the performance of the FIFO16 module for each speed grade as given in the *Virtex-4 Data Sheet*.

Design Files

All the necessary files required for the above design are contained in a ZIP archive downloadable from the Xilinx website at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=30163

Open the ZIP archive and extract `FIFO16_solution3.zip`.

Solution Summary

The following criteria can be used to choose a particular solution for the design.

- "Solution 1: Synchronous/Asynchronous Clock Work-Arounds" should be used if:
  - Design is currently supported in the CORE Generator tool
  - Design is required to run at the maximum FIFO16 clock rates
  - Exact values are required for the ALMOSTEMPTY and ALMOSTFULL Flags
  - Resource utilization is more than that for Solution 2 and Solution 3 (see Solution 1 for details)
  - Continuous RDCLK and WRCLK are available after RST
- "Solution 2: Work-Around Using a Third Fast Clock" should be used if:
  - Smallest resource utilization is required
  - RDCLK and WRCLK needs to be intermittently stopped after RST
  - Design is not required to run at the maximum FIFO16 clock rates (see Solution 2 for more details)
  - The generation of a third continuous fast clock is feasible
  - ALMOSTEMPTY and ALMOSTFULL flags can be delayed by from 1 to 2 RDCLK or WRCLK periods, respectively
- "Solution 3: FIFO Flag Generator Using Gray Code" should be used if:
  - Design is required to run at the maximum FIFO16 clock rates
  - Resource utilization smaller than Solution 1 is required
  - RDCLK and WRCLK needs to be intermittently stopped after RST
  - ALMOSTEMPTY and ALMOSTFULL flags need not be exact and can be within a range.

Built-in Block RAM Error Correction Code

Two vertically adjacent block RAMs can be configured as a single 512 x 64 RAM with built in Hamming error correction, using the extra eight bits in the 72-bit wide RAM. The operation is transparent to the user. The eight protection bits are generated during each write operation, and are used during each read operation to correct any single error, or to detect (but not correct) any double error. Two status outputs indicate the three possible read results: No error, single error corrected, double error detected. The read operation does not correct the error in the memory array, it only presents corrected data on DO.

This error correction code (ECC) configuration option is available with almost all block RAM pairs as long as the lower RAM is instantiated in an even numbered row. However, the ECC configuration cannot use the one block RAM immediately above or below the PowerPC® 405 blocks in Virtex-4 devices.

The functionality of the block RAM is changed when using the ECC mode.

- The two block RAM ports still have independent address, clocks, and enable inputs, but one port is a dedicated write port, and the other is a dedicated read port.
- DO represents the read data after correction.
- DO stays valid until the next active read operation.
- Simultaneous reading and writing, even with asynchronous clocks, is allowed, but requires careful clock timing if read and write addresses are identical.
- The READ_FIRST or WRITE_FIRST modes of the normal block RAM operation are not applicable to the ECC configuration.

Top-Level View of the Block RAM ECC Architecture

Figure 4-34 shows the top-level view of a Virtex-4 FPGA block RAM in ECC mode.

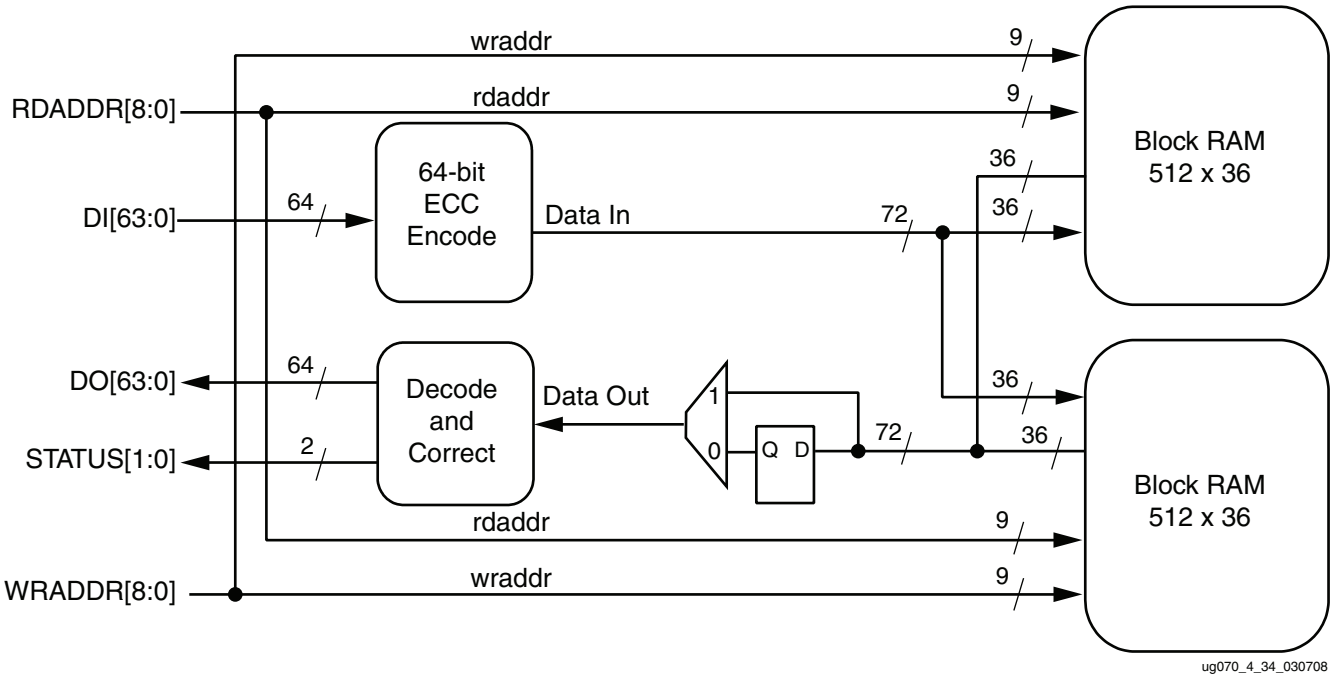


Figure 4-34: **Top-Level View of Block RAM ECC**

Block RAM ECC Primitive

Figure 4-35 shows RAMB32_S64_ECC, the block RAM ECC primitive.

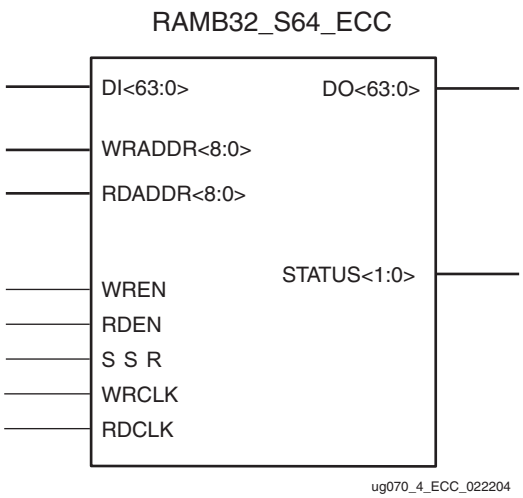


Figure 4-35: **RAMB32_S64_ECC: Block RAM ECC Primitive**

Block RAM ECC Port Description

Table 4-17 lists and describes the block RAM ECC I/O port names.

Table 4-17: **Block RAM ECC Port Names and Descriptions**

Port Name	Direction	Signal Description
DI<63:0>	Input	Data input bus
WRADDR<8:0>	Input	Write address bus
RDADDR<8:0>	Input	Read address bus
WREN	Input	Write enable. When WREN = 1, data will be written into memory. When WREN = 0, write is disabled
RDEN	Input	Read enable. When RDEN = 1, data will be read from memory. When RDEN = 0, read is disabled
SSR	Input	Not supported when using the block RAM ECC primitive. Always connect to GND.
WRCLK	Input	Clock for write operations
RDCLK	Input	Clock for read operations
DO<63:0>	Output	Data output bus
STATUS<1:0>(1)	Output	Error status bus

Notes:

1. Hamming code implemented in the block RAM ECC logic detects one of three conditions: no detectable error, single-bit error detected and corrected on DO (but not corrected in the memory), and double-bit error detected without correction. The result of STATUS<1:0> indicates these three conditions.

Error Status Description

The block RAM ECC is able to detect single- and double-bit errors from the block RAM. However, only the single-bit error can be corrected. The ECC logic does not correct the bit in the actual block RAM storage location. If the block RAM location containing the bit error is not overwritten, then further reads from that location causes the ECC logic to continue to correct the error. Table 4-18 is the truth table for the STATUS bits.

Table 4-18: **STATUS Bit Truth Table**

STATUS[1:0]	Description
00	No bit error.
01	Single-bit error. The block RAM ECC macro detects and automatically corrects a single-bit error.
10	Double-bit error. The block RAM ECC macro detects a double-bit error.
11	Undefined, not a valid status error.

Block RAM ECC Attribute

In addition to the built-in registers in the decode and correct logic, the RAMB32_S64_ECC primitive allows the use of optional pipeline registers to produce higher performance with one additional latency. Valid values for the DO_REG attribute are 0 or 1.

Block RAM ECC VHDL and Verilog Templates

VHDL and Verilog templates are available in the Libraries Guide.

Block RAM ECC VHDL Template

```
-- RAMB32_S64_ECC: To incorporate this function into the design,
--      VHDL     : the following instance declaration needs to be placed
--    instance   : in the architecture body of the design code. The
--   declaration : instance name (RAMB32_S64_ECC_inst) and/or the port
--      code     : declarations after the "=>" assignment can be changed
--               : to properly connect this function to the design.
--               : All inputs and outputs must be connected.
--     Library   : In addition to adding the instance declaration, a
--   declaration : use declaration statement for the UNISIM.v
--       for     : components library needs to be added before the
--     Xilinx    : entity declaration. This library contains the
--    primitives : component declarations for all Xilinx primitives
--               : and points to the models that will be used for
--               : simulation.

--  Copy the following two statements and paste them before the
--  Entity declaration, unless they already exists.

Library UNISIM;
use UNISIM.vcomponents.all;

--  <---Cut code below this line and paste into the architecture body-->

   -- RAMB32_S64_ECC: Virtex-4 512 x 64 Error Correction Block RAM
   -- Virtex-4 FPGA User Guide
```

```
RAMB32_S64_ECC_inst: RAMB32_S64_ECC_inst (
generic map (
   DO_REG => 0, -- Optional output registers (0 or 1)
   port map (
      DO => DO,          -- 64-bit output data
      STATUS => STATUS,  -- 2-bit status output
      DI => DI,          -- 64-bit data input
      RDADDR => RDADDR,  -- 9-bit data address input
      RDCLK => RDCLK,    -- 1-bit read clock input
      RDEN => RDEN,      -- 1-bit read enable input
      SSR => SSR,        -- 1-bit synchronous reset
      WRADDR =>WRADDR,   -- 9-bit write address input
      WRCLK => WRCLK,    -- 1-bit write clock input
      WREN => WREN       -- 1-bit write enable input
   );

-- End of RAMB32_S64_ECC_inst instantiation
```

Block RAM ECC Verilog Template

RAMB32_S64_ECC Verilog:

```
// RAMB32_S64_ECC: To incorporate this function into the design,
//    Verilog    : the following instance declaration needs to be placed
//   instance    : in the body of the design code. The instance name
//  declaration  : (RAMB32_S64_ECC_inst) and/or the port declarations
//     code      : within the parenthesis can be changed to properly
//               : reference and connect this function to the design.
//               : All inputs and outputs must be connected.

//  <-----Cut code below this line---->

   // RAMB32_S64_ECC: Virtex-4 512 x 64 Error Correction Block RAM
   // Virtex-4 FPGA User Guide

   RAMB32_S64_ECC #(
      .DO_REG(0),  // Optional output registers (0 or 1)
    ) RAMB32_S64_ECC_inst (
      .DO(DO),          // 64-bit output data
      .STATUS(STATUS), // 2-bit status output
      .DI(DI),          // 64-bit data input
      .RDADDR(RDADDR), // 9-bit data address input
      .RDCLK(RDCLK),    // 1-bit read clock input
      .RDEN(RDEN),      // 1-bit read enable input
      .SSR(SSR),        // 1-bit synchronous reset
      .WRADDR(WRADDR), // 9-bit write address input
      .WRCLK(WRCLK),    // 1-bit write clock input
      .WREN(WREN)       // 1-bit write enable input
   );

   // End of RAMB32_S64_ECC_inst instantiation
```

Chapter 5

Configurable Logic Blocks (CLBs)

CLB Overview

The Configurable Logic Blocks (CLBs) are the main logic resource for implementing sequential as well as combinatorial circuits. Each CLB element is connected to a switch matrix to access to the general routing matrix (shown in Figure 5-1). A CLB element contains four interconnected slices. These slices are grouped in pairs. Each pair is organized as a column. SLICEM indicates the pair of slices in the left column, and SLICEL designates the pair of slices in the right column. Each pair in a column has an independent carry chain; however, only the slices in SLICEM have a common shift chain.

The Xilinx® tools designate slices with the following definitions. An "X" followed by a number identifies a column of slices. The number counts up in sequence from the left to the right. A "Y" followed by a number identifies the position of each slice in a pair as well as the CLB row. The "Y" number counts slices starting from the bottom in sequence: 0, 1, 0, 1 (the first CLB row); 2, 3, 2, 3 (the second CLB row); etc. Figure 5-1 shows the CLB located in the bottom-left corner of the die. Slices X0Y0 and X0Y1 constitute the SLICEM column-pair, and slices X1Y0 and X1Y1 constitute the SLICEL column-pair. For each CLB, SLICEM indicates the pair of slices labeled with an even number – SLICE(0) or SLICE(2), and SLICEL designates the pair of slices with an odd number – SLICE(1) or SLICE(3).

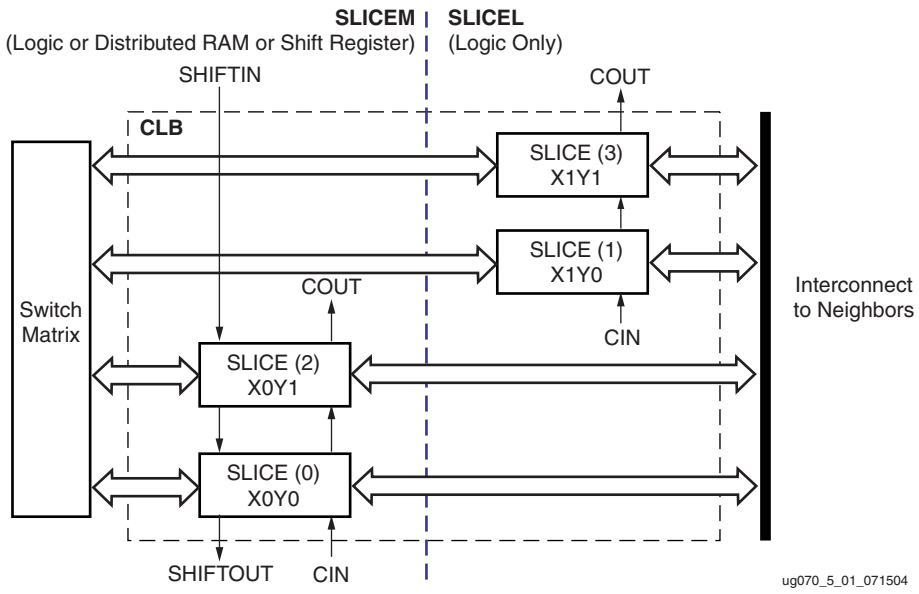


Figure 5-1: **Arrangement of Slices within the CLB**

Slice Description

The elements common to both slice pairs (SLICEM and SLICEL) are two logic-function generators (or look-up tables), two storage elements, wide-function multiplexers, carry logic, and arithmetic gates. These elements are used by both SLICEM and SLICEL to provide logic, arithmetic, and ROM functions. SLICEM supports two additional functions: storing data using distributed RAM and shifting data with 16-bit registers. SLICEM (shown in Figure 5-2, page 185) represents a superset of elements and connections found in all slices. SLICEL is shown in Figure 5-3, page 186.

CLB/Slice Configurations

Table 5-1 summarizes the logic resources in one CLB. All of the CLBs are identical and each CLB or slice can be implemented in one of the configurations listed.Table 5-2 shows the available resources in all CLBs.

Table 5-1: **Logic Resources in One CLB**

Slices	LUTs	Flip-Flops	MULT_ANDs	Arithmetic & Carry Chains	Distributed RAM(1)	Shift Registers(1)
4	8	8	8	2	64 bits	64 bits

Notes:

1. SLICEM only

Table 5-2: **Virtex-4 FPGA Logic Resources Available in All CLBs**

Device	CLB Array: Row x Column	Number of Slices	Number of LUTs	Maximum Distributed RAM or Shift Registers (Kb)	Number of Flip-Flops
XC4VLX15	64 x 24	6,144	12,288	96	12,288
XC4VLX25	96 x 28	10,752	21,504	168	21,504
XC4VLX40	128 x 36	18,432	36,864	288	36,864
XC4VLX60	128 x 52	26,624	53,248	416	53,248
XC4VLX80	160 x 56	35,840	71,680	560	71,680
XC4VLX100	192 x 64	49,152	98,304	768	98,304
XC4VLX160	192 x 88	67,584	135,168	1056	135,168
XC4VLX200	192 x 116	89,088	178,176	1392	178,176
XC4VSX25	64 x 40	10,240	20,480	160	20,480
XC4VSX35	96 x 40	15,360	30,720	240	30,720
XC4VSX55	128 x 48	24,576	49,152	384	49,152
XC4VFX12	64 x 24	5,472	10,944	86	10,944
XC4VFX20	64 x 36	8,544	17,088	134	17,088
XC4VFX40	96 x 52	18,624	37,248	291	37,248
XC4VFX60	128 x 52	25,280	50,560	395	50,560
XC4VFX100	160 x 68	42,176	84,352	659	84,352
XC4VFX140	192 x 84	63,168	126,336	987	126,336

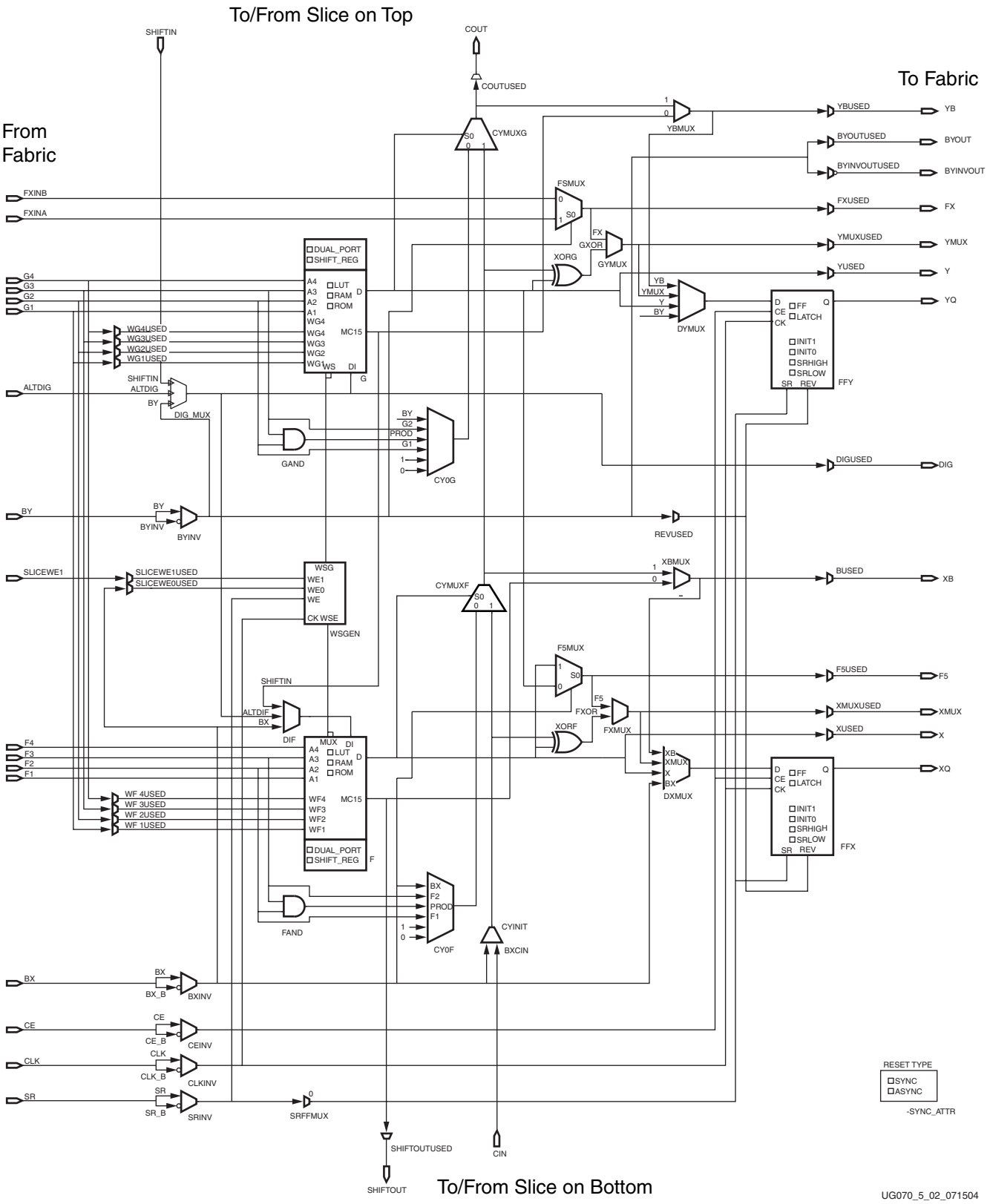


Figure 5-2: **Diagram of SLICEM**

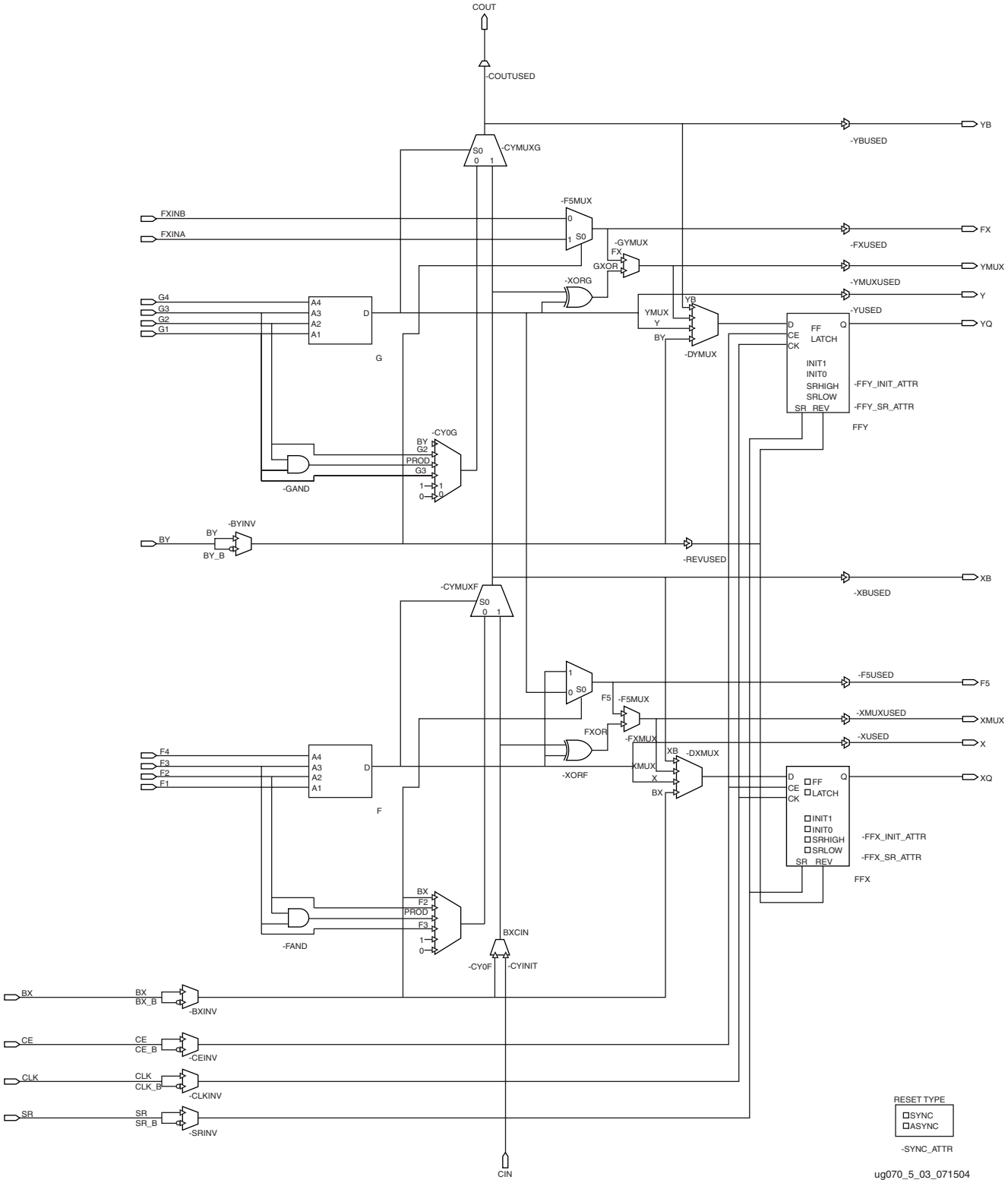


Figure 5-3: **Diagram of SLICEL**

Look-Up Table (LUT)

Virtex-4 FPGA function generators are implemented as 4-input look-up tables (LUTs). There are four independent inputs for each of the two function generators in a slice (F and G). The function generators are capable of implementing any arbitrarily defined four-input Boolean function. The propagation delay through a LUT is independent of the function implemented. Signals from the function generators can exit the slice (through the X or Y output), enter the XOR dedicated gate (see "Arithmetic Logic"), enter the select line of the carry-logic multiplexer (see "Fast Lookahead Carry Logic"), feed the D input of the storage element, or go to the MUXF5.

In addition to the basic LUTs, the Virtex-4 FPGA slices contain multiplexers (MUXF5 and MUXFX). These multiplexers are used to combine up to eight function generators to provide any function of five, six, seven, or eight inputs in a CLB. The MUXFX is either MUXF6, MUXF7, or MUXF8 according to the position of the slice in the CLB. The MUXFX can also be used to map any function of six, seven, or eight inputs and selected wide logic functions. Functions with up to nine inputs (MUXF5 multiplexer) can be implemented in one slice (see Figure 5-14, page 198). Wide function multiplexers can effectively combine LUTs within the same CLB or across different CLBs making logic functions with even more input variables.

Storage Elements

The storage elements in a Virtex-4 FPGA slice can be configured as either edge-triggered D-type flip-flops or level-sensitive latches. The D input can be driven directly by a LUT output via the DX or DY multiplexer, or by the slice inputs bypassing the function generators via the BX or BY input.

The control signals clock (CLK), clock enable (CE) and set/reset (SR) are common to both storage elements in one slice. All of the control signals have independent polarity. Any inverter placed on a control input is automatically absorbed. The clock-enable signal (CE) is active High by default. If left unconnected, the clock enable defaults to the active state.

In addition to clock (CLK) and clock enable (CE) signals, each slice has set and reset signals (SR and BY slice inputs). SR forces the storage element into the state specified by the attribute SRHIGH or SRLOW. SRHIGH forces a logic High when SR is asserted. SRLOW forces a logic Low. When SR is used, an optional second input (BY) forces the storage element into the opposite state via the REV pin. The reset condition is predominant over the set condition. (See Figure 5-4.) The truth tables for SR are described in"ILOGIC Resources" in Chapter 7.

The initial state after configuration or global initial state is defined by a separate INIT0 and INIT1 attribute. By default, setting the SRLOW attribute sets INIT0, and setting the SRHIGH attribute sets INIT1.

For each slice, set and reset can be synchronous or asynchronous. Virtex-4 devices can set INIT0 and INIT1 independent of SRHIGH and SRLOW.

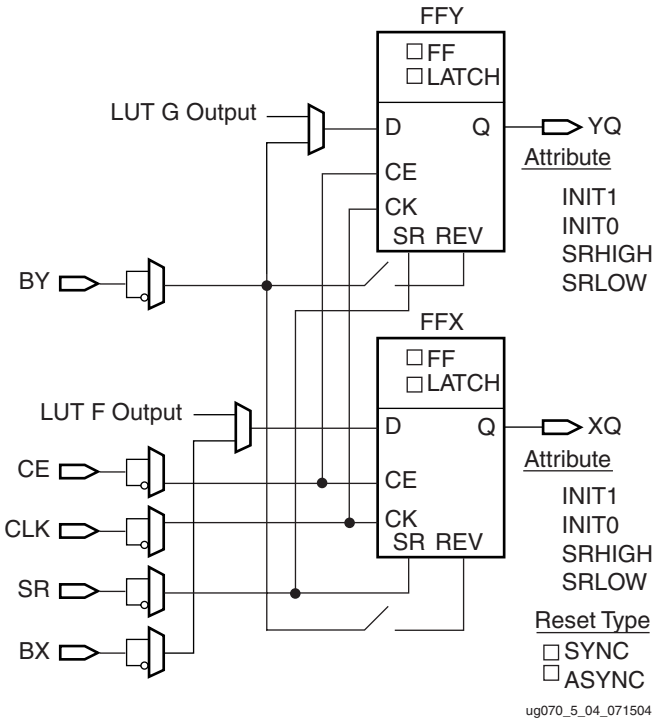


Figure 5-4: **Register/Latch Configuration in a Slice**

The configuration options for the set and reset functionality of a register or a latch are as follows:

- No set or reset
- Synchronous set
- Synchronous reset
- Synchronous set and reset
- Asynchronous set (preset)
- Asynchronous reset (clear)
- Asynchronous set and reset (preset and clear)

Distributed RAM and Memory (Available in SLICEM only)

Multiple left-hand LUTs in SLICEMs can be combined in various ways to store larger amounts of data.

The function generators (LUTs) in SLICEM can be implemented as a 16 x 1-bit synchronous RAM resource called a distributed RAM element. RAM elements are configurable within a CLB to implement the following:

- Single-Port 16 x 4-bit RAM
- Single-Port 32 x 2-bit RAM
- Single-Port 64 x 1-bit RAM
- Dual-Port 16 x 2-bit RAM

Distributed RAM modules are synchronous (write) resources. A synchronous read can be implemented with a storage element in the same slice. The distributed RAM and the

storage element share the same clock input. For a write operation, the Write Enable (WE) input, driven by the SR pin, must be set High.

Table 5-3 shows the number of LUTs (two per slice) occupied by each distributed RAM configuration.

Table 5-3: **Distributed RAM Configuration**

RAM	Number of LUTs
16 x 1S	1
16 x 1D	2
32 x 1S	2
64 x 1S	4

Notes:

1. S = single-port configuration; D = dual-port configuration

For single-port configurations, distributed RAM has a common address port for synchronous writes and asynchronous reads.

For dual-port configurations, distributed RAM has one port for synchronous writes and asynchronous reads and another port for asynchronous reads. The function generator (LUT) has separated read address inputs and write address inputs.

In single-port mode, read and write addresses share the same address bus. In dual-port mode, one function generator (R/W port) is connected with shared read and write addresses. The second function generator has the A inputs (Read) connected to the second read-only port address and the W inputs (Write) shared with the first read/write port address.

Figure 5-5, Figure 5-6, and Figure 5-7 illustrate various example distributed RAM configurations occupying one slice.

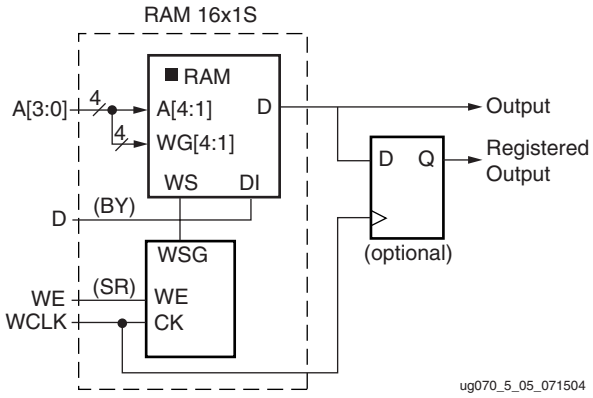


Figure 5-5: **Distributed RAM (RAM16x1S)**

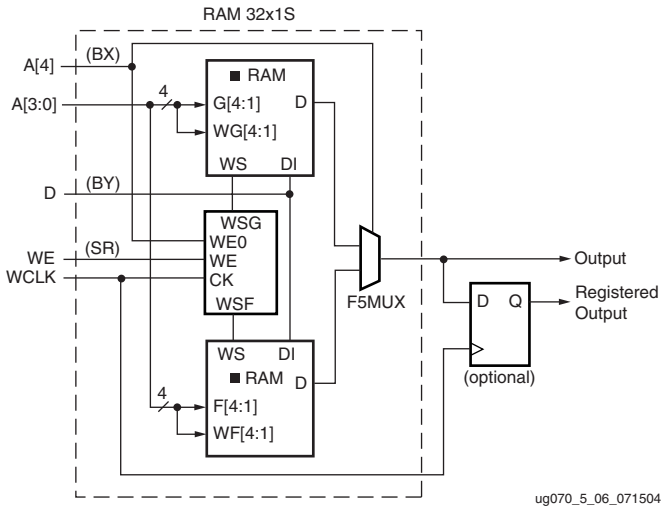


Figure 5-6: **Single-Port Distributed RAM (RAM32x1S)**

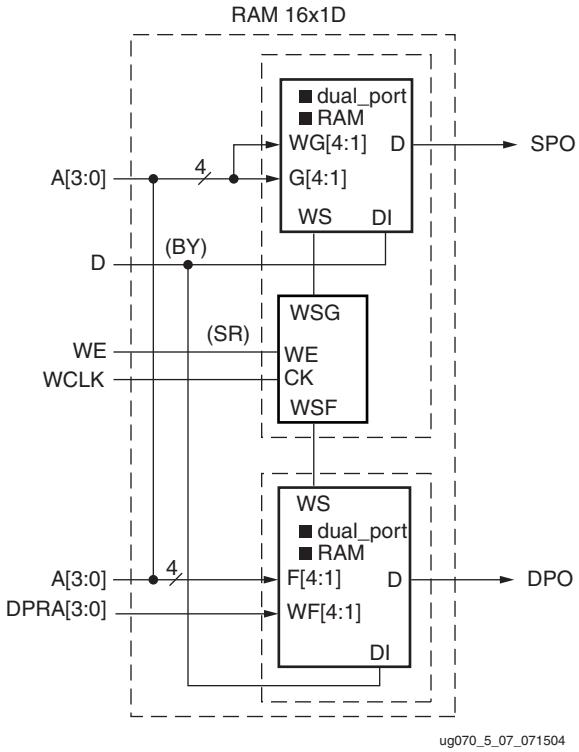


Figure 5-7: **Dual-Port Distributed RAM (RAM16x1D)**

If two dual-port 16 x 1-bit modules are built, the two RAM16X1D primitives can occupy two slices in a CLB, as long as they share the same clock and write enable, as illustrated in Figure 5-8.

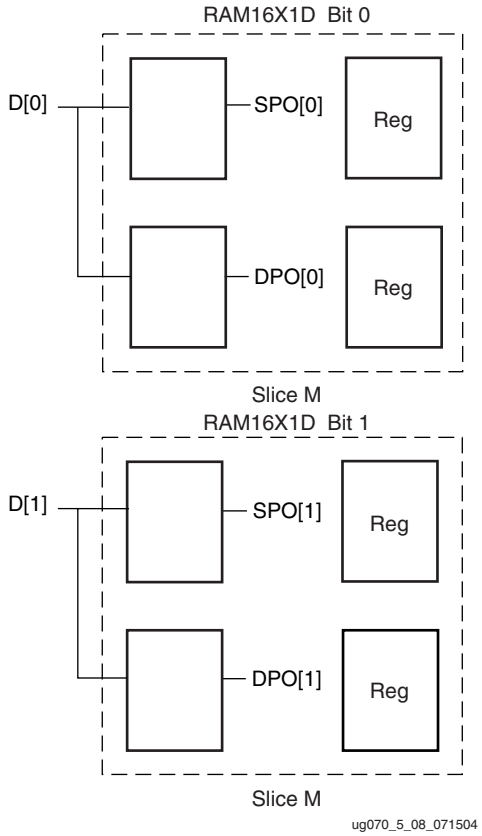


Figure 5-8: **Two RAM16X1D Placement**

The RAM64X1S primitive occupies two slices. The RAM64X1S read path is built on the MUXF5 and MUXF6 multiplexers.

Read Only Memory (ROM)

Each function generator in SLICEM and SLICEL can implement a 16 x 1-bit ROM. Four configurations are available: ROM16x1, ROM32x1, ROM64x1, and ROM128x1. The ROM elements are cascadable to implement wider and/or deeper ROM. ROM contents are loaded at device configuration. Table 5-4 shows the number of LUTs occupied by each configuration.

Table 5-4: **ROM Configuration**

ROM	Number of LUTs
16 x 1	1
32 x 1	2
64 x 1	4
128 x 1	8
256 x 1	16 (2 CLBs)

Shift Registers (Available in SLICEM only)

A SLICEM function generator can also be configured as a 16-bit shift register without using the flip-flops available in a slice. Used in this way, each LUT can delay serial data anywhere from one to 16 clock cycles. The SHIFTIN and SHIFTOUT lines cascade LUTs to form larger shift registers. The four left-hand LUTs (in SLICEM) of a single CLB are thus cascaded to produce delays up to 64 clock cycles. It is also possible to combine shift registers across more than one CLB. The resulting programmable delays can be used to balance the timing of data pipelines.

Applications requiring delay or latency compensation use these shift registers to develop efficient designs. Shift registers are also useful in synchronous FIFO and content-addressable memory (CAM) designs. To quickly generate a Virtex-4 FPGA shift register without using flip-flops (i.e., using the SRL16 element(s)), use the CORE Generator™ tool RAM-based shift-register module.

The write operation is synchronous with a clock input (CLK) and an optional clock enable, as shown in Figure 5-9. A dynamic read access is performed through the 4-bit address bus, A[3:0]. The configurable 16-bit shift register cannot be set or reset. The read is asynchronous; however, a storage element or flip-flop is available to implement a synchronous read. By placing this flip-flop, the shift register performance is improved by decreasing the delay into the clock-to-out value of the flip-flop. However, an additional clock latency is added. Any of the 16 bits can be read out asynchronously by varying the LUT address. This is useful in making smaller shift registers (less than 16 bits.) For example, when building an 8-bit shift register, simply set the addresses to the 8th bit.

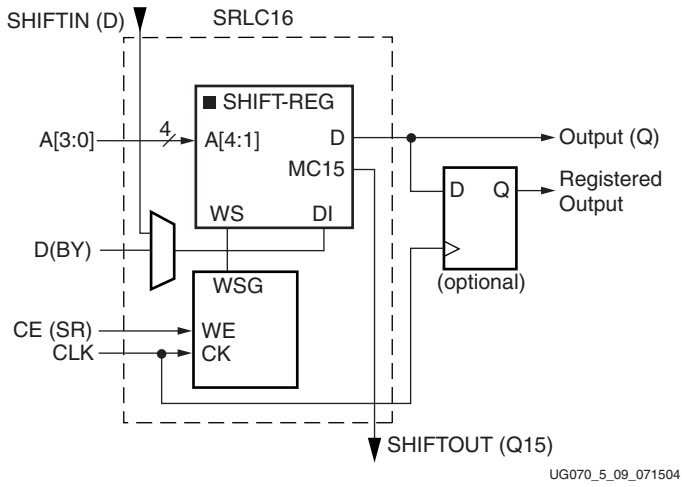


Figure 5-9: **Shift Register Configurations**

Figure 5-10 is an equivalent representation of the shift register.

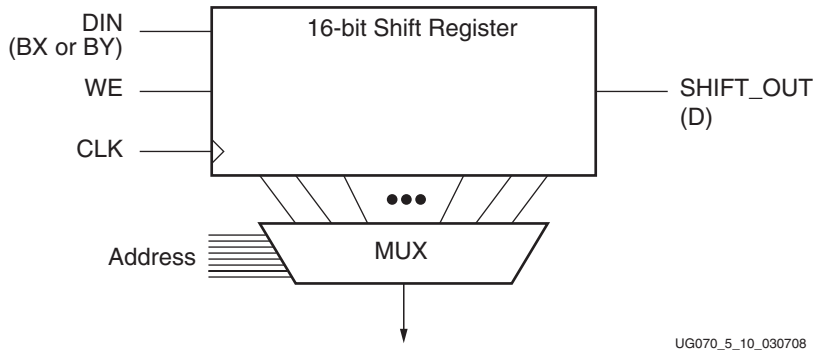


Figure 5-10: **Representation of a Shift Register**

An additional dedicated connection between shift registers allows connecting the last bit of one shift register to the first bit of the next, without using the LUT D-output (see Figure 5-11). Longer shift registers can be built with dynamic access to any bit in the chain. The shift register chaining and the MUXF5, and MUXF6 multiplexers allow up to a 64-bit shift register with addressable access to be implemented in one CLB.

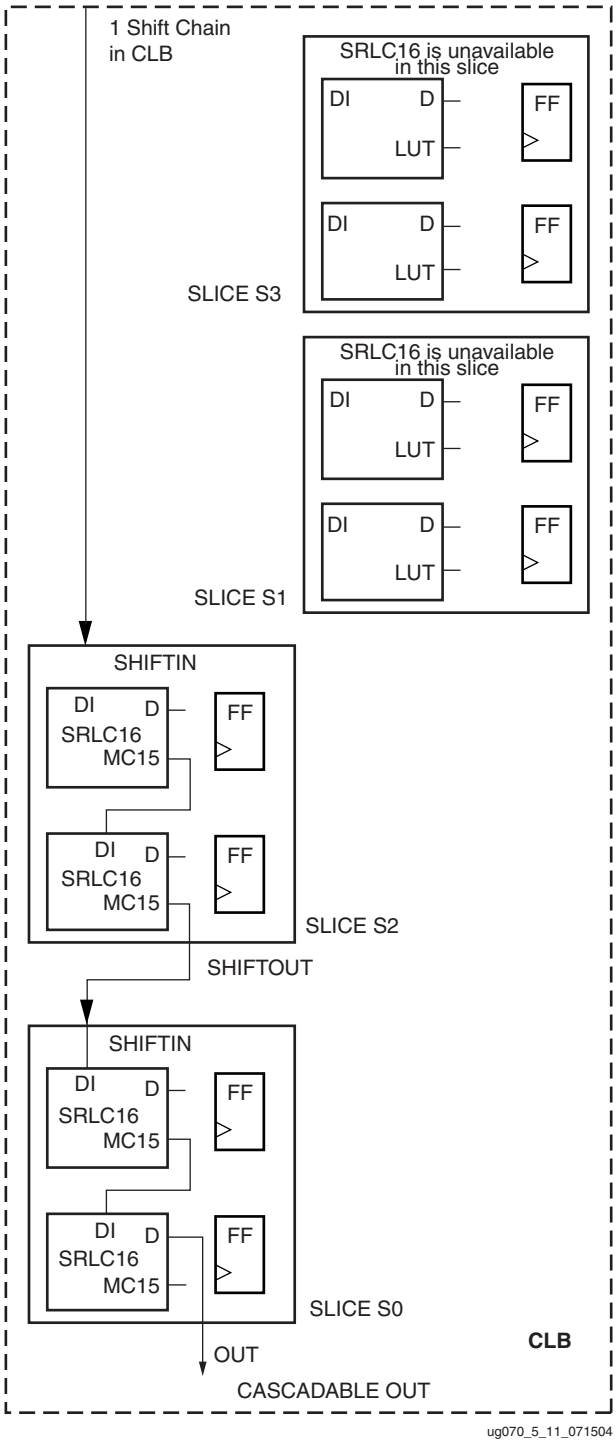


Figure 5-11: **Cascadable Shift Register**

The block diagrams of the shift register (SRL16E) and the cascadable shift register (SRLC16E) are illustrated in Figure 5-12. The pin descriptions of SRL16E and SRLC16E are located in the "SRL Primitives and Submodules" section.

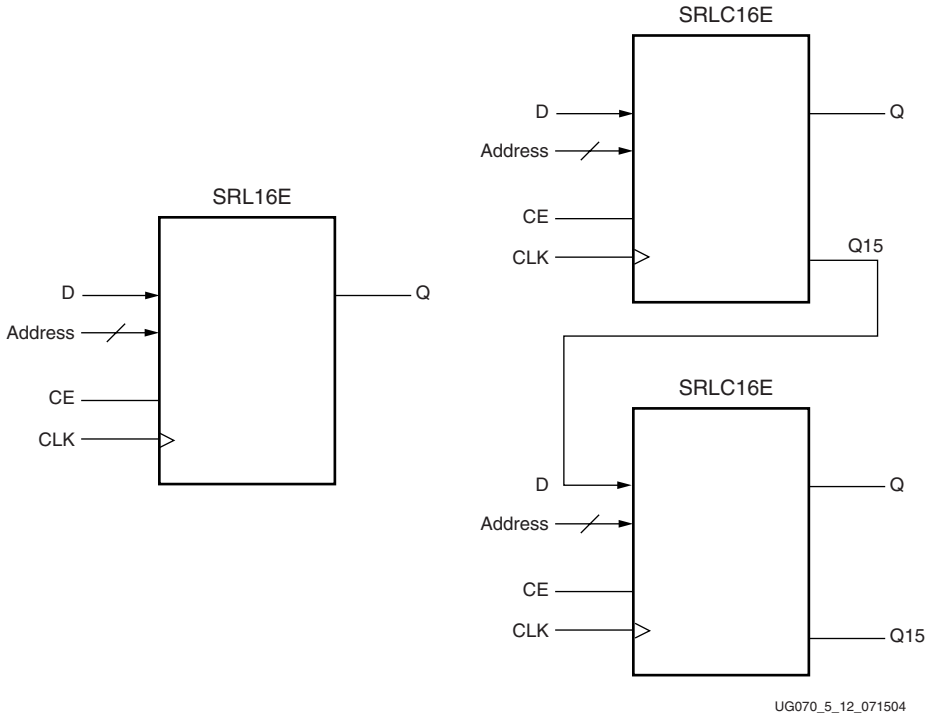


Figure 5-12: **Simplified Shift Register and Cascadable Shift Register**

Shift Register Data Flow

Shift Operation

The shift operation is a single clock-edge operation, with an active High clock enable feature. When enable is High, the input (D) is loaded into the first bit of the shift register, and each bit is shifted to the next highest bit position. In a cascadable shift register configuration (such as SRLC16), the last bit is shifted out on the Q15 output.

The bit selected by the 4-bit address appears on the Q output.

Dynamic Read Operation

The Q output is determined by the 4-bit address. Each time a new address is applied to the 4-input address pins, the new bit position value is available on the Q output after the time delay to access the LUT. This operation is asynchronous and independent of the clock and clock enable signals.

Static Read Operation

If the 4-bit address is fixed, the Q output always uses the same bit position. This mode implements any shift-register length from one to 16 bits in one LUT. Shift register length is (N+1) where N is the input address.

The Q output changes synchronously with each shift operation. The previous bit is shifted to the next position and appears on the Q output.

Shift Register Summary

- A shift operation requires one clock edge.
- Dynamic-length read operations are asynchronous (Q output).
- Static-length read operations are synchronous (Q output).
- The data input has a setup-to-clock timing specification.
- In a cascadable configuration, the Q15 output always contains the last bit value.
- The Q15 output changes synchronously after each shift operation.

Multiplexers

Virtex-4 FPGA function generators and associated multiplexers can implement the following:

- 4:1 multiplexer in one slice
- 8:1 multiplexer in two slices
- 16:1 multiplexer in one CLB element (4 slices)
- 32:1 multiplexer in two CLB elements (8 slices - 2 adjacent CLBs)

Wide input multiplexers are implemented in one level of logic (or LUT) and by dedicated MUXFX. These multiplexers are fully combinatorial.

Each Virtex-4 FPGA slice has one MUXF5 multiplexer and one MUXFX multiplexer. The MUXFX multiplexer implements the MUXF6, MUXF7, or MUXF8, according to the slice position in the CLB, as shown in Figure 5-13. Each CLB element has two MUXF6 multiplexers, one MUXF7 multiplexer and one MUXF8 multiplexer. MUXFX are designed to allow LUT combinations of up to 16 LUTs in two adjacent CLBs. Any LUT can implement a 2:1 multiplexer. Examples of multiplexers are shown in the Designing Large Multiplexers section.

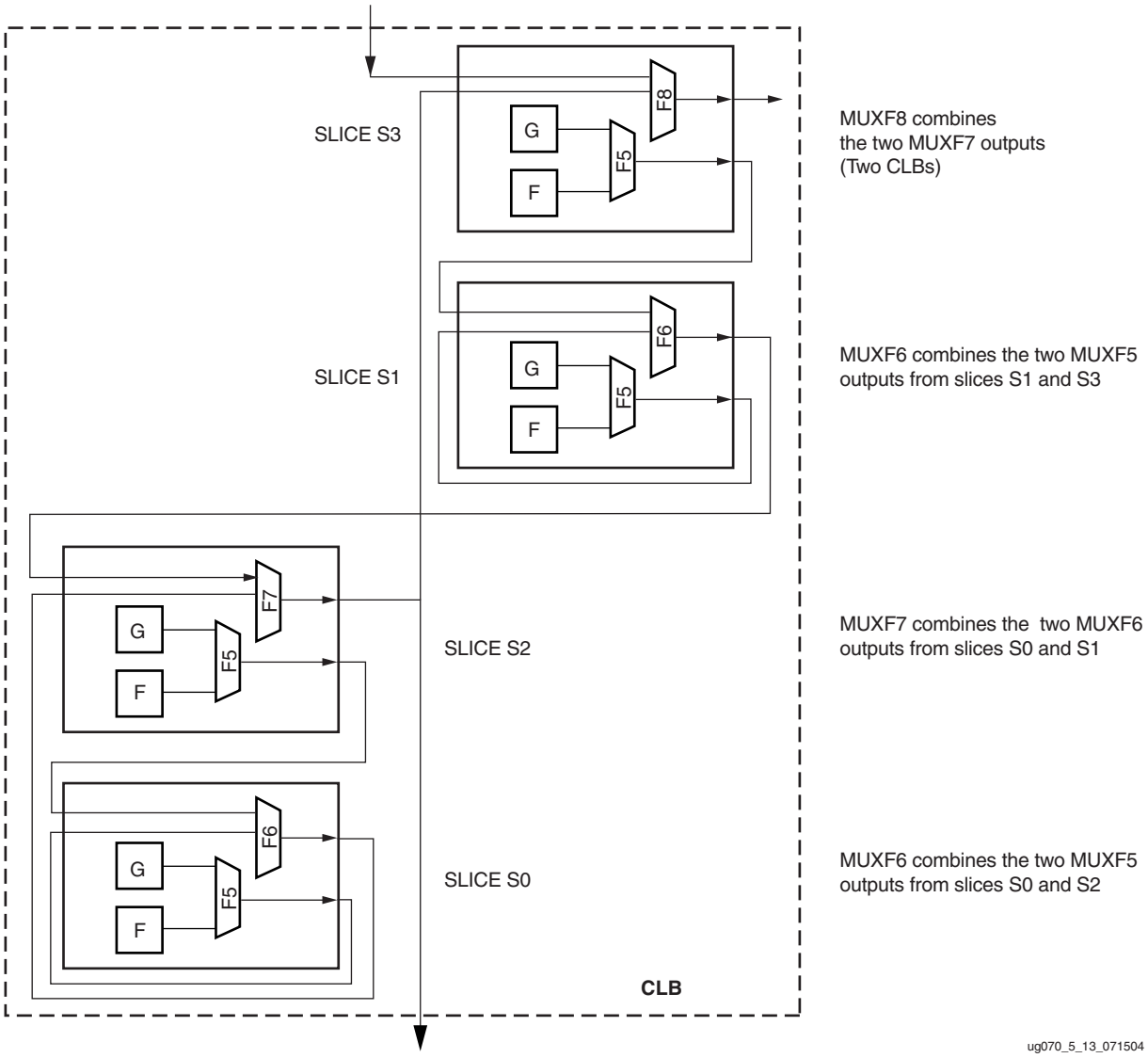


Figure 5-13: **MUXF5 and MUXFX Multiplexers**

Designing Large Multiplexers

4:1 Multiplexer

Each Virtex-4 FPGA slice has a MUXF5 to combine the outputs of the two LUTs and an extra MUXFX. Figure 5-14 illustrates a valid combinatorial function with up to nine inputs (or a 4:1 MUX) in one slice.

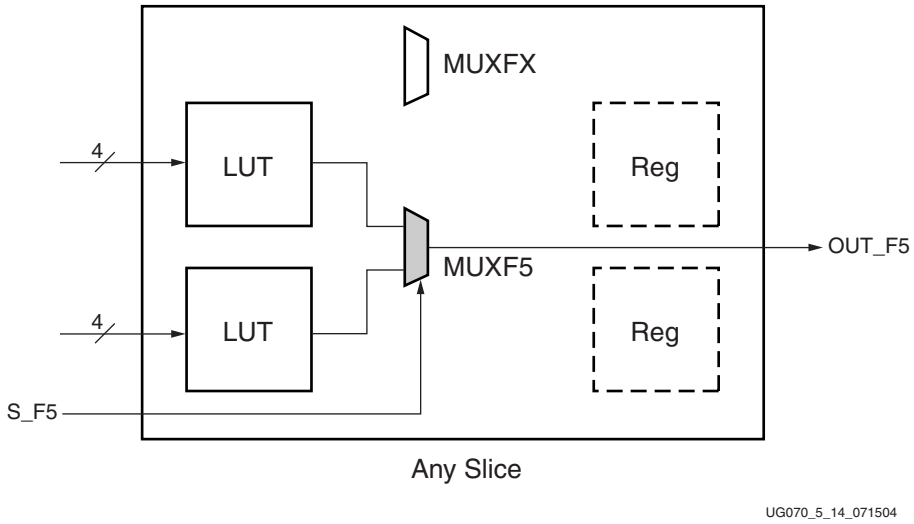


Figure 5-14: **LUTs and MUXF5 in a Slice**

8:1 Multiplexer

Slice S0 and S1 have a MUXF6. MUXF6 is designed to combine the outputs of two MUXF5 resources. Figure 5-15 illustrates a combinatorial function up to 18 inputs (or an 8:1 MUX) in the slices S0 and S2, or in the slices S1 and S3.

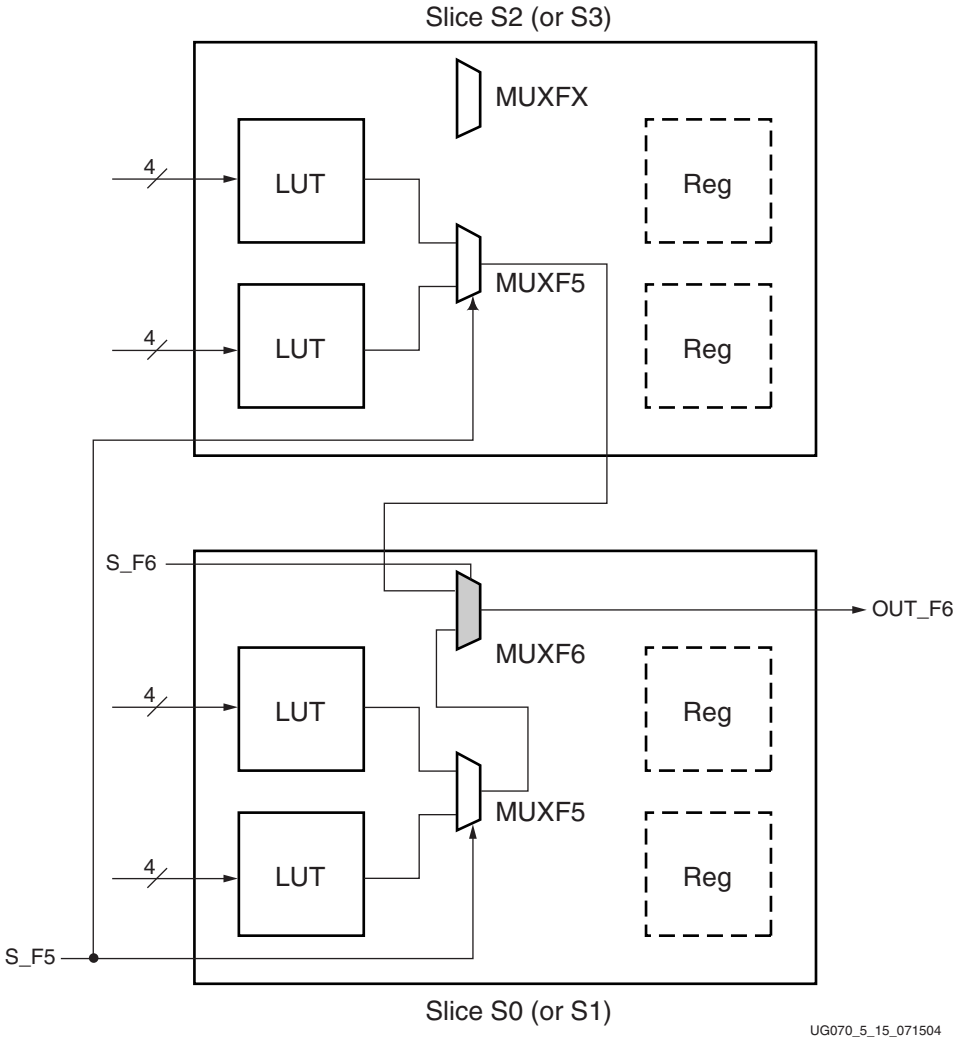


Figure 5-15: **LUTs and (MUXF5 and MUXF6) in Two Slices**

16:1 Multiplexer

Slice S2 has a MUXF7. MUXF7 is designed to combine the outputs of two MUXF6. Figure 5-16 illustrates a combinatorial function up to 35 inputs (or a 16:1 MUX) in a Virtex-4 FPGA CLB.

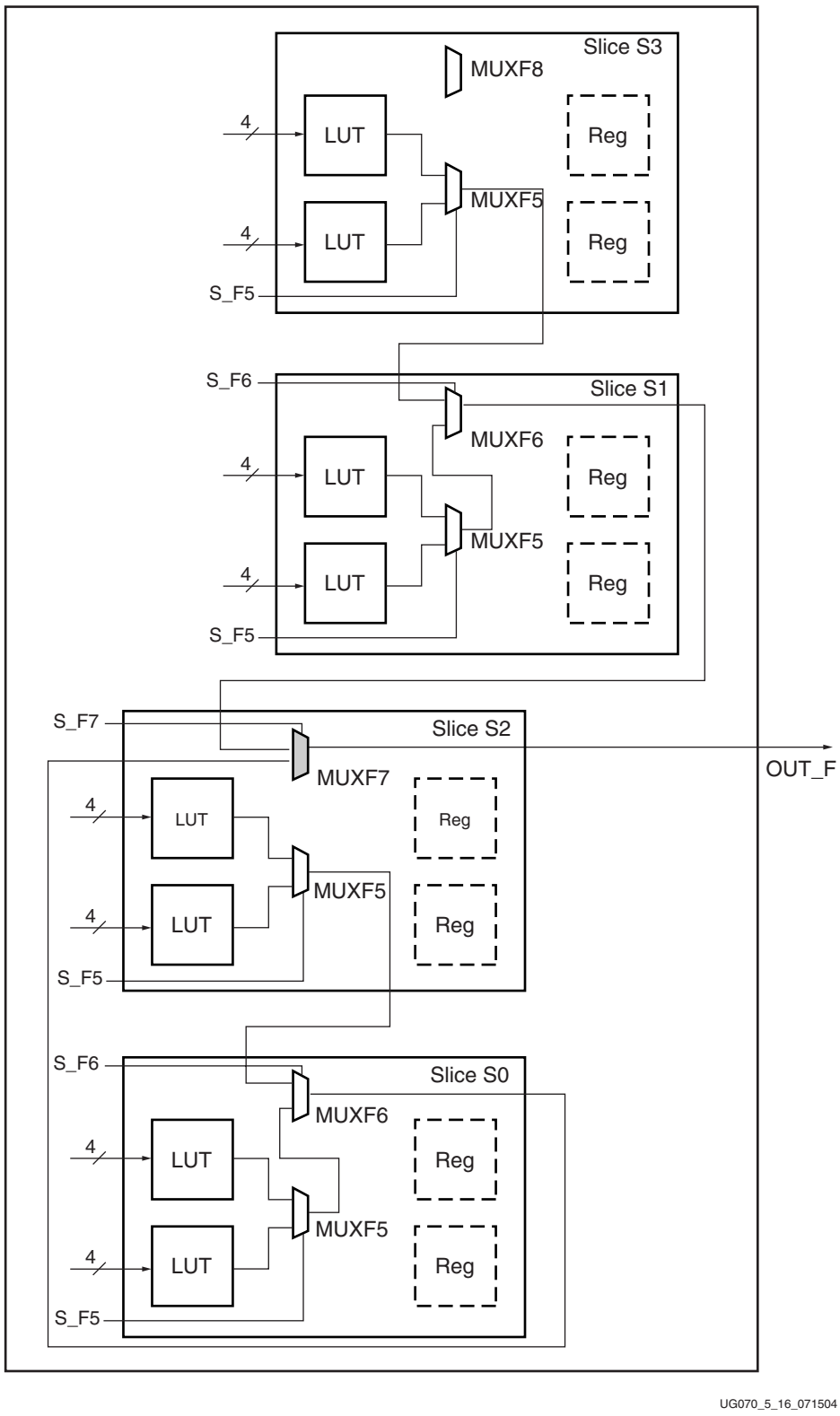


Figure 5-16: **LUTs and (MUXF5, MUXF6, and MUXF7) in One CLB**

32:1 Multiplexer

Slice S3 of each CLB has a MUXF8. Combinatorial functions of up to 68 inputs (or a 32:1 MUX) fit in two CLBs as shown in Figure 5-17. The outputs of two MUXF7 are combined through dedicated routing resources between two adjacent CLBs in a column.

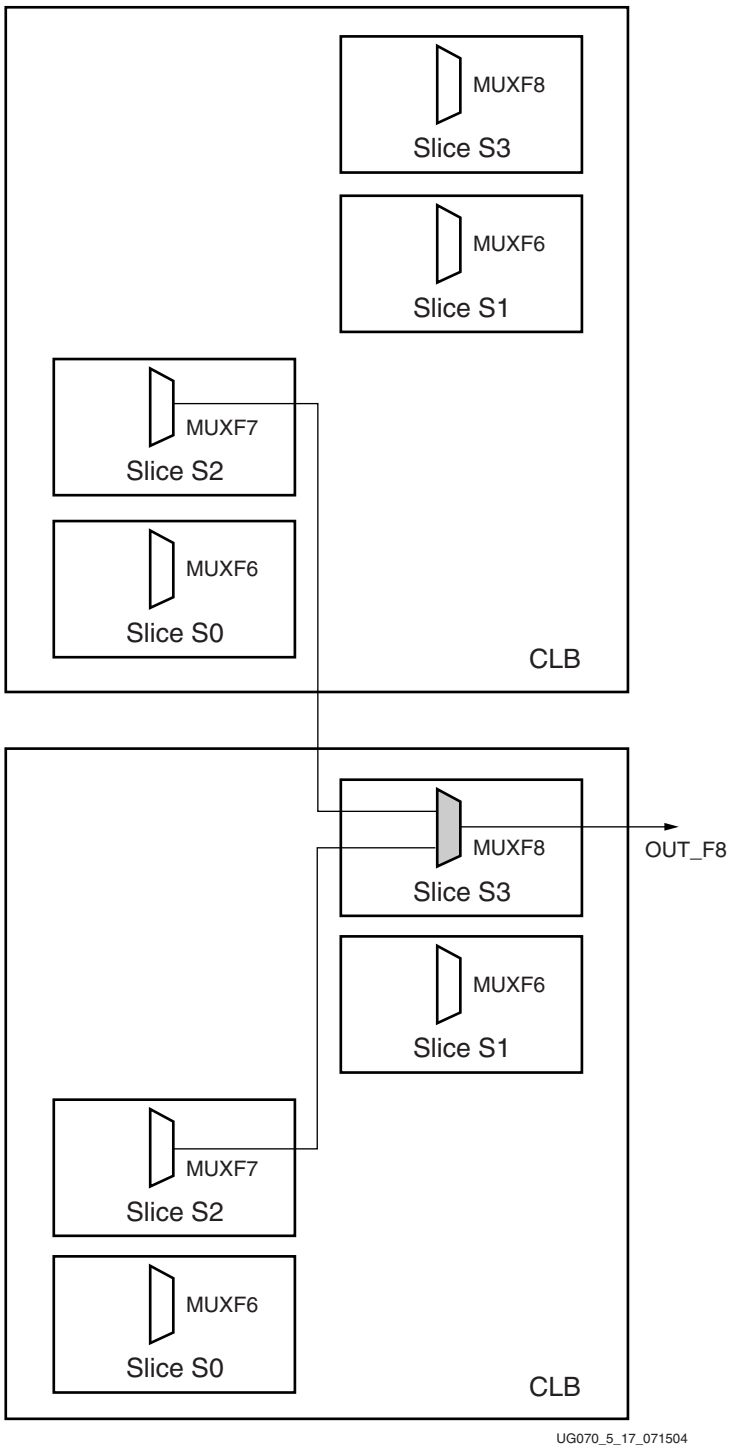


Figure 5-17: **MUXF8 Combining Two Adjacent CLBs**

Wide-Input Multiplexer Summary

Each LUT can implement a 2:1 multiplexer. In each slice, the MUXF5 and two LUTs can implement a 4:1 multiplexer. The MUXF6 and two slices can implement a 8:1 multiplexer. The MUXF7 and the four slices of any CLB can implement a 16:1, and the MUXF8 and two CLBs can implement a 32:1 multiplexer. Figure 5-18 summarizes the implementation of a wide-input multiplexer. The section "Multiplexer Verilog/VHDL Examples" has code for the wide-input multiplexers.

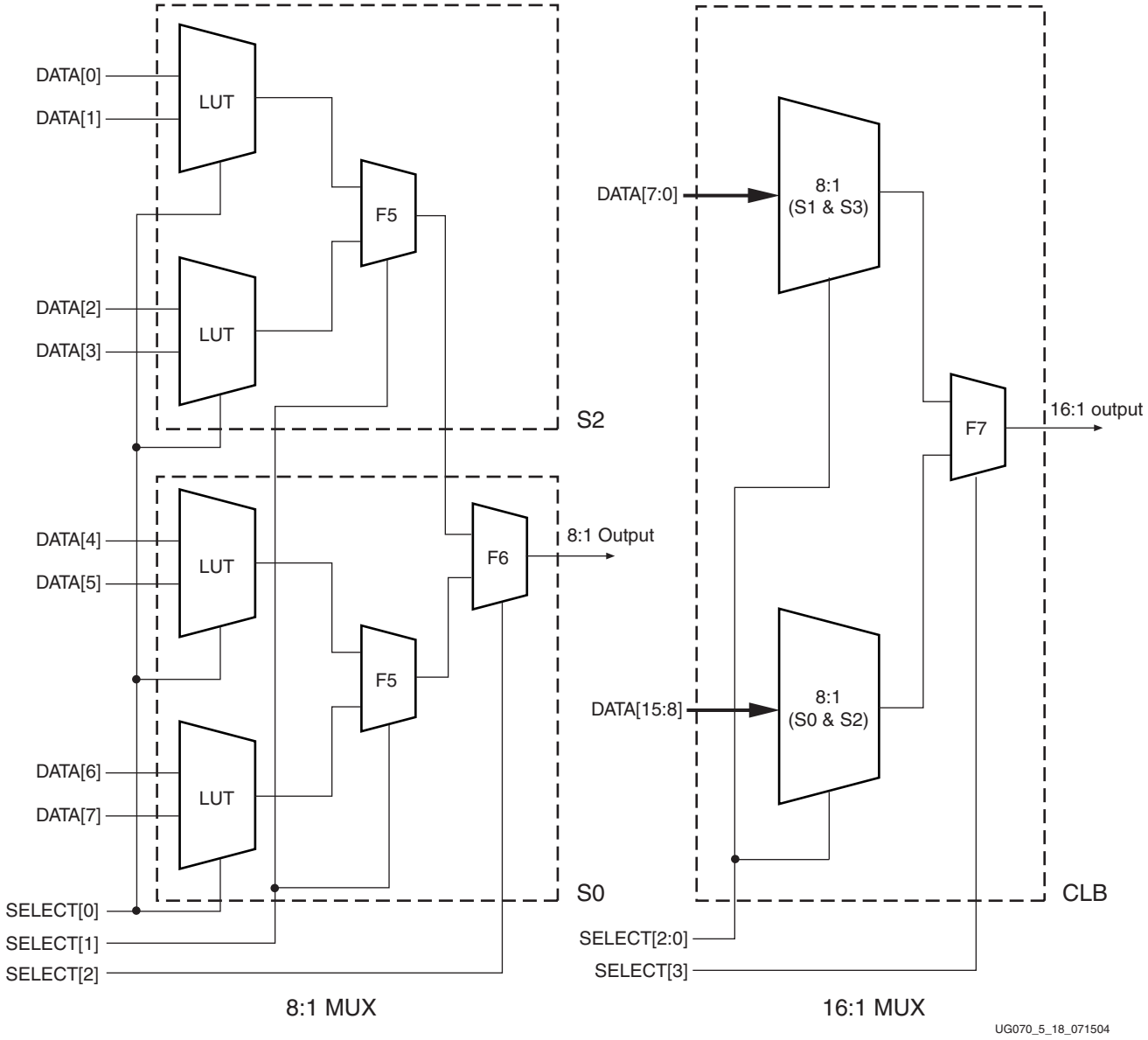


Figure 5-18: **8:1 and 16:1 Multiplexers**

Fast Lookahead Carry Logic

Dedicated carry logic provides fast arithmetic addition and subtraction. The Virtex-4 FPGA CLB has two separate carry chains, as shown in the Figure 5-19.

The height of the carry chains is two bits per slice. The carry chain in the Virtex-4 device is running upward. The dedicated carry path and carry multiplexer (MUXCY) can also be used to cascade function generators for implementing wide logic functions.

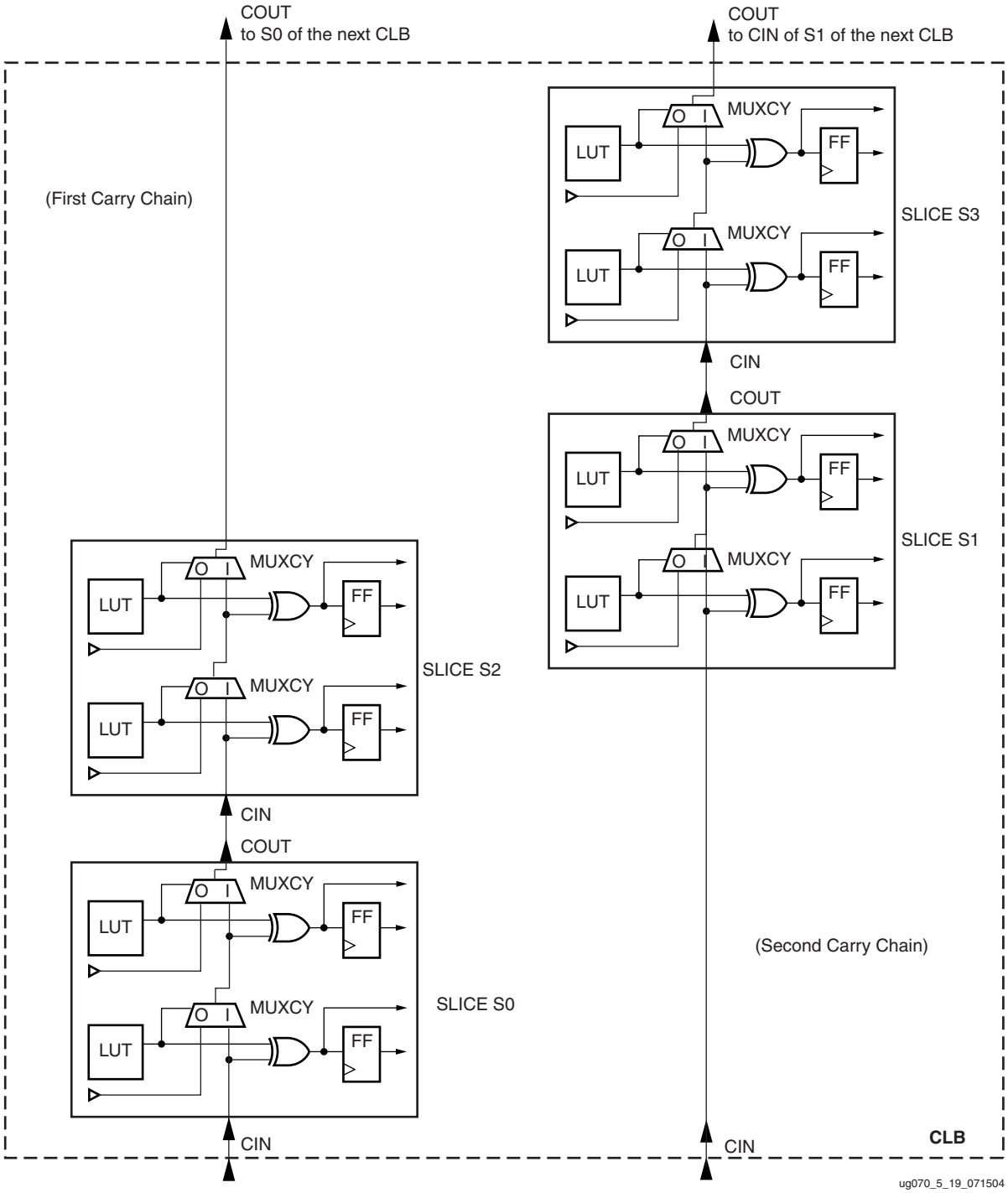


Figure 5-19: **Fast Carry Logic Path**

Arithmetic Logic

The arithmetic logic includes an XOR gate that allows a 2-bit full adder to be implemented within a slice. In addition, a dedicated AND (FAND or GAND) gate (shown in Figure 5-2) improves the efficiency of multiplier implementation.

CLB / Slice Timing Models

Due to the large size and complexity of Virtex-4 FPGAs, understanding the timing associated with the various paths and functional elements has become a difficult and important task. Although it is not necessary to understand the various timing parameters to implement most designs using Xilinx software, a thorough timing model can assist advanced users in analyzing critical paths or planning speed-sensitive designs.

Three timing model sections are described.

- Functional element diagram - basic architectural schematic illustrating pins and connections.
- Timing parameters - definitions of *Virtex-4 Data Sheet* timing parameters.
- Timing Diagram - illustrates functional element timing parameters relative to each other.

Use the models in this chapter in conjunction with both the Xilinx Timing Analyzer software (TRCE) and the section on switching characteristics in the *Virtex-4 Data Sheet*. All pin names, parameter names, and paths are consistent with the post-route timing and pre-route static timing reports. Most of the timing parameters found in the section on switching characteristics are described in this chapter.

All timing parameters reported in the *Virtex-4 Data Sheet* are associated with slices and configurable logic blocks (CLBs). The following sections correspond to specific switching characteristics sections in the *Virtex-4 Data Sheet*:

- "General Slice Timing Model and Parameters" (CLB Switching Characteristics)
- "Slice Distributed RAM Timing Model and Parameters (Available in SLICEM only)" (CLB Distributed RAM Switching Characteristics)
- "Slice SRL Timing Model and Parameters (Available in SLICEM only)" (CLB SRL Switching Characteristics)
- "Slice Carry-Chain Timing Model and Parameters" (CLB Application Switching Characteristics)

General Slice Timing Model and Parameters

A simplified Virtex-4 FPGA slice is shown in Figure 5-20. Some elements of the Virtex-4 FPGA slice are omitted for clarity. Only the elements relevant to the timing paths described in this section are shown.

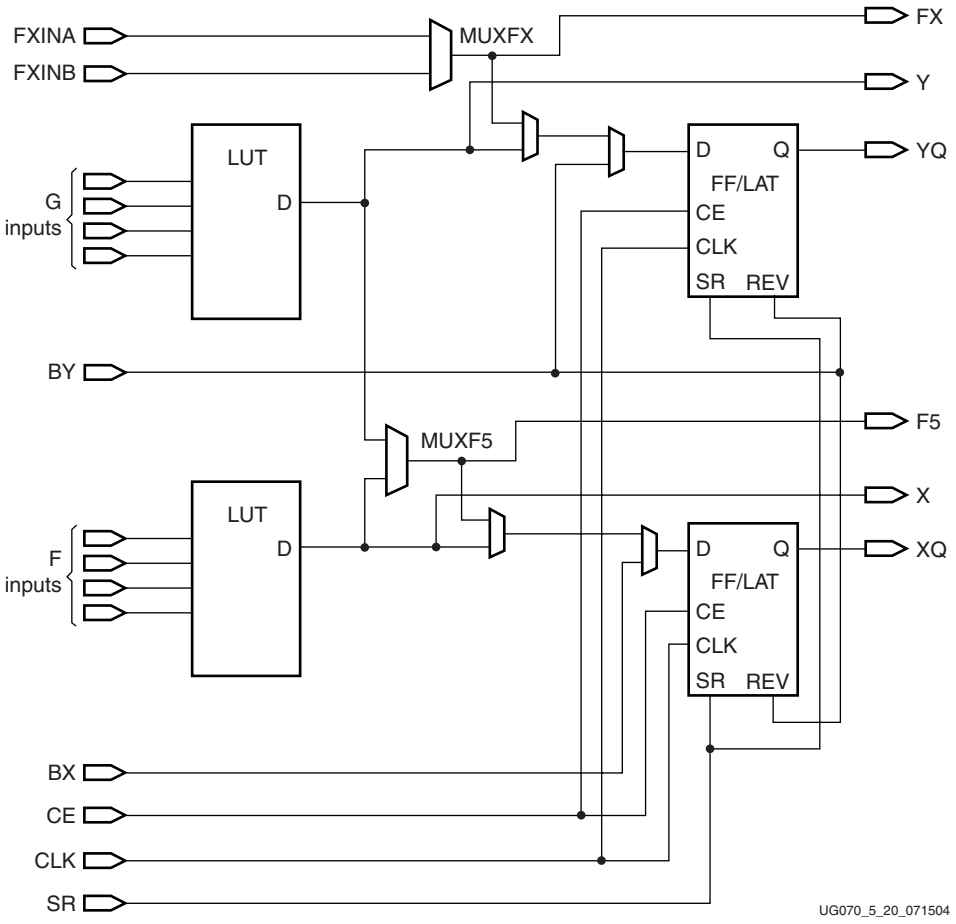


Figure 5-20: **Simplified Virtex-4 FPGA General SLICEL/SLICEM**

Timing Parameters

Table 5-5 shows the general slice timing parameters for a majority of the paths in Figure 5-20.

Table 5-5: **General Slice Timing Parameters**

Parameter	Function	Description
Combinatorial Delays		
T_{ILO}	F/G inputs to X/Y outputs	Propagation delay from the F/G inputs of the slice, through the look-up tables (LUTs), to the X/Y outputs of the slice.
T_{IF5}	F/G inputs to F5 output	Propagation delay from the F/G inputs of the slice, through the LUTs and MUXF5 to the F5 output of the slice.
T_{IF5X}	F/G inputs to XMUX output	Propagation delay from the F/G inputs of the slice, through the LUTs and MUXF5 to the XMUX output of the slice.

Table 5-5: **General Slice Timing Parameters *(Continued)***

Parameter	Function	Description
T_{IF6Y}	F_{XINA}/F_{XINB} inputs to YMUX output	Propagation delay from the F_{XINA}/F_{XINB} inputs, through F6MUX to the YMUX output of the slice.
T_{INAFX}/T_{INBFX}	F_{XINA}/F_{XINB} inputs to FX output	Propagation delay from the F_{XINA}/F_{XINB} inputs, through F6MUX to the FX output of the slice.
Sequential Delays		
T_{CKO}	FF Clock (CLK) to XQ/YQ outputs	Time after the clock that data is stable at the XQ/YQ outputs of the slice sequential elements (configured as a flip-flop).
T_{CKLO}	Latch Clock (CLK) to XQ/YQ outputs	Time after the clock that data is stable at the XQ/YQ outputs of the slice sequential elements (configured as a latch).
Setup and Hold for Slice Sequential Elements		
T_{xxCK} = Setup time (before clock edge) T_{CKxx} = Hold time (after clock edge)		
T_{DICK}/T_{CKDI}	BX/BY Inputs	Time before Clock (CLK) that data from the BX or BY inputs of the slice must be stable at the D-input of the slice sequential elements (configured as a flip-flop).
T_{FXCK}/T_{CKFX}	F_{XINA}/F_{XINB} Input	Time before Clock (CLK) that data from the F_{XINA} or F_{XINB} inputs of the slice must be stable at the D-input of the slice sequential elements (configured as a flip-flop).
T_{CECK}/T_{CKCE}	CE input	Time before Clock (CLK) that the CE (Clock Enable) input of the slice must be stable at the CE-input of the slice sequential elements (configured as a flip-flop).
T_{SRCK}/T_{CKSR}	SR/BY inputs	Time before Clock (CLK) that the SR (Set/Reset) and the BY (Rev) inputs of the slice must be stable at the SR/Rev-inputs of the slice sequential elements (configured as a flip-flop). Synchronous set/reset only.
Set/Reset		
TRPW		Minimum Pulse Width for the SR (Set/Reset) and BY (Rev) pins.
TRQ		Propagation delay for an asynchronous Set/Reset of the slice sequential elements. From SR/BY inputs to XQ/YQ outputs.
FTOG		Toggle Frequency - Maximum Frequency that a CLB flip-flop can be clocked: $1/(T_{CH}+T_{CL})$.

Timing Characteristics

Figure 5-21 illustrates the general timing characteristics of a Virtex-4 FPGA slice.

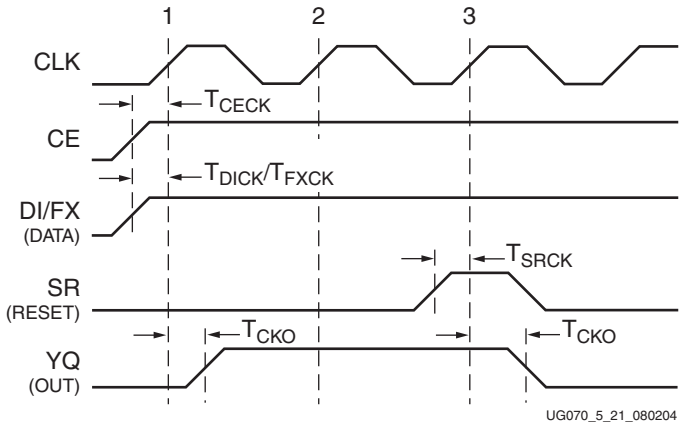


Figure 5-21: **General Slice Timing Characteristics**

- At time T_{CECK} before clock event (1), the clock-enable signal becomes valid-High at the CE input of the slice register.
- At time T_{DICK} or T_{FXCK} before clock event (1), data from either BX, BY, FXINA or FXINB inputs become valid-High at the D input of the slice register and is reflected on either the XQ or YQ pin at time T_{CKO} after clock event (1).
- At time T_{SRCK} before clock event (3), the SR signal (configured as synchronous reset in this case) becomes valid-High, resetting the slice register. This is reflected on the XQ or YQ pin at time T_{CKO} after clock event (3).

Slice Distributed RAM Timing Model and Parameters (Available in SLICEM only)

Figure 5-22 illustrates the details of distributed RAM implemented in a Virtex-4 FPGA slice. Some elements of the Virtex-4 FPGA slice are omitted for clarity. Only the elements relevant to the timing paths described in this section are shown.

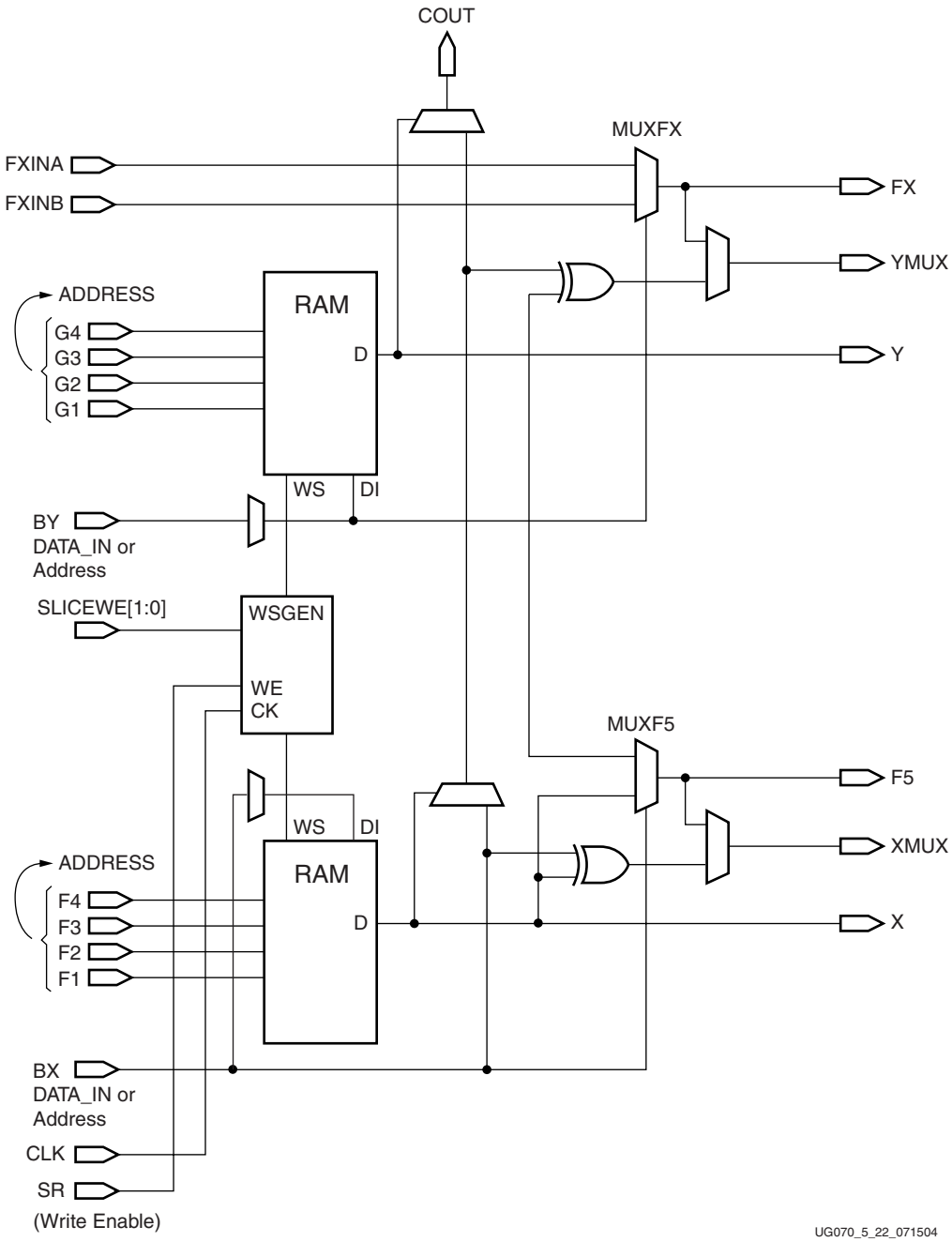


Figure 5-22: **Simplified Virtex-4 FPGA SLICEM Distributed RAM**

Distributed RAM Timing Parameters

Table 5-6 shows the timing parameters for the distributed RAM in SLICEM for a majority of the paths in Figure 5-22.

Table 5-6: **Distributed RAM Timing Parameters**

Parameter	Function	Description
Sequential Delays for Slice LUT Configured as RAM (Distributed RAM)		
T_{SHCKO}	CLK to X	Time after the Clock (CLK) of a Write operation that the data written to the distributed RAM is stable on the X output of the slice.
$T_{SHCKOF5}$	CLK to F5 output (WE active)	Time after the Clock (CLK) of a Write operation that the data written to the distributed RAM is stable on the F5 output of the slice.
Setup and Hold for Slice LUT Configured as RAM (Distributed RAM)		
T_{xS} = Setup time (before clock edge) T_{xH} = Hold time (after clock edge)		The following descriptions are for setup times only.
T_{DS}/T_{DH}	BX/BY configured as data input (DI)	Time before the clock that data must be stable at the BX/BY input of the slice.
T_{AS}/T_{AH}	F/G Address inputs	Time before the clock that address signals must be stable at the F/G inputs of the slice LUT (configured as RAM).
T_{WS}/T_{WH}	WE input (SR)	Time before the clock that the write enable signal must be stable at the WE input of the slice LUT (configured as RAM).
Clock CLK		
T_{WC}		Minimum clock period to meet address write cycle time.

Distributed RAM Timing Characteristics

The timing characteristics of a 16-bit distributed RAM implemented in a Virtex-4 FPGA slice (LUT configured as RAM) are shown in Figure 5-23.

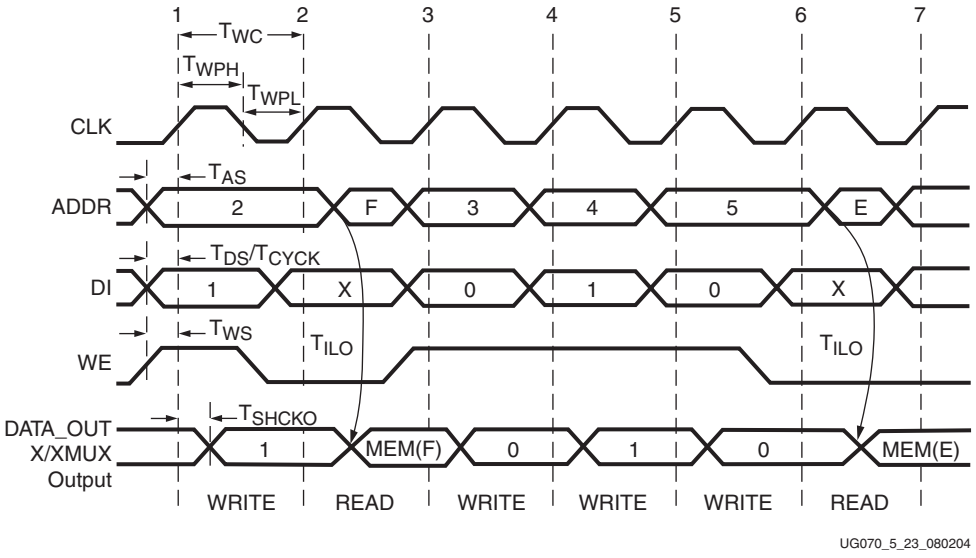


Figure 5-23: **Slice Distributed RAM Timing Characteristics**

Clock Event 1: Write Operation

During a Write operation, the contents of the memory at the address on the ADDR inputs are changed. The data written to this memory location is reflected on the X/Y outputs synchronously.

- At time T_{WS} before clock event 1, the write-enable signal (WE) becomes valid-High, enabling the RAM for the following Write operation.
- At time T_{AS} before clock event 1, the address (2) becomes valid at the F/G inputs of the RAM.
- At time T_{DS} or T_{CYCK} before clock event 1, the DATA becomes valid (1) at the DI input of the RAM and is reflected on the X/XMUX output at time T_{SHCKO} after clock event 1.

This is also applicable to the XMUX, YMUX, XB, YB, C_{OUT}, and F5 outputs at time T_{WOSCO}, T_{WOSX}, T_{WOSXB}, T_{WOSYB}, and $T_{SHCKOF5}$ after clock event 1.

Clock Event 2: Read Operation

All Read operations are asynchronous in distributed RAM. As long as WE is Low, the address bus can be asserted at any time. The contents of the RAM on the address bus are reflected on the X/Y outputs after a delay of length T_{ILO} (propagation delay through a LUT). The address (F) is asserted *after* clock event 2, and the contents of the RAM at address (F) are reflected on the output after a delay of length T_{ILO}.

Slice SRL Timing Model and Parameters (Available in SLICEM only)

Figure 5-24 illustrates shift register implementation in a Virtex-4 FPGA slice. Some elements of the Virtex-4 FPGA slice have been omitted for clarity. Only the elements relevant to the timing paths described in this section are shown.

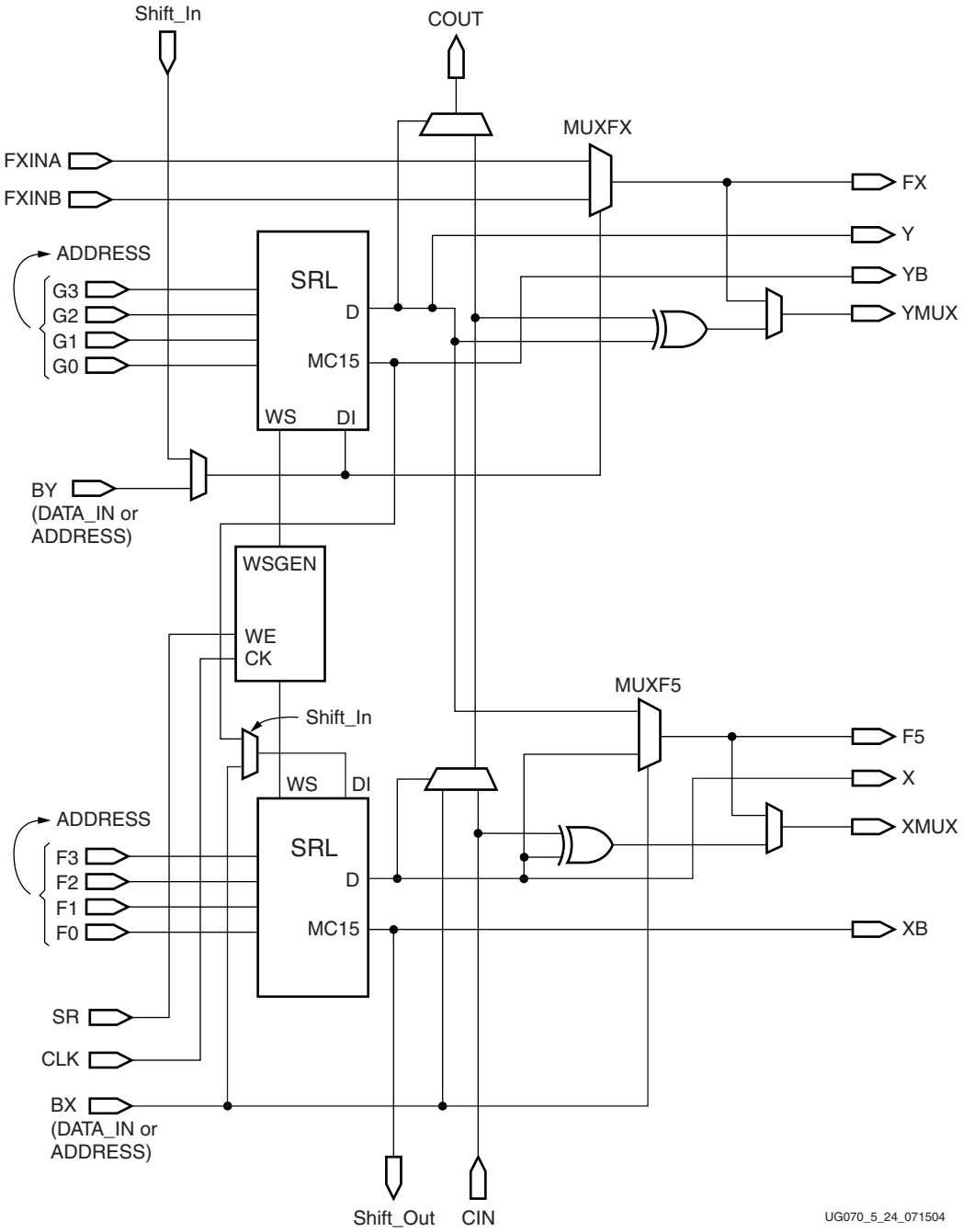


Figure 5-24: **Simplified Virtex-4 FPGA Slice SRL**

Slice SRL Timing Parameters

Table 5-7 shows the SLICEM SRL timing parameters for a majority of the paths in Figure 5-24.

Table 5-7: **Slice SRL Timing Parameters**

Parameter	Function	Description
Sequential Delays for Slice LUT Configured as SRL (Select Shift Register)		
T_{REG}	CLK to X/Y outputs	Time after the Clock (CLK) of a Write operation that the data written to the SRL is stable on the X/Y outputs of the slice.
T_{CKSH}	CLK to Shift_out	Time after the Clock (CLK) of a Write operation that the data written to the SRL is stable on the Shift_out or XB/YB outputs of the slice.
T_{REGF5}	CLK to F5 output	Time after the Clock (CLK) of a Write operation that the data written to the SRL is stable on the F5 output of the slice.
T_{REGXB}/ T_{REGYB}	CLK to XB/YB outputs	Time after the Clock (CLK) of a Write operation that the data written to the SRL is stable on the XB/YB outputs of the slice.
Setup/Hold Times for Slice LUT Configured as SRL (Select Shift Register)		
T_{xxS} = Setup time (before clock edge) T_{xxH}= Hold time (after clock edge)		The following descriptions are for setup times only.
T_{WS}/ T_{WH}	CE input (WE)	Time before the clock that the write enable signal must be stable at the WE input of the slice LUT (configured as SRL).
T_{DS}/ T_{DH}	BX/BY configured as data input (DI)	Time before the clock that the data must be stable at the BX/BY input of the slice.

Slice SRL Timing Characteristics

Figure 5-25 illustrates the timing characteristics of a 16-bit shift register implemented in a Virtex-4 FPGA slice (LUT configured as SRL).

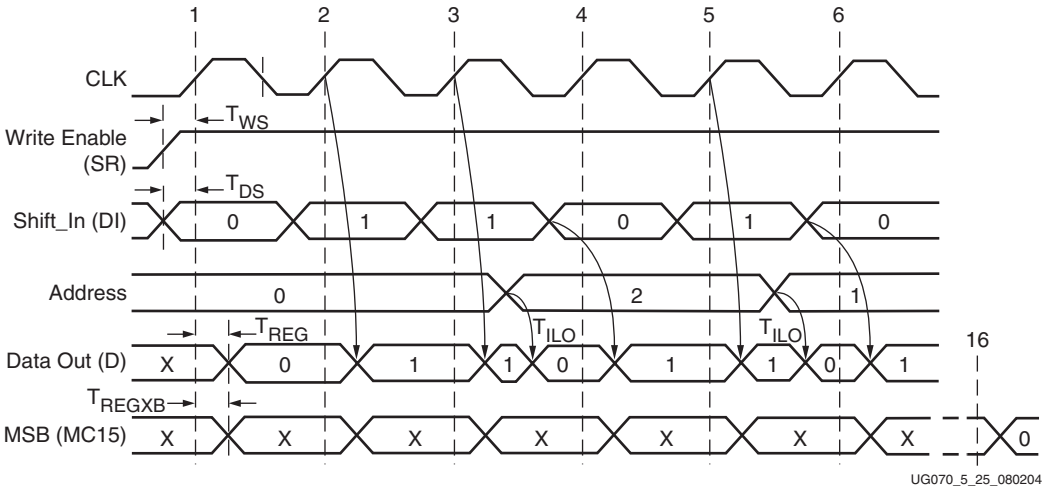


Figure 5-25: **Slice SRL Timing Characteristics**

Clock Event 1: Shift_In

During a Write (Shift_In) operation, the single-bit content of the register at the address on the ADDR inputs is changed, as data is shifted through the SRL. The data written to this register is reflected on the X/Y outputs synchronously, if the address is unchanged during the clock event. If the ADDR inputs are changed during a clock event, the value of the data at the addressable output (D) is invalid.

- At time T_{WSS} before clock event 1, the write-enable signal (SR) becomes valid-High, enabling the SRL for the Write operation that follows.
- At time T_{DS} before clock event 1 the data becomes valid (0) at the DI input of the SRL and is reflected on the X/Y output after a delay of length T_{REG} after clock event 1. Since the address 0 is specified at clock event 1, the data on the DI input is reflected at the D output, because it is written to register 0.

Clock Event 2: Shift_In

- At time T_{DS} before clock event 2, the data becomes valid (1) at the DI input of the SRL and is reflected on the X/Y output after a delay of length T_{REG} after clock event 2. Since the address 0 is still specified at clock event 2, the data on the DI input is reflected at the D output, because it is written to register 0.

Clock Event 3: Shift_In/Addressable (Asynchronous) READ

All Read operations are asynchronous to the CLK signal. If the address is changed (between clock events), the contents of the register at that address are reflected at the addressable output (X/Y outputs) after a delay of length T_{ILO} (propagation delay through a LUT).

- At time T_{DS} before clock event 3 the data becomes valid (1) at the DI input of the SRL, and is reflected on the X/Y output T_{REG} time after clock event 3.
- The address is changed (from 0 to 2) some time after clock event 3. The value stored in register 2 at this time is a 0 (in this example, this was the first data shifted in), and it is reflected on the X/Y output after a delay of length T_{ILO}.

Clock Event 16: MSB (Most Significant Bit) Changes

At time T_{REGXB} after clock event 16, the first bit shifted into the SRL becomes valid (logical 0 in this case) on the XB output of the slice via the MC15 output of the LUT (SRL). This is also applicable for the XMUX, YMUX, XB, YB, C_{OUT}, and F5 outputs at time T_{WOSCO}, T_{WOSX}, T_{WOSXB}, and T_{WOSYB} after clock event 16.

Slice Carry-Chain Timing Model and Parameters

Figure 5-26 illustrates a carry-chain in a Virtex-4 FPGA slice. Some elements of the Virtex-4 FPGA slice have been omitted for clarity. Only the elements relevant to the timing paths described in this section are shown.

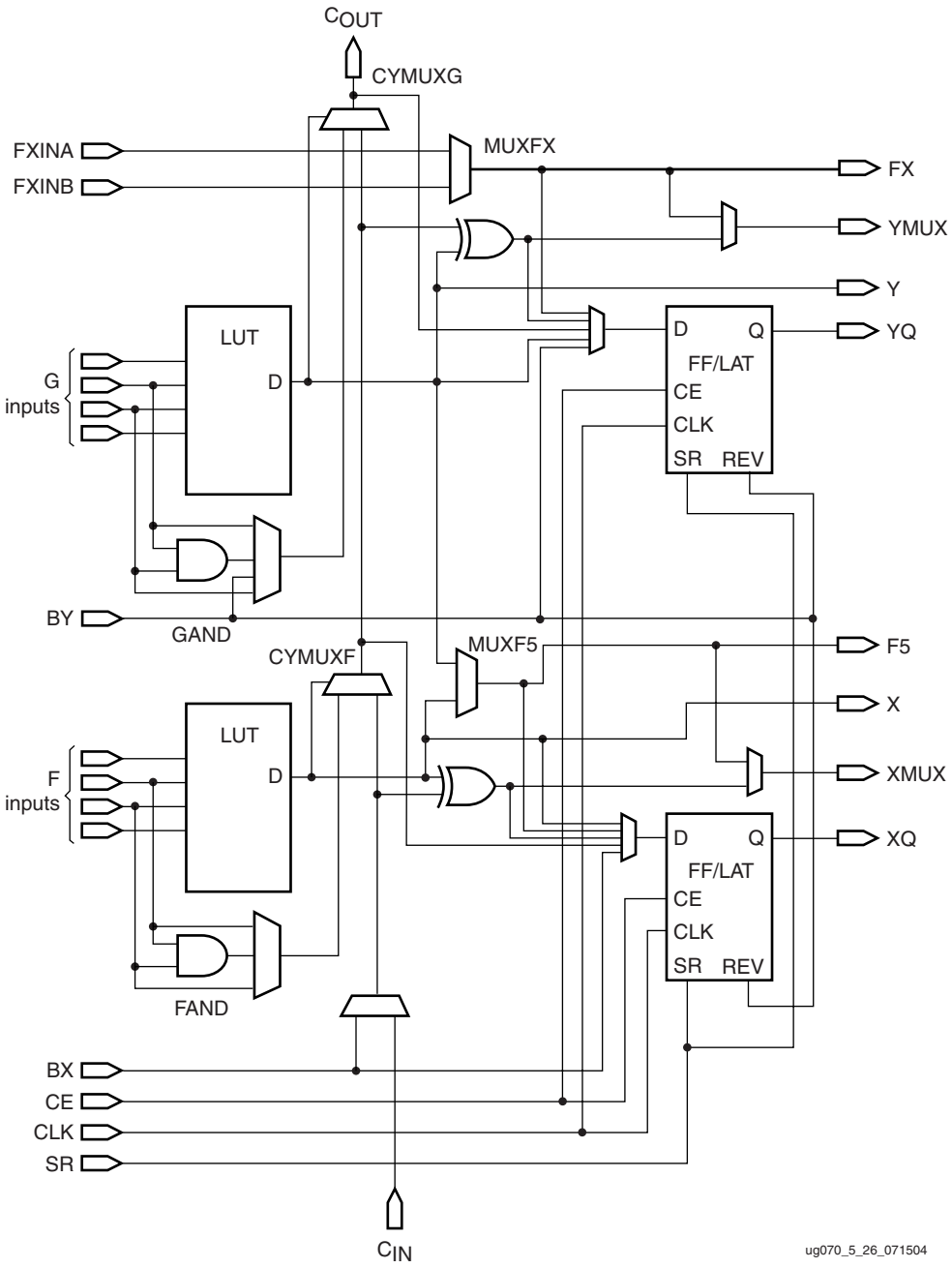


Figure 5-26: **Simplified Virtex-4 FPGA Slice Carry-Chain Diagram**

Slice Carry-Chain Timing Parameters

Table 5-8 shows the slice carry-chain timing parameters for a majority of the paths in Figure 5-26.

Table 5-8: **Slice Carry-Chain Timing Parameters**

Parameter	Function	Description
Sequential Delays for Slice LUT Configured as Carry Chain		
T_{BXCY}/ T_{BYCY}	BX/BY input to C_{OUT} output	Propagation delay from the BX/BY inputs of the slice, to C_{OUT} output of the slice.
T_{BYP}	C_{IN} input to C_{OUT} output	Propagation delay from the C_{IN} input of the slice, to C_{OUT} output of the slice.
T_{FANDCY}/ T_{GANDCY}	F/G input to C_{OUT} output	Propagation delay from the F/G inputs of the slice, to C_{OUT} output of the slice using FAND (product).
T_{OPCYF}/ T_{OPCYG}	F/G input to C_{OUT} output	Propagation delay from the F/G input of the slice to C_{OUT} output of the slice.
T_{OPX}/ T_{OPY}	F/G input to XMUX/YMUX output	Propagation delay from the F/G inputs of the slice, to XMUX/YMUX output of the slice using XOR (sum).
Setup/Hold Times for Slice LUT Configured as Carry Chain		
T_{xxS} = Setup time (before clock edge) T_{xxH} = Hold time (after clock edge)		The following descriptions are for setup times only.
T_{CINCK}/ T_{CKCIN}	C_{IN} Data inputs (DI)	Time before Clock (CLK) that data from the C_{IN} input of the slice must be stable at the D-input of the slice sequential elements (configured as a flip-flop). Figure 5-27 shows the worst-case path.

Slice Carry-Chain Timing Characteristics

Figure 5-27 illustrates the timing characteristics of a slice carry chain implemented in a Virtex-4 FPGA slice.

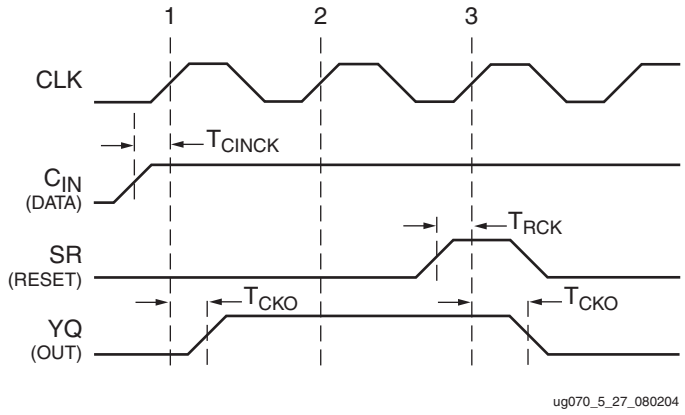


Figure 5-27: **Slice Carry-Chain Timing Characteristics**

- At time T_{CINCK} before clock event 1, data from C_{IN} input becomes valid-High at the D input of the slice register. This is reflected on either the XQ or YQ pin at time T_{CKO} after clock event 1.

- At time T_{SRCK} before clock event 3, the SR signal (configured as synchronous reset in this case) becomes valid-High, resetting the slice register. This is reflected on either the XQ or YQ pin at time T_{CKO} after clock event 3.

CLB Primitives and Verilog/VHDL Examples

Distributed RAM Primitives

Four primitives are available; from 16 x 1-bit to 64 x 1-bit. Three primitives are single-port RAM, and one primitive is a dual-port RAM, as shown in Table 5-9.

Table 5-9: **Single-Port and Dual-Port Distributed RAM**

Primitive	RAM Size	Type	Address Inputs
RAM16X1S	16 bits	single-port	A3, A2, A1, A0
RAM32X1S	32 bits	single-port	A4, A3, A2, A1, A0
RAM64X1S	64 bits	single-port	A5, A4, A3, A2, A1, A0
RAM16X1D	16 bits	dual-port	A3, A2, A1, A0

The input and output data are 1-bit wide. However, several distributed RAMs can be used to implement wide memory blocks.

Figure 5-28 shows generic single-port and dual-port distributed RAM primitives. The A and DPRA signals are address busses.

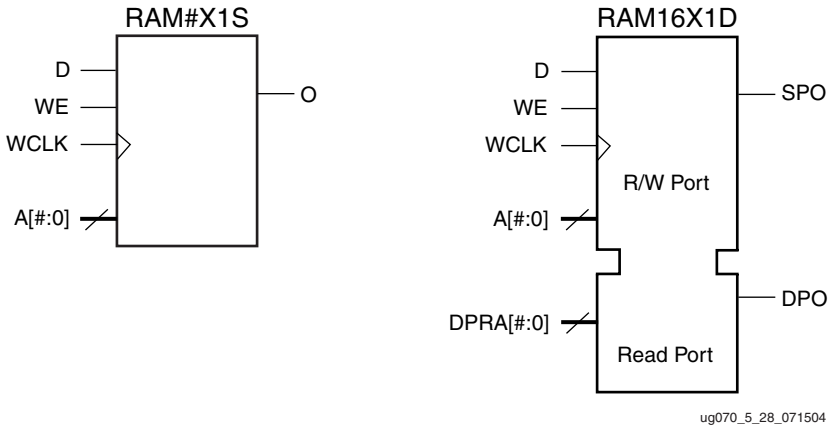


Figure 5-28: **Single-Port and Dual-Port Distributed RAM Primitive**

As shown in Table 5-10, wider primitives are available for 2-bit, 4-bit, and 8-bit RAM.

Table 5-10: **Wider Primitives**

Primitive	RAM Size	Data Inputs	Address Inputs	Data Outputs
RAM16X2S	16 x 2-bit	D1, D0	A3, A2, A1, A0	O1, O0
RAM32X2S	32 x 2-bit	D1, D0	A4, A3, A2, A1, A0	O1, O0
RAM16X4S	16 x 4-bit	D3, D2, D1, D0	A3, A2, A1, A0	O3, O2, O1, O0

VHDL and Verilog Instantiations

VHDL and Verilog instantiation templates are available as examples (see VHDL and Verilog Templates).

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

The RAM_#S templates (with # = 16, 32, 64) are single-port modules and instantiate the corresponding RAM#X1S primitive.

RAM_16D templates are dual-port modules and instantiate the corresponding RAM16X1D primitive.

Port Signals

Each distributed RAM port operates independently of the other while reading the same set of memory cells.

Clock - WCLK

The clock is used for the synchronous write. The data and the address input pins have setup time referenced to the WCLK pin.

Enable - WE

The enable pin affects the write functionality of the port. An inactive Write Enable prevents any writing to memory cells. An active Write Enable causes the clock edge to write the data input signal to the memory location pointed to by the address inputs.

Address - A0, A1, A2, A3 (A4, A5)

The address inputs select the memory cells for read or write. The width of the port determines the required address inputs. Note that the address inputs are not a bus in VHDL or Verilog instantiations.

Data In - D

The data input provides the new data value to be written into the RAM.

Data Out - O, SPO, and DPO

The data out O (Single-Port or SPO) and DPO (Dual-Port) reflects the contents of the memory cells referenced by the address inputs. Following an active write clock edge, the data out (O or SPO) reflects the newly written data.

Inverting Control Pins

The two control pins (WCLK and WE) each have an individual inversion option. Any control signal, including the clock, can be active at 0 (negative edge for the clock) or at 1 (positive edge for the clock) without requiring other logic resources.

Global Set/Reset - GSR

The global set/reset (GSR) signal does not affect distributed RAM modules. For more information on the GSR, see the BUFGSR section in the Xilinx Software Manual.

Attributes

Content Initialization - INIT

With the INIT attributes, users can define the initial memory contents after configuration. By default distributed RAM is initialized with all zeros during the device configuration sequence. The initialization attribute INIT represents the specified memory contents. Each INIT is a hex-encoded bit vector. Table 5-11 shows the length of the INIT attribute for each primitive.

Table 5-11: **INIT Attributes Length**

Primitive	Template	INIT Attribute Length
RAM16X1S	RAM_16S	4 digits
RAM32X1S	RAM_32S	8 digits
RAM64X1S	RAM_64S	16 digits
RAM16X1D	RAM_16S	4 digits

Initialization in VHDL or Verilog Codes

Distributed RAM structures can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the distributed RAM instantiation and are copied in the EDIF output file to be compiled by Xilinx Alliance Series tools. The VHDL code simulation uses a generic parameter to pass the attributes. The Verilog code simulation uses a *defparam* parameter to pass the attributes.

The distributed RAM instantiation templates (in VHDL and Verilog) illustrate these techniques (“VHDL and Verilog Templates”).

Location Constraints

The CLB has four slices S0, S1, S2 and S3. As an example, in the bottom left CLB, the slices have the coordinates shown in Figure 5-1.

Distributed RAM instances can have LOC properties attached to them to constrain placement. The RAM16X1S primitive fits in any LUT of slices S0 or S2.

For example, the instance U_RAM16 is placed in slice X0Y0 with the following LOC properties:

```
INST "U_RAM16" LOC = "SLICE_X0Y0";
```

Distributed RAM placement locations use the slice location naming convention, allowing LOC properties to transfer easily from array to array.

Creating Larger RAM Structures

Wider and/or deeper memory structures can be created using multiple distributed RAM instances. Table 5-12 shows the generic VHDL and Verilog distributed RAM examples provided to implement n-bit-wide memories.

Table 5-12: **VHDL and Verilog Submodules**

Submodules	Primitive	Size	Type
XC4V_RAM16XN_S	RAM16X1S	16 words x *n*-bit	single-port
XC4V_RAM32XN_S	RAM32X1S	32 words x *n*-bit	single-port
XC4V_RAM64XN_S	RAM64X1S	64 words x *n*-bit	single-port
XC4V_RAM16XN_D	RAM16X1D	16 words x *n*-bit	dual-port

By using the read/write port for the write address and the second read port for the read address, a FIFO that can read and write simultaneously is easily generated. Simultaneous access doubles the effective throughput of the memory.

VHDL and Verilog Templates

VHDL and Verilog templates are available for all single-port and dual-port primitives. The number in each template indicates the number of bits (for example, RAM_16S is the template for the 16 x 1-bit RAM); S indicates single-port, and D indicates dual-port.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

The single-port templates are:

- RAM_16S
- RAM_32S
- RAM_64S

The dual-port templates are:

- RAM_16D

Templates for the RAM_16S module are provided in VHDL and Verilog code as examples.

VHDL Template

```
--
-- Module: RAM_16S
--
-- Description: VHDL instantiation template
--              Distributed RAM
--              Single Port 16 x 1
--              can be used also for RAM16X1S_1
--
-- Device: Virtex-4 Family
--
------------------------------------------------------------------------
--
-- Components Declarations:
--
```

```
component RAM16X1S
   generic (
        INIT : bit_vector := X"0000"
        );
  port (
        D    : in std_logic;
        WE   : in std_logic;
        WCLK : in std_logic;
        A0   : in std_logic;
        A1   : in std_logic;
        A2   : in std_logic;
        A3   : in std_logic;
        O    : out std_logic
        );
end component;
--
-----------------------------------------------------------------------
--
-- Architecture section:
--
-- Attributes for RAM initialization ("0" by default):
attribute INIT: string;
--
attribute INIT of U_RAM16X1S: label is "0000";
--
-- Distributed RAM Instantiation
U_RAM16X1S: RAM16X1S
  port map (
   D      => , -- insert input signal
   WE     => , -- insert Write Enable signal
   WCLK   => , -- insert Write Clock signal
   A0     => , -- insert Address 0 signal
   A1     => , -- insert Address 1 signal
   A2     => , -- insert Address 2 signal
   A3     => , -- insert Address 3 signal
   O      =>   -- insert output signal
  );
--
-----------------------------------------------------------------------
```

Verilog Template

```
//
// Module: RAM_16S
//
// Description: Verilog instantiation template
//              Distributed RAM
//              Single Port 16 x 1
//              can be used also for RAM16X1S_1
//
// Device: Virtex-4 Family
//
//-----------------------------------------------------------------------
//Distributed RAM Instantiation
RAM16X1S U_RAM16X1S (  .D(),      // insert input signal
                 .WE(),     // insert Write Enable signal
                .WCLK(),   // insert Write Clock signal
                 .A0(),     // insert Address 0 signal
                 .A1(),     // insert Address 1 signal
```

```
.A2(),      // insert Address 2 signal
.A3(),      // insert Address 3 signal
.O()        // insert output signal
 );
```

Shift Registers (SRLs) Primitives and Verilog/VHDL Example

This section provides generic VHDL and Verilog submodules and reference code examples for implementing from 16-bit up to 64-bit shift registers. These submodules are built from 16-bit shift-register primitives and from dedicated MUXF5, MUXF6, MUXF7, and MUXF8 multiplexers.

SRL Primitives and Submodules

Eight primitives are available that offer optional clock enable (CE), inverted clock ($\overline{CLK}$) and cascadable output (Q15) combinations.

Table 5-13 lists all of the available primitives for synthesis and simulation.

Table 5-13: **Shift Register Primitives**

Primitive	Length	Control	Address Inputs	Output
SRL16	16 bits	CLK	A3,A2,A1,A0	Q
SRL16E	16 bits	CLK, CE	A3,A2,A1,A0	Q
SRL16_1	16 bits	$\overline{CLK}$	A3,A2,A1,A0	Q
SRL16E_1	16 bits	$\overline{CLK}$, CE	A3,A2,A1,A0	Q
SRLC16	16 bits	CLK	A3,A2,A1,A0	Q, Q15
SRLC16E	16 bits	CLK, CE	A3,A2,A1,A0	Q, Q15
SRLC16_1	16 bits	$\overline{CLK}$	A3,A2,A1,A0	Q, Q15
SRLC16E_1	16 bits	$\overline{CLK}$, CE	A3,A2,A1,A0	Q, Q15

In addition to the 16-bit primitives, 32-bit and 64-bit cascadable shift registers can be implemented in VHDL and Verilog. Table 5-14 lists the available submodules.

Table 5-14: **Shift Register Submodules**

Submodule	Length	Control	Address Inputs	Output
SRLC32E_MACRO	32 bits	CLK, CE	A4,A3,A2,A1,A0	Q, Q31
SRLC64E_MACRO	64 bits	CLK, CE	A5, A4, A3,A2,A1,A0	Q, Q63

The submodules are based on SRLC16E primitives and are associated with dedicated multiplexers (MUXF5, MUXF6, and so forth). This implementation allows a fast static- and dynamic-length mode, even for very large shift registers.

Figure 5-29 represents the cascadable shift registers (32-bit and 64-bit) implemented by the submodules in Table 5-14.

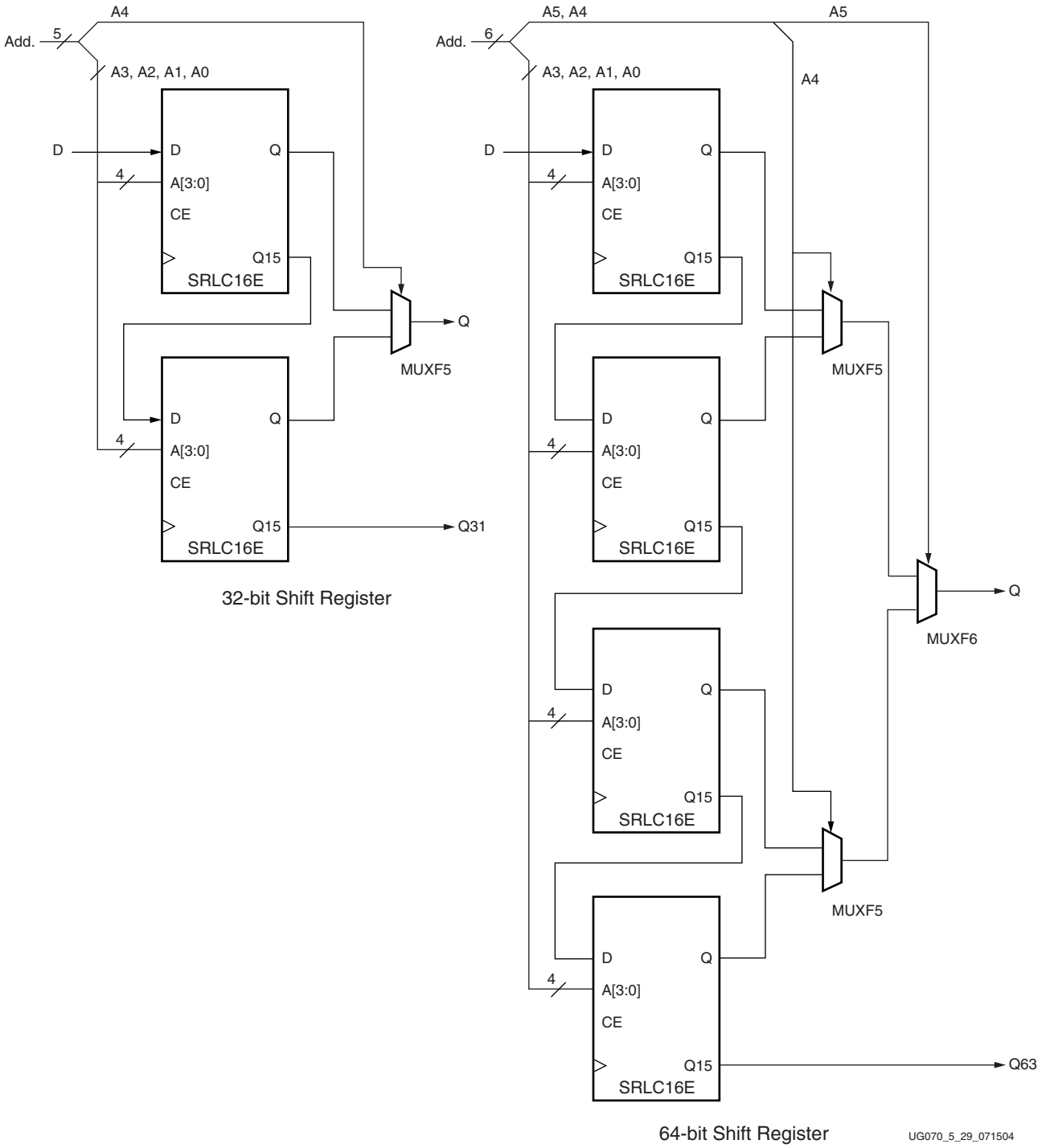


Figure 5-29: **Shift-Register Submodules (32-bit, 64-bit)**

All clock enable (CE) and clock (CLK) inputs are connected to one global clock enable and one clock signal per submodule. If a global static- or dynamic-length mode is not required, the SRLC16E primitive can be cascaded without multiplexers.

Initialization in VHDL or Verilog Code

A shift register can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attribute is attached to the 16-bit shift register instantiation and is copied in the EDIF output file to be compiled by Xilinx Alliance Series tools. The VHDL code simulation uses a generic parameter to pass the attributes. The Verilog code simulation uses a *defparam* parameter to pass the attributes.

The Virtex-4_SRL16E shift register instantiation code examples (in VHDL and Verilog) illustrate these techniques ("VHDL and Verilog Templates"). `Virtex-4_SRL16E.vhd` and `Virtex-4_SRL16E.v` files are not a part of the documentation.

Port Signals

Clock - CLK

Either the rising edge or the falling edge of the clock is used for the synchronous shift operation. The data and clock enable input pins have setup times referenced to the chosen edge of CLK.

Data In - D

The data input provides new data (one bit) to be shifted into the shift register.

Clock Enable - CE (optional)

The clock enable pin affects shift functionality. An inactive clock enable pin does not shift data into the shift register and does not write new data. Activating the clock enable allows the data in (D) to be written to the first location and all data to be shifted by one location. When available, new data appears on output pins (Q) and the cascadable output pin (Q15).

Address - A0, A1, A2, A3

Address inputs select the bit (range 0 to 15) to be read. The nth bit is available on the output pin (Q). Address inputs have no effect on the cascadable output pin (Q15); it is always the last bit of the shift register (bit 15).

Data Out - Q

The data output Q provides the data value (1 bit) selected by the address inputs.

Data Out - Q15 (optional)

The data output Q15 provides the last bit value of the 16-bit shift register. New data becomes available after each shift-in operation.

Inverting Control Pins

The two control pins (CLK, CE) have an individual inversion option. The default is the rising clock edge and active High clock enable.

Global Set/Reset - GSR

The global set/reset (GSR) signal has no impact on shift registers.

Attributes

Content Initialization - INIT

The INIT attribute defines the initial shift register contents. The INIT attribute is a hex-encoded bit vector with four digits (0000).The left-most hexadecimal digit is the most significant bit. By default the shift register is initialized with all zeros during the device configuration sequence, but any other configuration value can be specified.

Location Constraints

Each CLB resource has four slices: S0, S1, S2, and S3. As an example, in the bottom left CLB resource, each slice has the coordinates shown in Table 5-15.

Table 5-15: **Slice Coordinates in the Bottom-Left CLB Resource**

Slice S3	Slice S2	Slice S1	Slice S0
X1Y1	X0Y1	X1Y0	X0Y0

To constrain placement, shift register instances can have LOC properties attached to them. Each 16-bit shift register fits in one LUT.

A 32-bit shift register in static or dynamic address mode fits in one slice (two LUTs and one MUXF5). This shift register can be placed in SLICEM only.

A 64-bit shift register in static or dynamic address mode fits in two slices. These slices are S0 and S2. Figure 5-30 illustrates the position of the four LUTs in a CLB resource.

The dedicated CLB shift chain runs from the top slice to the bottom slice. The data input pin must either be in slice S0 or in S2. The address selected as the output pin (Q) is the MUXF6 output.

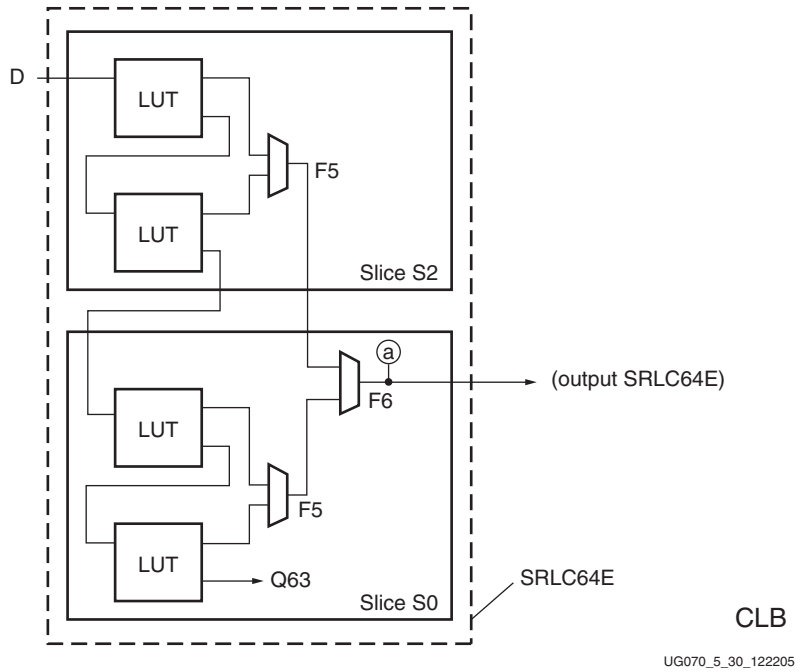


Figure 5-30: **Shift Register Placement**

Fully Synchronous Shift Registers

All shift-register primitives and submodules do not use the register(s) available in the same slice(s). To implement a fully synchronous read and write shift register, output pin Q must be connected to a flip-flop. Both the shift register and the flip-flop share the same clock, as shown in Figure 5-31.

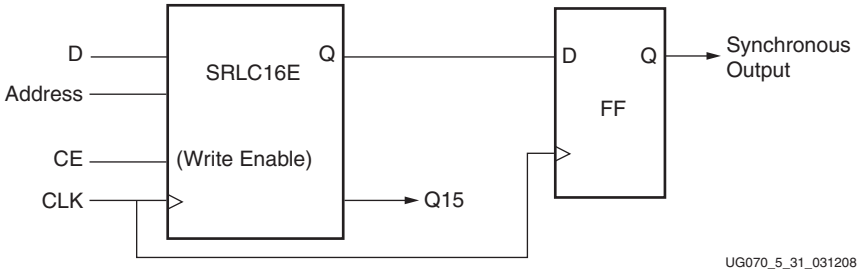


Figure 5-31: **Fully Synchronous Shift Register**

This configuration provides a better timing solution and simplifies the design. Because the flip-flop must be considered to be the last register in the shift-register chain, the static or dynamic address should point to the desired length minus one. If needed, the cascadable output can also be registered in a flip-flop.

Static-Length Shift Registers

The cascadable16-bit shift register implements any static length mode shift register without the dedicated multiplexers (MUXF5, MUXF6,…). Figure 5-32 illustrates a 40-bit shift register. Only the last SRLC16E primitive needs to have its address inputs tied to `0111`. Alternatively, shift register length can be limited to 39 bits (address tied to `0110`) and a flip-flop can be used as the last register. (In an SRLC16E primitive, the shift register length is the address input + 1.)

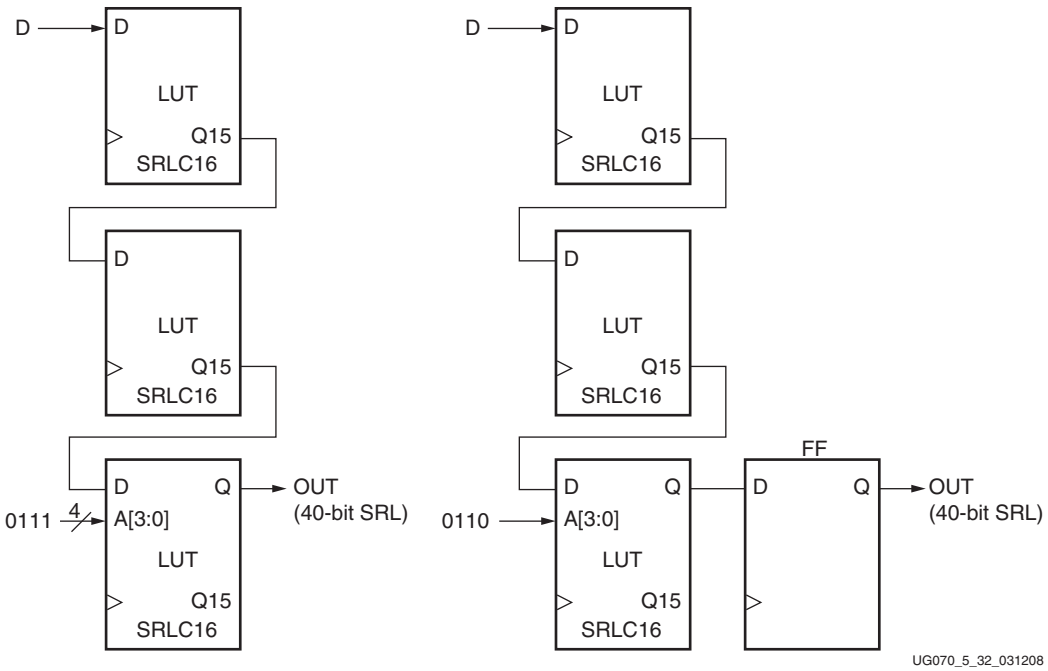


Figure 5-32: **40-bit Static-Length Shift Register**

VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available for all primitives and submodules.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

The ShiftRegister_C_x (with x = 16, 32, or 64) templates are cascadable modules and instantiate the corresponding SRLCxE primitive (16) or submodule (32 or 64).

The ShiftRegister_16 template can be used to instantiate an SRL16 primitive.

VHDL and Verilog Templates

In template nomenclature, the number indicates the number of bits (for example, SHIFT_REGISTER_16 is the template for the 16-bit shift register). A "C" extension means the template is cascadable.

The following are templates for primitives:

- SHIFT_REGISTER_16
- SHIFT_REGISTER_C_16

The following are templates for submodules:

- SHIFT_REGISTER_C_32 (submodule: SRLC32E_SUBM)
- SHIFT_REGISTER_C_64 (submodule: SRLC64E_SUBM)

The corresponding submodules have to be synthesized with the design.

Templates for the SHIFT_REGISTER_16_C module are provided in VHDL and Verilog code as an example.

VHDL Template

```
-- Module: SHIFT_REGISTER_C_16
-- Description: VHDL instantiation template
--  CASCADABLE 16-bit shift register with enable (SRLC16E)
-- Device: Virtex-4 Family
-----------------------------------------------------------------------
-- Components Declarations:
--
component SRLC16E
        INIT : bit_vector := X"0000"
  );
  port (
         D : in std_logic;
        CE   : in std_logic;
        CLK  : in std_logic;
        A0   : in std_logic;
        A1   : in std_logic;
        A2   : in std_logic;
        A3   : in std_logic;
        Q    : out std_logic;
        Q15  : out std_logic
  );
end component;
-- Architecture  Section:
--
-- Attributes for Shift Register initialization ("0" by default):
```

```
attribute INIT: string;
--
attribute INIT of U_SRLC16E: label is "0000";
--
-- ShiftRegister Instantiation
U_SRLC16E: SRLC16E
  port map (
  D      => , -- insert input signal
  CE     => , -- insert Clock Enable signal (optional)
  CLK    => , -- insert Clock signal
  A0     => , -- insert Address 0 signal
  A1     => , -- insert Address 1 signal
  A2     => , -- insert Address 2 signal
  A3     => , -- insert Address 3 signal
  Q      => , -- insert output signal
  Q15    =>   -- insert cascadable output signal
  );
```

Verilog Template

```
// Module: SHIFT_REGISTER_16
// Description: Verilog instantiation template
// Cascadable 16-bit Shift Register with Clock Enable (SRLC16E)
// Device: Virtex-4 Family
//-----------------------------------------------------------------------
  defparam
  SRLC16E U_SRLC16E   (   .D(),
                          .A0(),
                          .A1(),
                          .A2(),
                          .A3(),
                          .CLK(),
                          .CE(),
                          .Q(),
                          .Q15()
            );
```

Multiplexer Primitives and Verilog/VHDL Examples

This section provides generic VHDL and Verilog reference code implementing multiplexers. These submodules are built from LUTs and the dedicated MUXF5, MUXF6, MUXF7, and MUXF8 multiplexers. To automatically generate large multiplexers using these dedicated elements, use the CORE Generator software Bit Multiplexer and Bus Multiplexer modules.

For applications such as comparators, encoder-decoders or “case” statement in VHDL or Verilog, these resources offer an optimal solution.

Multiplexer Primitives and Submodules

Four primitives are available for access to the dedicated MUXFX in each slice. In the example shown in Table 5-16, MUXF7 is available only in slice S2.

Table 5-16: **MUXFX Resources**

Primitive	Slice	Control	Input	Output
MUXF5	S0, S1, S2, S3	S	I0, I1	O
MUXF6	S0, S1	S	I0, I1	O
MUXF7	S2	S	I0, I1	O
MUXF8	S3	S	I0, I1	O

In addition to the primitives, five submodules to implement multiplexers from 2:1 to 32:1 are provided in VHDL and Verilog code. Synthesis tools can automatically infer these primitives (MUXF5, MUXF6, MUXF7, and MUXF8); however, the submodules described in this section use instantiation of the new MUXFX to guarantee an optimized result. Table 5-17 lists available submodules.

Table 5-17: **Available Submodules**

Submodule	Multiplexer	Control	Input	Output
MUX_2_1_SUBM	2:1	SELECT_I	DATA_I[1:0]	DATA_O
MUX_4_1_SUBM	4:1	SELECT_I[1:0]	DATA_I[3:0]	DATA_O
MUX_8_1_SUBM	8:1	SELECT_I[2:0]	DATA_I[8:0]	DATA_O
MUX_16_1_SUBM	16:1	SELECT_I[3:0]	DATA_I[15:0]	DATA_O
MUX_32_1_SUBM	32:1	SELECT_I[4:0]	DATA_I[31:0]	DATA_O

Port Signals

Data In - DATA_I

The data input provides the data to be selected by the SELECT_I signal(s).

Control In - SELECT_I

The select input signal or bus determines the DATA_I signal to be connected to the output DATA_O. For example, the MUX_4_1_SUBM multiplexer has a 2-bit SELECT_I bus and a 4-bit DATA_I bus. Table 5-18 shows the DATA_I selected for each SELECT_I value.

Table 5-18: **Selected Inputs**

SELECT_I[1:0]	DATA_O
0 0	DATA_I[0]
0 1	DATA_I[1]
1 0	DATA_I[2]
1 1	DATA_I[3]

Data Out - DATA_O

The data output O provides the data value (1 bit) selected by the control inputs.

Multiplexer Verilog/VHDL Examples

Multiplexers are used in various applications. These are often inferred by synthesis tools when a "case" statement is used (see the following example). Comparators, encoder-decoders and wide-input combinatorial functions are optimized when they are based on one level of LUTs and dedicated MUXFX resources of the Virtex-4 FPGA CLBs.

VHDL and Verilog Instantiation

The primitives (MUXF5, MUXF6, and so forth) can be instantiated in VHDL or Verilog code, to design wide-input functions.

The submodules (MUX_2_1_SUBM, MUX_4_1_SUBM, and so forth) can be instantiated in VHDL or Verilog code to implement multiplexers. However the corresponding submodule must be added to the design directory as hierarchical submodule. For example, if a module is using the MUX_16_1_SUBM, the MUX_16_1_SUBM.vhd file (VHDL code) or MUX_16_1_SUBM.v file (Verilog code) must be compiled with the design source code. The submodule code can also be copied into the designer source code.

VHDL and Verilog Submodules

VHDL and Verilog submodules are available to implement multiplexers up to 32:1. They illustrate how to design with the MUXFX resources. When synthesis infers the corresponding MUXFX resource(s), the VHDL or Verilog code is behavioral code ("case" statement). Otherwise, the equivalent "case" statement is provided in comments and the correct MUXFX are instantiated. However, most synthesis tools support the inference of all of the MUXFX. The examples are guidelines for designing other wide-input functions. The available submodules are:

- MUX_2_1_SUBM (behavioral code)
- MUX_4_1_SUBM
- MUX_8_1_SUBM
- MUX_16_1_SUBM
- MUX_32_1_SUBM

The corresponding submodules have to be synthesized with the design. The submodule MUX_16_1_SUBM is provided as an example in VHDL and Verilog.

VHDL Template

```
-- Module: MUX_16_1_SUBM
-- Description: Multiplexer 16:1
-- Device: Virtex-4 Family
--------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity MUX_16_1_SUBM is
    port (
        DATA_I: in std_logic_vector (15 downto 0);
        SELECT_I: in std_logic_vector (3 downto 0);
```

```
        DATA_O: out std_logic
  );
end MUX_16_1_SUBM;
architecture MUX_16_1_SUBM_arch of MUX_16_1_SUBM is
-- Component Declarations:
component MUXF7
    port (
    I0: in std_logic;
    I1: in std_logic;
    S: in std_logic;
    O: out std_logic
    );
end component;
signal DATA_MSB : std_logic;
signal DATA_LSB : std_logic;
--
begin
SELECT_PROCESS_LSB: process (SELECT_I, DATA_I)
begin
  case SELECT_I (2 downto 0) is
    when "000" => DATA_LSB <= DATA_I (0);
    when "001" => DATA_LSB <= DATA_I (1);
    when "010" => DATA_LSB <= DATA_I (2);
    when "011" => DATA_LSB <= DATA_I (3);
    when "100" => DATA_LSB <= DATA_I (4);
    when "101" => DATA_LSB <= DATA_I (5);
    when "110" => DATA_LSB <= DATA_I (6);
    when "111" => DATA_LSB <= DATA_I (7);
    when others => DATA_LSB <= 'X';
  end case;
end process SELECT_PROCESS_LSB;
--
SELECT_PROCESS_MSB: process (SELECT_I, DATA_I)
begin
  case SELECT_I (2 downto 0) is
    when "000" => DATA_MSB <= DATA_I (8);
    when "001" => DATA_MSB <= DATA_I (9);
    when "010" => DATA_MSB <= DATA_I (10);
    when "011" => DATA_MSB <= DATA_I (11);
    when "100" => DATA_MSB <= DATA_I (12);
    when "101" => DATA_MSB <= DATA_I (13);
    when "110" => DATA_MSB <= DATA_I (14);
    when "111" => DATA_MSB <= DATA_I (15);
    when others => DATA_MSB <= 'X';
  end case;
end process SELECT_PROCESS_MSB;
--
-- MUXF7 instantiation
U_MUXF7: MUXF7
    port map (
    I0 => DATA_LSB,
    I1 => DATA_MSB,
    S  => SELECT_I (3),
    O  => DATA_O
    );
--
end MUX_16_1_SUBM_arch;
--
```

Verilog Template

```
// Module: MUX_16_1_SUBM
//
// Description: Multiplexer 16:1
// Device: Virtex-4 Family
//-----------------------------------------------------------------------
//
module MUX_16_1_SUBM (DATA_I, SELECT_I, DATA_O);
input [15:0]DATA_I;
input [3:0]SELECT_I;

output DATA_O;

wire [2:0]SELECT;

reg DATA_LSB;
reg DATA_MSB;

assign SELECT[2:0] = SELECT_I[2:0];
always @ (SELECT or DATA_I)
    case (SELECT)
        3'b000 : DATA_LSB <= DATA_I[0];
  3'b001 : DATA_LSB <= DATA_I[1];
  3'b010 : DATA_LSB <= DATA_I[2];
  3'b011 : DATA_LSB <= DATA_I[3];
        3'b100 : DATA_LSB <= DATA_I[4];
  3'b101 : DATA_LSB <= DATA_I[5];
  3'b110 : DATA_LSB <= DATA_I[6];
  3'b111 : DATA_LSB <= DATA_I[7];
  default : DATA_LSB <= 1'bx;
    endcase

always @ (SELECT or DATA_I)

    case (SELECT)
        3'b000 : DATA_MSB <= DATA_I[8];
  3'b001 : DATA_MSB <= DATA_I[9];
  3'b010 : DATA_MSB <= DATA_I[10];
  3'b011 : DATA_MSB <= DATA_I[11];
        3'b100 : DATA_MSB <= DATA_I[12];
  3'b101 : DATA_MSB <= DATA_I[13];
  3'b110 : DATA_MSB <= DATA_I[14];
  3'b111 : DATA_MSB <= DATA_I[15];
  default : DATA_MSB <= 1'bx;
    endcase

// MUXF7 instantiation

MUXF7 U_MUXF7   (.I0(DATA_LSB),
     .I1(DATA_MSB),
     .S(SELECT_I[3]),
     .O(DATA_O)
     );
endmodule
//
*/
```

Chapter 6

SelectIO Resources

I/O Tile Overview

Input/output characteristics and logic resources are covered in three consecutive chapters. Chapter 6, "SelectIO Resources," describes the electrical behavior of the output drivers and input receivers, and gives detailed examples of many standard interfaces. Chapter 7, "SelectIO Logic Resources," describes the input and output data registers and their Double-Data-Rate (DDR) operation, and the programmable input delay (IDELAY). Chapter 8, "Advanced SelectIO Logic Resources," describes the data serializer/deserializer (SERDES).

An I/O tile contains two IOBs, two ILOGICs, and two OLOGICs. Figure 6-1 shows a Virtex-4 FPGA I/O tile.

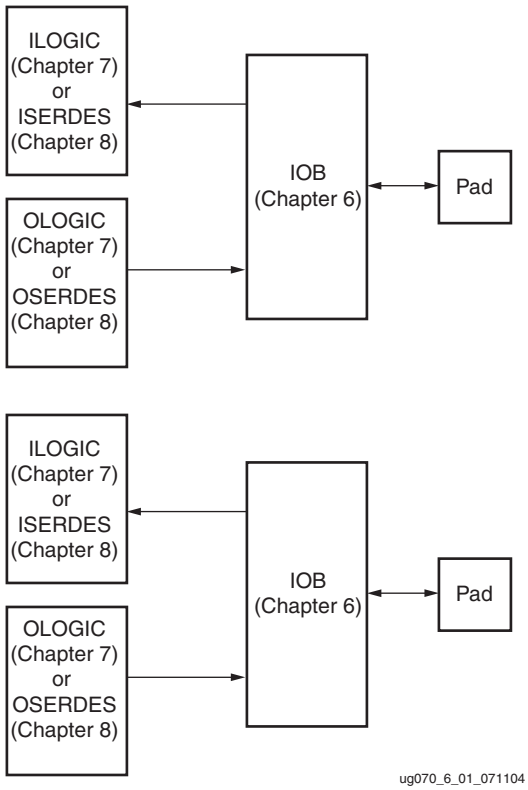


Figure 6-1: **Virtex-4 FPGA I/O Tile**

SelectIO Resources Introduction

All Virtex-4 FPGAs have configurable high-performance SelectIO™ technology drivers and receivers, supporting a wide variety of standard interfaces. The robust feature set includes programmable control of output strength and slew rate, and on-chip termination using Digitally Controlled Impedance (DCI). All banks can support 3.3V I/O.

Each IOB contains both input, output, and 3-state SelectIO drivers. These drivers can be configured to various I/O standards. Differential I/O uses the two IOBs grouped together in one tile.

- Single-ended I/O standards (LVCMOS, LVTTL, HSTL, SSTL, GTL, PCI)
- Differential I/O standards (LVDS, LDT, LVPECL, BLVDS, CSE Differential HSTL and SSTL)

Note: Differential and V_{REF}-dependent inputs are powered by V_{CCAUX}.

Each Virtex-4 FPGA I/O tile contains two IOBs, and also two ILOGIC blocks and two OLOGIC blocks, as described in Chapter 7, "SelectIO Logic Resources".

Figure 6-2 shows the basic IOB and its connections to the internal logic and the device pad.

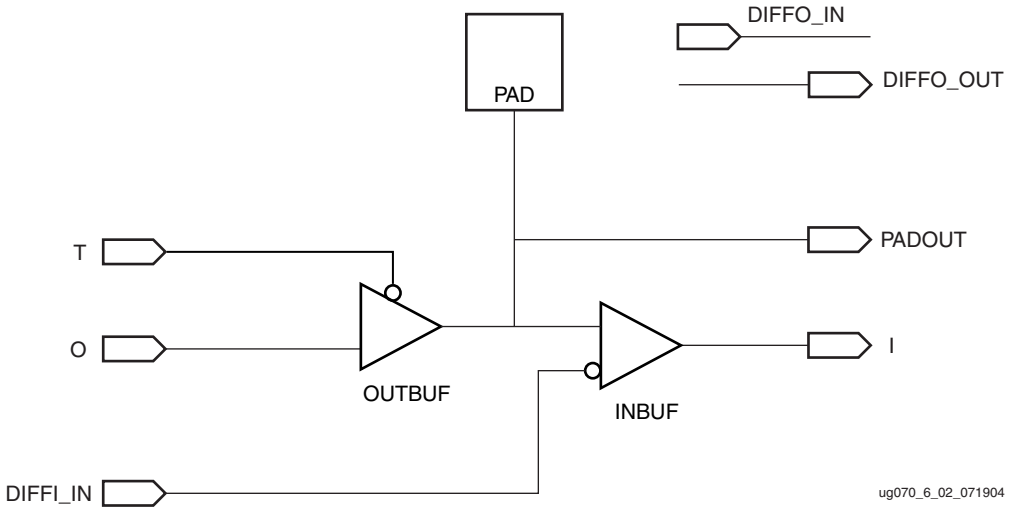


Figure 6-2: **Basic IOB Diagram**

Each IOB has a direct connection to an ILOGIC/OLOGIC pair containing the input and output logic resources for data and 3-state control for the IOB. When using multiple clocks in Virtex-4 FPGA I/O tiles, the input clocks to the two ILOGIC blocks and the two OLOGIC blocks are not shared. Both ILOGIC and OLOGIC can be configured as ISERDES and OSERDES, respectively, as described in Chapter 8, "Advanced SelectIO Logic Resources."

SelectIO Technology Resources General Guidelines

This section summarizes the general guidelines to be considered when designing with the SelectIO technology resources of Virtex-4 FPGAs.

Virtex-4 FPGA I/O Bank Rules

The number of banks available in Virtex-4 devices is not limited to eight as in previous Xilinx® architectures. In Virtex-4 devices, with some exceptions in the center column, an I/O bank consists of 64 IOBs (32 CLBs and two clock regions). As a result, the number of banks depends upon the device size. In the *Virtex-4 Family Overview* the total number of I/O banks is listed by device type. The XC4VLX25 has 10 usable I/O banks and one configuration bank. Figure 6-3 is an example of a columnar floorplan showing the XC4VLX25 I/O banks.

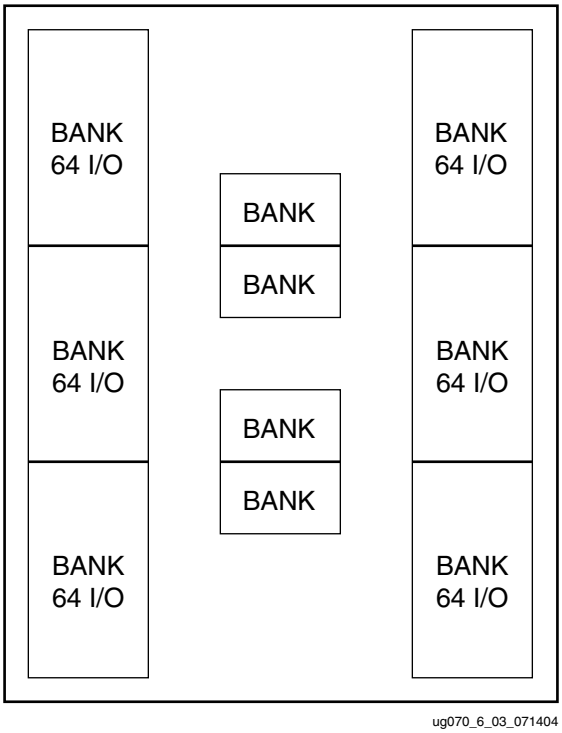


Figure 6-3: **XC4VLX25 I/O Banks**

3.3V I/O Support

The Virtex-4 architecture supports 3.3V single-ended I/O standards in all banks.

Reference Voltage (V_{REF}) Pins

Low-voltage, single-ended I/O standards with a differential amplifier input buffer require an input reference voltage (V_{REF}). V_{REF} is an external input into Virtex-4 devices. Within each I/O bank, one of every 16 I/O pins is automatically configured as a V_{REF} input, if using a single-ended I/O standard requiring a differential amplifier input buffer.

Output Drive Source Voltage (V_{CCO}) Pins

Many of the low-voltage I/O standards supported by Virtex-4 devices require a different output drive voltage (V_{CCO}). As a result, each device often supports multiple output drive source voltages.

Output buffers within a given V_{CCO} bank must share the same output drive source voltage. The following input buffers use the V_{CCO} voltage: LVTTL, LVCMOS, PCI, LVDCI and other DCI standards.

Virtex-4 FPGA Digitally Controlled Impedance (DCI)

Introduction

As FPGAs get bigger and system clock speeds get faster, PC board design and manufacturing becomes more difficult. With ever faster edge rates, maintaining signal integrity becomes a critical issue. PC board traces must be properly terminated to avoid reflections or ringing.

To terminate a trace, resistors are traditionally added to make the output and/or input match the impedance of the receiver or driver to the impedance of the trace. However, due to increased device I/Os, adding resistors close to the device pins increases the board area and component count, and can in some cases be physically impossible. To address these issues and to achieve better signal integrity, Xilinx developed the Digitally Controlled Impedance (DCI) technology.

DCI adjusts the output impedance or input termination to accurately match the characteristic impedance of the transmission line. DCI actively adjusts the impedance of the I/O to equal an external reference resistance. This compensates for changes in I/O impedance due to process variation. It also continuously adjusts the impedance of the I/O to compensate for variations of temperature and supply voltage fluctuations.

In the case of controlled impedance drivers, DCI controls the driver impedance to match two reference resistors, or optionally, to match half the value of these reference resistors. DCI eliminates the need for external series termination resistors.

DCI provides the parallel or series termination for transmitters or receivers. This eliminates the need for termination resistors on the board, reduces board routing difficulties and component count, and improves signal integrity by eliminating stub reflection. Stub reflection occurs when termination resistors are located too far from the end of the transmission line. With DCI, the termination resistors are as close as possible to the output driver or the input buffer, thus, eliminating stub reflections.

Xilinx DCI

DCI uses two multi-purpose reference pins in each bank to control the impedance of the driver or the parallel termination value for all of the I/Os of that bank. The N reference pin (VRN) must be pulled up to V_{CCO} by a reference resistor, and the P reference pin (VRP) must be pulled down to ground by another reference resistor. The value of each reference resistor should be equal to the characteristic impedance of the PC board traces, or should be twice that value (see section "Driver with Termination to VCCO / 2 (Split Termination)," page 241).

When a DCI I/O standard is used on a particular bank, the two multi-purpose reference pins cannot be used as regular I/Os. However, if DCI I/O standards are not used in the bank, these pins are available as regular I/O pins. The *Virtex-4 Packaging and Pinout Specification* gives detailed pin descriptions.

DCI adjusts the impedance of the I/O by selectively turning transistors in the I/Os on or off. The impedance is adjusted to match the external reference resistors. The impedance adjustment process has two phases. The first phase compensates for process variations by controlling the larger transistors in the I/Os. It occurs during the device startup sequence. The second phase maintains the impedance in response to temperature and supply voltage changes by controlling the smaller transistors in the I/Os. It begins immediately after the first phase and continues indefinitely, even while the device is operating. By default, the DONE pin does not go High until the first phase of the impedance adjustment process is complete.

The coarse impedance calibration during first phase of impedance adjustment can be invoked after configuration by instantiating the DCIRESET primitive. By toggling the RST input to the DCIRESET primitive while the device is operating, the DCI state machine is reset and both phases of impedance adjustment proceed in succession. All I/Os using DCI will be unavailable until the LOCKED output from the DCIRESET block is asserted.

This functionality is useful in applications where the temperature and/or supply voltage changes significantly from device power-up to the nominal operating condition. Once at the nominal operating temperature and voltage, performing the first phase of impedance adjustment allows optimal headroom for the second phase of impedance adjustment.

For controlled impedance output drivers, the impedance can be adjusted either to match the reference resistors or half the resistance of the reference resistors. For on-chip termination, the termination is always adjusted to match the reference resistors.

DCI can configure output drivers to be the following types:

1. Controlled Impedance Driver (Source Termination)
2. Controlled Impedance Driver with Half Impedance (Source Termination)

It can also configure inputs to have the following types of on-chip terminations:

1. Input termination to V_{CCO} (Single Termination)
2. Input termination to $V_{CCO}/2$ (Split Termination, Thevenin equivalent)

For bidirectional operation, both ends of the line can be DCI-terminated permanently:

1. Driver with termination to V_{CCO} (Single Termination)
2. Driver with termination to $V_{CCO}/2$ (Split Termination, Thevenin equivalent)

Alternatively, bidirectional point-to-point lines can use controlled-impedance drivers (with 3-state buffers) on both ends.

Controlled Impedance Driver (Source Termination)

Some I/O standards, such as LVCMOS, must have a drive impedance matching the characteristic impedance of the driven line. DCI can provide controlled impedance output drivers to eliminate reflections without an external source termination. The impedance is set by the external reference resistors with resistance equal to the trace impedance.

The DCI I/O standards supporting the controlled impedance driver are: LVDCI_15, LVDCI_18, LVDCI_25, LVDCI_33, HSLVDCI_15, HSLVDCI_18, HSLVDCI_25, and HSLVDCI_33. Figure 6-4 illustrates a controlled impedance driver in a Virtex-4 device.

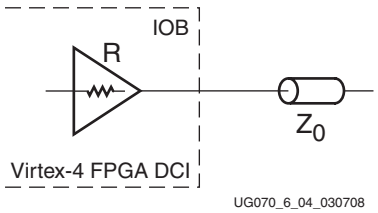


Figure 6-4: **Controlled Impedance Driver**

Controlled Impedance Driver with Half Impedance (Source Termination)

DCI also provides drivers with one half of the impedance of the reference resistors. This doubling of the reference resistor value reduces the static power consumption through

these resistors by a factor of half. The DCI I/O standards supporting controlled impedance drivers with half-impedance are LVDCI_DV2_15, LVDCI_DV2_18, and LVDCI_DV2_25.

Figure 6-5 illustrates a controlled driver with half impedance inside a Virtex-4 device. The reference resistors R must be $2 \times Z_0$ in order to match the impedance of Z_0.

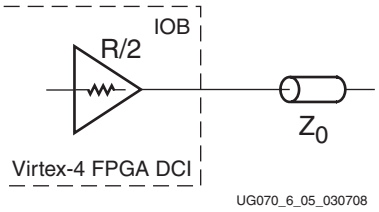


Figure 6-5: **Controlled Impedance Driver with Half Impedance**

Input Termination to V_{CCO} (Single Termination)

Some I/O standards require an input termination to V_{CCO} (see Figure 6-6).

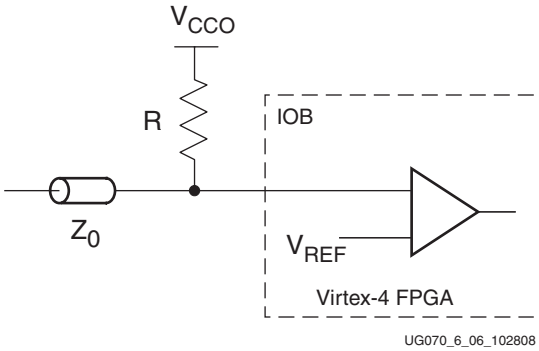


Figure 6-6: **Input Termination to V_{CCO} without DCI**

DCI can also provide input termination to V_{CCO} using single termination. The termination resistance is set by the reference resistors. Both GTL and HSTL standards are controlled by 50Ω reference resistors. The DCI I/O standards supporting single termination are: GTL_DCI, GTLP_DCI, HSTL_III_DCI, HSTL_III_DCI_18, HSTL_IV_DCI, and HSTL_IV_DCI_18.

Figure 6-7 illustrates DCI single termination inside a Virtex-4 device.

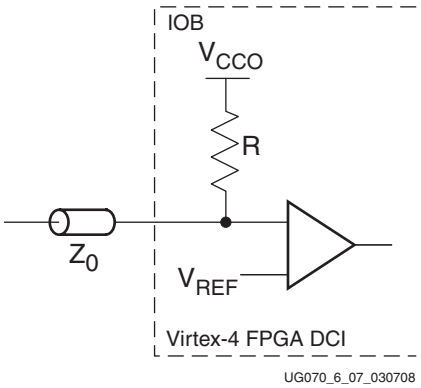


Figure 6-7: **Input Termination Using DCI Single Termination**

Input Termination to $V_{CCO}/2$ (Split Termination)

Some I/O standards (e.g., HSTL Class I and II) require an input termination voltage of $V_{CCO}/2$ (see Figure 6-8).

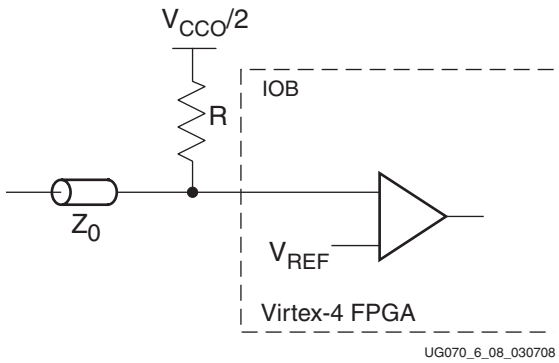


Figure 6-8: **Input Termination to $V_{CCO}/2$ without DCI**

This is equivalent to having a split termination composed of two resistors. One terminates to V_{CCO}, the other to ground. The resistor values are 2R. DCI provides termination to $V_{CCO}/2$ using split termination. The termination resistance is set by the external reference resistors, i.e., the resistors to V_{CCO} and ground are each twice the reference resistor value. Both HSTL and SSTL standards need 50Ω external reference resistors. The DCI I/O standards supporting split termination are: HSTL_I_DCI, HSTL_I_DCI_18, HSTL_II_DCI, HSTL_II_DCI_18, DIFF_HSTL_II_DCI, DIFF_HSTL_II_DCI_18, SSTL2_I_DCI, SSTL2_II_DCI, SSTL18_I_DCI, SSTL18_II_DCI, DIFF_SSTL2_II_DCI, and DIFF_SSTL18_II_DCI.

Figure 6-9 illustrates split termination inside a Virtex-4 device.

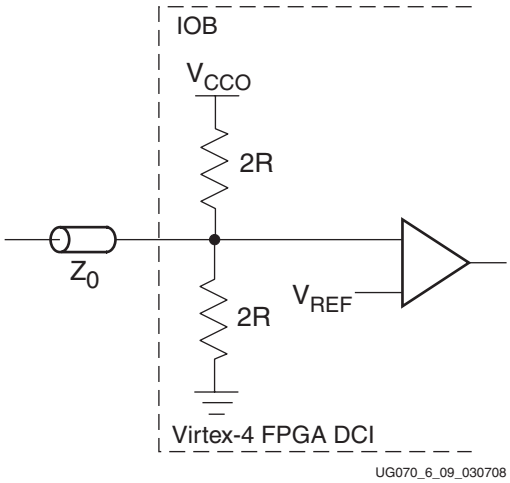


Figure 6-9: **Input Termination to $V_{CCO}/2$ Using DCI Split Termination**

Driver with Termination to V_{CCO} (Single Termination)

Some I/O standards (e.g., HSTL Class IV) require an output termination to V_{CCO}. Figure 6-10 illustrates an output termination to V_{CCO}.

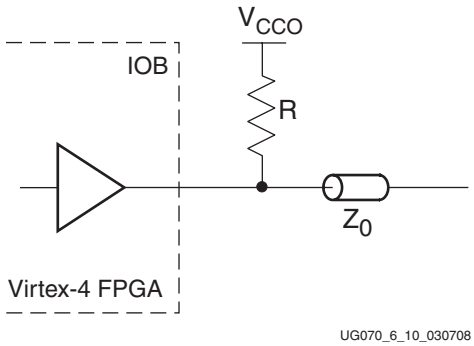


Figure 6-10: **Driver with Termination to V_{CCO} without DCI**

DCI can provide an output termination to V_{CCO} using single termination. In this case, DCI only controls the impedance of the termination, but not the driver. Both GTL and HSTL standards need 50Ω external reference resistors. The DCI I/O standards supporting drivers with single termination are: GTL_DCI, GTLP_DCI, HSTL_IV_DCI, and HSTL_IV_DCI_18.

Figure 6-11 illustrates a driver with single termination inside a Virtex-4 device.

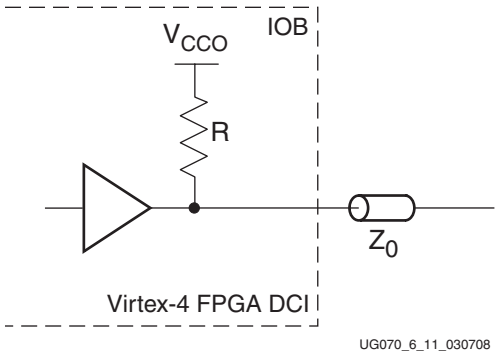


Figure 6-11: **Driver with Termination to V_{CCO} Using DCI Single Termination**

Driver with Termination to $V_{CCO}/2$ (Split Termination)

Some I/O standards, such as HSTL Class II, require an output termination to $V_{CCO}/2$ (see Figure 6-12).

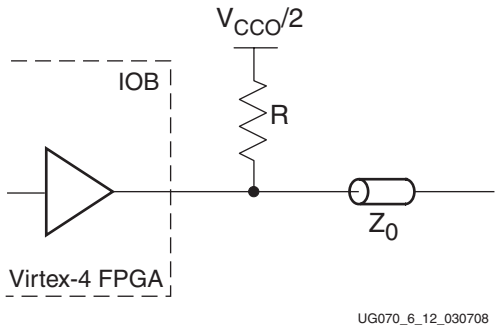


Figure 6-12: **Driver with Termination to $V_{CCO}/2$ without DCI**

DCI can provide output termination to $V_{CCO}/2$ using split termination. DCI only controls the impedance of the termination, but not the driver. Both HSTL and SSTL standards need 50Ω external reference resistors. The DCI I/O standards supporting drivers with split termination are: HSTL_II_DCI, HSTL_II_DCI_18, SSTL2_II_DCI, SSTL18_II_DCI, DIFF_HSTL_II_DCI, DIFF_HSTL_II_DCI_18, DIFF_SSTL2_II_DCI, and DIFF_SSTL18_II_DCI.

Figure 6-13 illustrates a driver with split termination inside a Virtex-4 device.

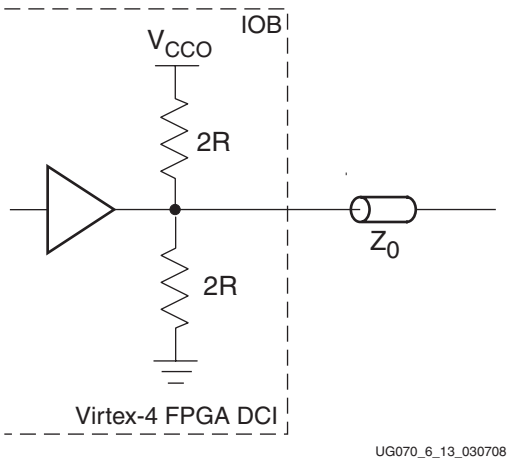


Figure 6-13: **Driver with Termination to $V_{CCO}/2$ Using DCI Split Termination**

DCI in Virtex-4 FPGA Hardware

DCI works with single-ended I/O standards and the 2.5V LVDS I/O standard. DCI supports the following Virtex-4 FPGA standards:

LVDCI, HSLVDCI, LVDCI_DV2, GTL_DCI, GTLP_DCI, HSTL_I_DCI, HSTL_II_DCI, HSTL_III_DCI, HSTL_IV_DCI, HSTL_I_DCI_18, HSTL_II_DCI_18, HSTL_III_DCI_18, HSTL_IV_DCI_18, SSTL2_I_DCI, SSTL2_II_DCI, SSTL18_I_DCI, SSTL18_II_DCI, DIFF_HSTL_II_DCI, DIFF_HSTL_II_DCI_18, DIFF_SSTL2_II_DCI, DIFF_SSTL18_II_DCI, LVDS_25_DCI, and LVDSEXT_25_DCI.

To correctly use DCI in a Virtex-4 device, users must follow the following rules:

1. V_{CCO} pins must be connected to the appropriate V_{CCO} voltage based on the IOSTANDARDs in that bank.
2. Correct DCI I/O buffers must be used in the software either by using IOSTANDARD attributes or instantiations in the HDL code.
3. Some DCI standards require that external reference resistors be connected to multipurpose pins VRP and VRN in the bank. Where this is required, these two multipurpose pins cannot be used as regular user I/Os. Refer to the Virtex-4 FPGA pinouts for the specific pin locations. Pin VRN must be pulled up to V_{CCO} by its reference resistor. Pin VRP must be pulled down to ground by its reference resistor.

   However, some DCI standards do not require external reference resistors on the VRP/VRN pins. If these are the only DCI-based I/O standards in a bank, the VRP and VRN pins in that bank *can be used* as general-purpose I/Os.

   - The following DCI *outputs* do *not* require reference resistors on VRP/VRN:

     HSTL_I_DCI
     HSTL_III_DCI
     HSTL_I_DCI_18
     HSTL_III_DCI_18
     SSTL2_I_DCI
     SSTL18_I_DCI

   - The following *inputs* do *not* require reference resistors on VRP/VRN:

     LVDCI_15
     LVDCI_18
     LVDCI_25
     LVDCI_33
     LVDCI_DV2_15
     LVDCI_DV2_18
     LVDCI_DV2_25
     LVDCI_DV2_33
4. The value of the external reference resistors should be selected to give the desired output impedance. If using GTL_DCI, HSTL_DCI, or SSTL_DCI I/O standards, then the external reference resistors should be 50Ω.
5. The values of the reference resistors must be within the supported range (20Ω – 100Ω).
6. Follow the DCI I/O banking rules:
   a. V_{REF} must be compatible for all of the inputs in the same bank.
   b. V_{CCO} must be compatible for all of the inputs and outputs in the same bank.
   c. No more than one DCI I/O standard using single termination type is allowed per bank.
   d. No more than one DCI I/O standard using split termination type is allowed per bank.
   e. Single termination and split termination, controlled impedance driver, and controlled impedance driver with half impedance can co-exist in the same bank.
7. The following packages to not support DCI in Banks 1 and 2: SF363, FF668, FF676, FF672, and FF1152.
8. In addition, the following devices do not support DCI in Banks 1 and 2: XC4VLX15, XC4VLX25, XC4VSX25, XC4VSX35, XC4VFX12, XC4VFX20, XC4VFX40, and XC4VFX60.

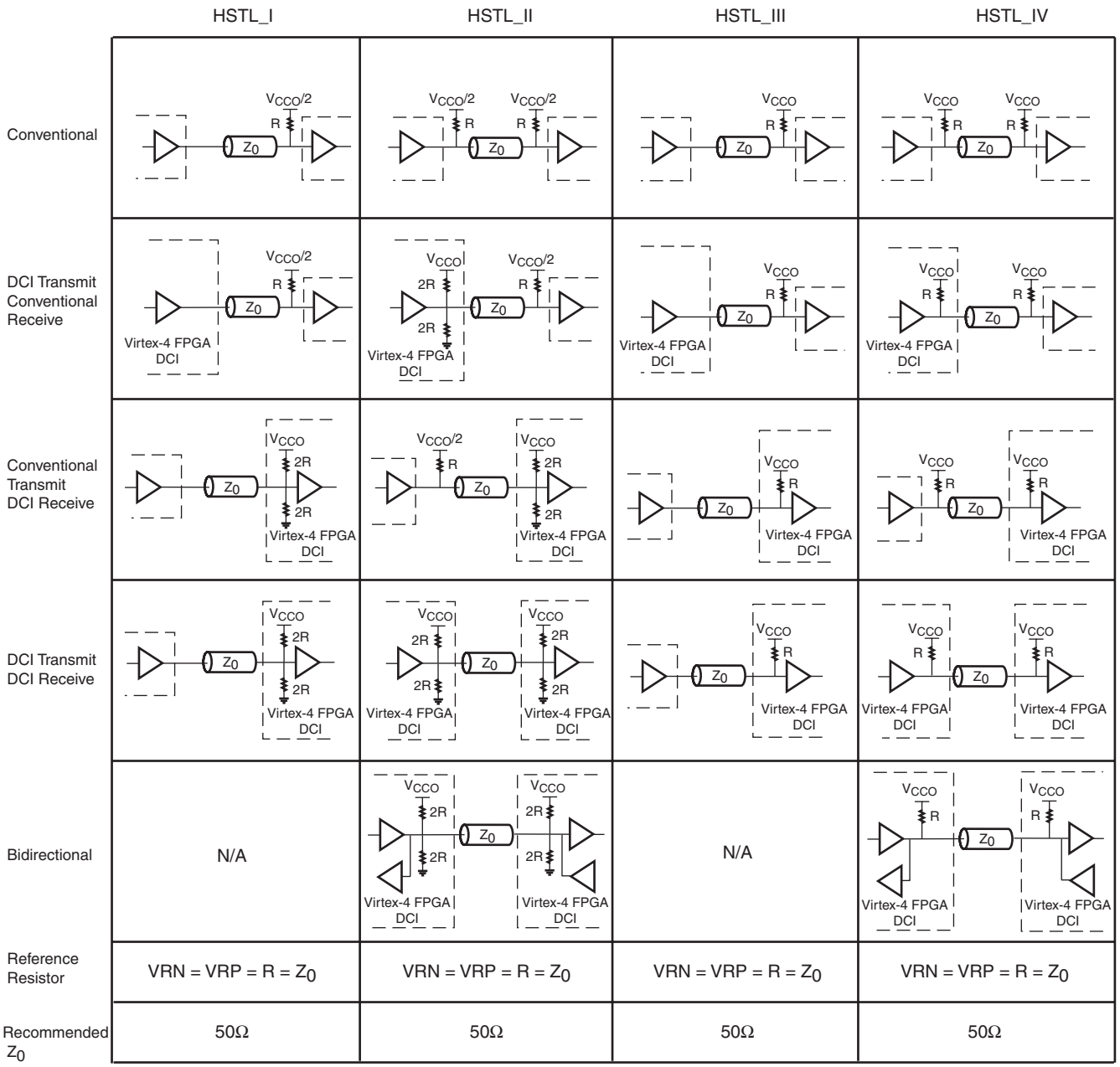
The behavior of a DCI 3-state outputs is as follows:

If a LVDCI or LVDCI_DV2 driver is in 3-state, the driver is 3-stated. If a driver with single or split termination is in 3-state, the driver is 3-stated but the termination resistor remains.

The following section lists actions that must be taken for each DCI I/O standard.

DCI Usage Examples

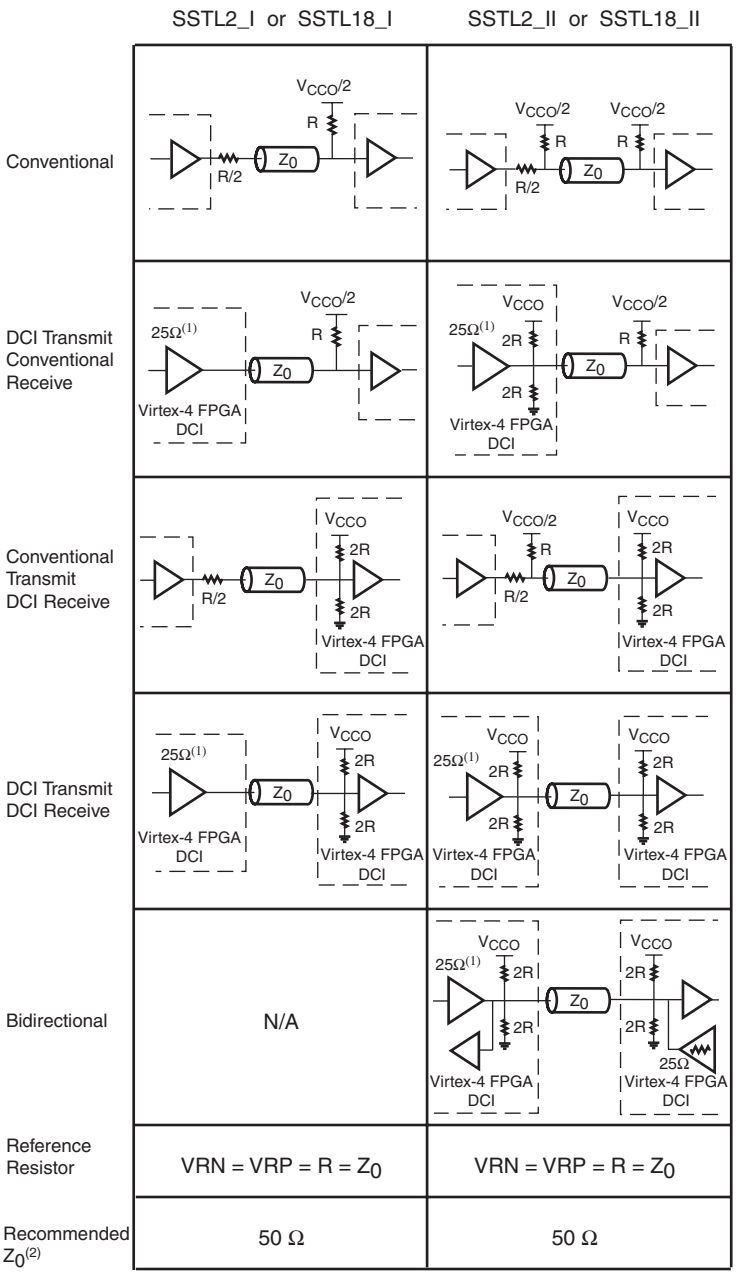
- Figure 6-14 provides examples illustrating the use of the HSTL_I_DCI, HSTL_II_DCI, HSTL_III_DCI, and HSTL_IV_DCI I/O standards.
- Figure 6-15 provides examples illustrating the use of the SSTL2_I_DCI and SSTL2_II_DCI I/O standards.
- Figure 6-16 provides examples illustrating the use of the LVDS_25_DCI and LVDSEXT_25_DCI I/O standards.



Notes:
1. Z_0 is the recommended PCB trace impedance.

UG070_6_14_031108

Figure 6-14: **HSTL DCI Usage Examples**



Notes:
1. The SSTL-compatible 25Ω series resistor is accounted for in the DCI buffer, and it is not DCI controlled.
2. Z_0 is the recommended PCB trace impedance.

Figure 6-15: **SSTL DCI Usage Examples**

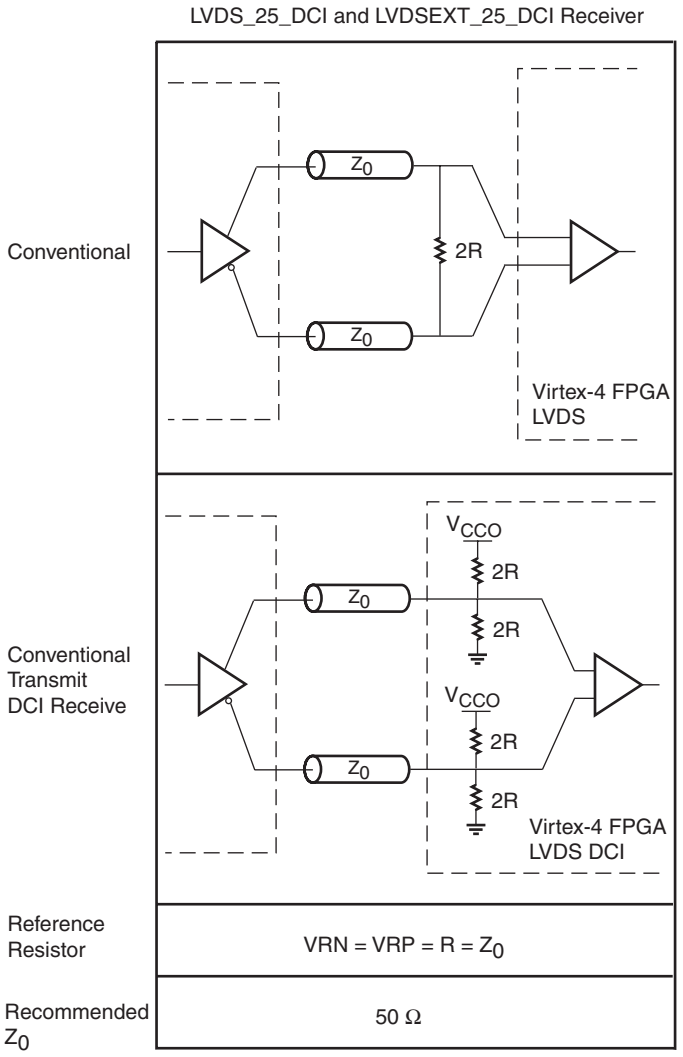


Figure 6-16: **LVDS DCI Usage Examples**

Virtex-4 FPGA SelectIO Primitives

The Xilinx software library includes an extensive list of primitives to support a variety of I/O standards available in the Virtex-4 FPGA I/O primitives. The following are five generic primitive names representing most of the available single-ended I/O standards.

- IBUF (input buffer)
- IBUFG (clock input buffer)
- OBUF (output buffer)
- OBUFT (3-state output buffer)
- IOBUF (input/output buffer)

These five generic primitive names represent most of the available differential I/O standards:

- IBUFDS (input buffer)
- IBUFGDS (clock input buffer)
- OBUFDS (output buffer)
- OBUFTDS (3-state output buffer)
- IOBUFDS (input/output buffer)

IBUF and IBUFG

Signals used as inputs to Virtex-4 devices must use an input buffer (IBUF). The generic Virtex-4 FPGA IBUF primitive is shown in Figure 6-17.

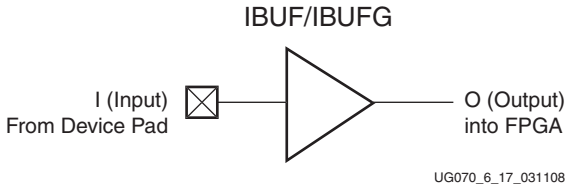


Figure 6-17: **Input Buffer (IBUF/IBUFG) Primitives**

The IBUF and IBUFG primitives are the same. IBUFGs are used when an input buffer is used as a clock input. In the Xilinx software tools, an IBUFG is automatically placed at clock input sites.

OBUF

An output buffer (OBUF) must be used to drive signals from Virtex-4 devices to external output pads. A generic Virtex-4 FPGA OBUF primitive is shown in Figure 6-18.

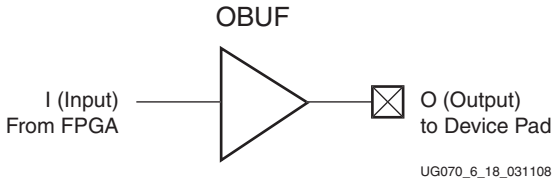


Figure 6-18: **Output Buffer (OBUF) Primitive**

OBUFT

The generic 3-state output buffer OBUFT, shown in Figure 6-19, typically implements 3-state outputs or bidirectional I/O.

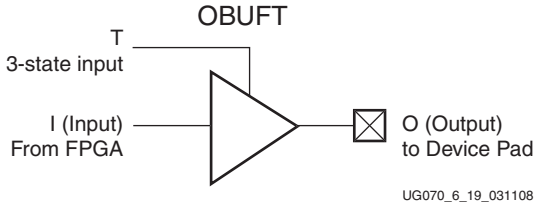


Figure 6-19: **3-State Output Buffer (OBUFT) Primitive**

IOBUF

The IOBUF primitive is needed when bidirectional signals require both an input buffer and a 3-state output buffer with an active High 3-state pin. Figure 6-20 shows a generic Virtex-4 FPGA IOBUF.

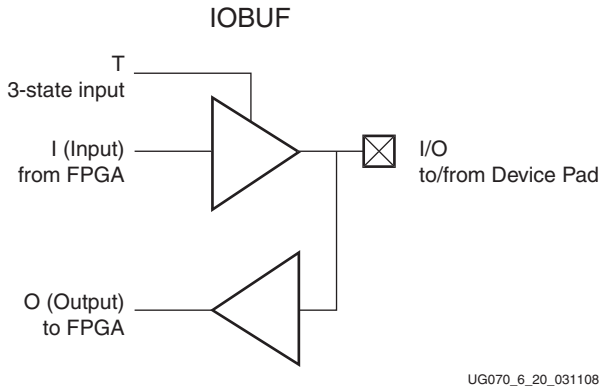


Figure 6-20: **Input/Output Buffer (IOBUF) Primitive**

IBUFDS and IBUFGDS

The usage and rules corresponding to the differential primitives are similar to the single-ended SelectIO primitives. Differential SelectIO primitives have two pins to and from the device pads to show the P and N channel pins in a differential pair. N channel pins have a "B" suffix.

Figure 6-21 shows the differential input buffer primitive.

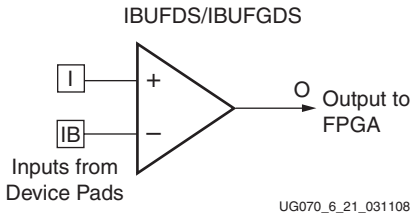


Figure 6-21: **Differential Input Buffer Primitive (IBUFDS/IBUFGDS)**

OBUFDS

Figure 6-22 shows the differential output buffer primitive.

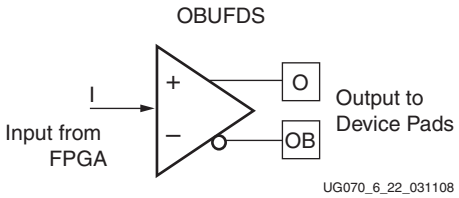


Figure 6-22: **Differential Output Buffer Primitive (OBUFDS)**

OBUFTDS

Figure 6-23 shows the differential 3-state output buffer primitive.

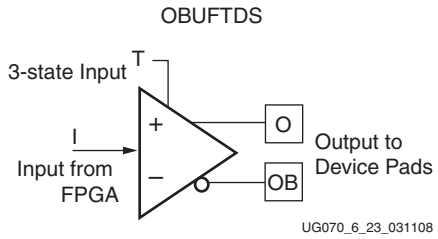


Figure 6-23: **Differential 3-state Output Buffer Primitive (OBUFTDS)**

IOBUFDS

Figure 6-24 shows the differential input/output buffer primitive.

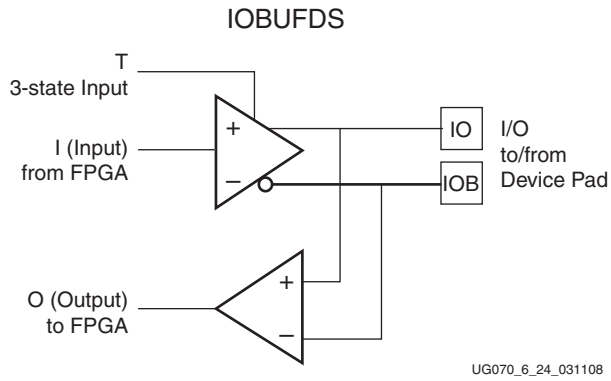


Figure 6-24: **Differential Input/Output Buffer Primitive (IOBUFDS)**

Virtex-4 FPGA SelectIO Attributes/Constraints

Access to some Virtex-4 FPGA I/O resource features (e.g., location constraints, input delay, output drive strength, and slew rate) is available through the attributes/constraints associated with these features. For more information a Constraints Guide is available on the Xilinx website with syntax examples and VHDL/Verilog reference code. This guide is available inside the Software Manuals at:
http://www.support.xilinx.com/support/software_manuals.htm

Location Constraints

The location constraint (LOC) must be used to specify the I/O location of an instantiated I/O primitive. The possible values for the location constraint are all the external port identifiers (e.g., A8, M5, AM6, etc.). These values are device and package size dependent.

The LOC attribute uses the following syntax in the UCF file:

```
INST <I/O_BUFFER_INSTANTIATION_NAME> LOC =
"<EXTERNAL_PORT_IDENTIFIER>";
```

Example:

```
INST MY_IO LOC=R7;
```

IOStandard Attribute

The IOSTANDARD attribute is available to choose the values for an I/O standard for all I/O buffers. The supported I/O standards are listed in Table 6-38. The IOSTANDARD attribute uses the following syntax in the UCF file:

```
INST <I/O_BUFFER_INSTANTIATION_NAME> IOSTANDARD="<IOSTANDARD VALUE>";
```

The IOSTANDARD default for single-ended I/O is LVCMOS25, for differential I/Os the default is LVDS_25.

Output Slew Rate Attributes

A variety of attribute values provide the option of choosing the desired slew rate for single-ended I/O output buffers. For LVTTL and LVCMOS output buffers (OBUF, OBUFT, and IOBUF), the desired slew rate can be specified with the SLEW attribute.

The allowed values for the SLEW attribute are:

- SLEW = SLOW (Default)
- SLEW = FAST

The SLEW attribute uses the following syntax in the UCF file:

```
INST <I/O_BUFFER_INSTANTIATION_NAME> SLEW = "<SLEW_VALUE>";
```

By the default, the slew rate for each output buffer is set to SLOW. This is the default used to minimize the power bus transients when switching non-critical signals.

Output Drive Strength Attributes

For LVTTL and LVCMOS output buffers (OBUF, OBUFT, and IOBUF), the desired drive strength (in mA) can be specified with the DRIVE attribute.

The allowed values for the DRIVE attribute are:

- DRIVE = 2
- DRIVE = 4
- DRIVE = 6
- DRIVE = 8
- DRIVE = 12 (Default)
- DRIVE = 16
- DRIVE = 24

The DRIVE attribute uses the following syntax in the UCF file:

```
INST <I/O_BUFFER_INSTANTIATION_NAME> DRIVE = "<DRIVE_VALUE>";
```

Lower Capacitance I/O Attributes

To lower the effective input capacitance, some I/O resources do not have differential driver circuits (LVDS_25, LVDSEXT_25, LVDS_25_DCI, LVDSEXT_25_DCI, ULVDS_25, RSDS_25, and LDT_25). Using these I/Os improves the signal integrity of high-speed clock inputs. Differential inputs and all output standards other than these are still supported by low capacitance I/Os. Refer to "Clock Capable I/O" in Chapter 1 for further information.

The allowed values for the CAPACITANCE attribute are:

- DONT_CARE (Default)

- NORMAL
- LOW

The CAPACITANCE attribute uses the following syntax in the UCF file:

```
INST <I/O_BUFFER_INSTANTIATION_NAME> CAPACITANCE=
"<CAPACITANCE_VALUE>";
```

PULLUP/PULLDOWN/KEEPER for IBUF, OBUFT, and IOBUF

When using 3-state output (OBUFT) or bidirectional (IOBUF) buffers, the output can have a weak pull-up, a weak pull-down, or a weak keeper circuit. For input (IBUF) buffers, the input can have either a weak pull-up or a weak pull-down circuit. These features can be invoked by adding one of the following possible constraint values to the buffer:

- PULLUP
- PULLDOWN
- KEEPER

Xilinx recommends that these internal termination circuits (weak pull-ups, weak pull-downs and weak keepers) *not* be used to hold a logic level for a 3-stated signal. It is highly likely that coupled noise from a PCB trace would swamp out the effect of these termination circuits. The intended application of the internal termination circuits is to hold logic values for *unconnected pins* to prevent spurious switching and consequent power loss. Internal termination circuits are not intended to drive board-level traces to a defined logic level.

Differential Termination Attribute

The differential termination (DIFF_TERM) attribute is designed for the Virtex-4 FPGA supported differential input I/O standards. It is used to turn the built-in 100Ω differential termination on or off.

The allowed values for the DIFF_TERM attribute are:

- TRUE
- FALSE (Default)

To specify the DIFF_TERM attribute, set the appropriate value in the generic map (VHDL) or inline parameter (Verilog) of the instantiated IBUFDS or IBUFGDS component. Please refer to the ISE® software Language Templates or the Virtex-4 FPGA HDL Libraries Guide for the proper syntax for instantiating this component and setting the DIFF_TERM attribute.

Virtex-4 FPGA I/O Resource VHDL/Verilog Examples

The following examples are VHDL and Verilog syntaxes to declare a standard for Virtex-4 FPGA I/O resources. The example uses IOBUF.

VHDL Template

```
--Example IOBUF component declaration

component IOBUF
    generic(
      CAPACITANCE : string     := "DONT_CARE";
      DRIVE       : integer    := 12;
      IOSTANDARD  : string     := "LVCMOS25";
```

```
      SLEW        : string     := "SLOW"
      );

    port(
      O  : out   std_ulogic;
      IO : inout std_ulogic;
      I  : in    std_ulogic;
      T  : in    std_ulogic
      );
  end component;

--Example IOBUF instantiation

U_IOBUF : IOBUF
Port map(
O => user_o,
IO => user_io,
I => user_i,
T => user_t
);
```

Verilog Template

```
//Example IOBUF module declaration

module IOBUF (O, IO, I, T);

    parameter CAPACITANCE = "DONT_CARE";
    parameter DRIVE = 12;
    parameter IOSTANDARD = "LVCMOS25";
    parameter SLEW = "SLOW";

    output O;
    inout  IO;
    input  I, T;

    tri0 GTS = glbl.GTS;

    or O1 (ts, GTS, T);
    bufif0 T1 (IO, I, ts);

    buf B1 (O, IO);

endmodule

//Example IOBUF instantiation

IOBUF U_IOBUF (
.O(user_o),
.IO(user_io),
.I(user_i),
.T(user_t));
```

Specific Guidelines for Virtex-4 FPGA I/O Supported Standards

The following sections provide an overview of the I/O standards supported by all Virtex-4 devices.

While most Virtex-4 FPGA I/O supported standards specify a range of allowed voltages, this chapter records typical voltage values only. Detailed information on each specification can be found on the Electronic Industry Alliance JEDEC website at http://www.jedec.org.

LVTTL (Low Voltage Transistor-Transistor Logic)

The low-voltage TTL (LVTTL) standard is a general purpose EIA/JESDSA standard for 3.3V applications using an LVTTL input buffer and a push-pull output buffer. This standard requires a 3.3V input and output supply voltage (V_{CCO}), but does not require the use of a reference voltage (V_{REF}) or a termination voltage (V_{TT}).

Sample circuits illustrating both unidirectional and bidirectional LVTTL termination techniques are shown in Figure 6-25 and Figure 6-26.

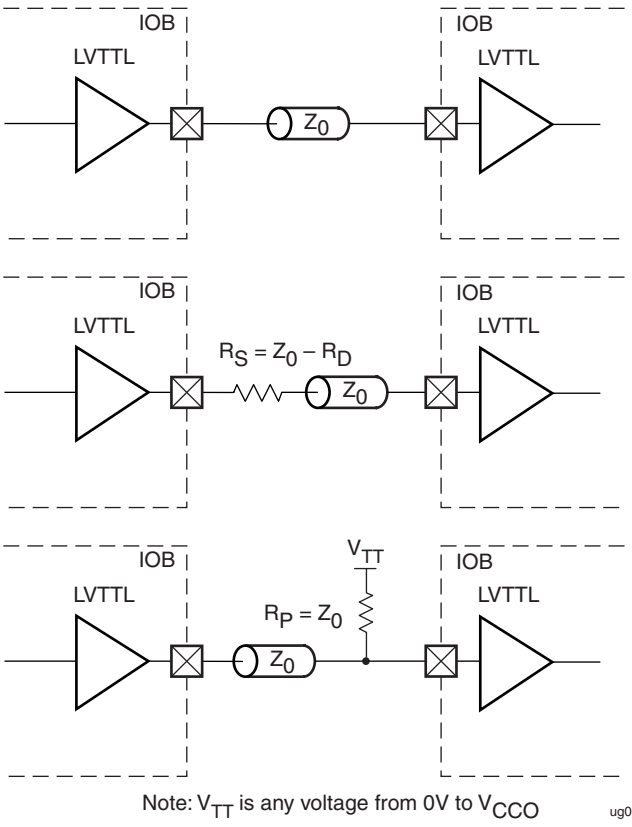


Figure 6-25: **LVTTL Unidirectional Termination**

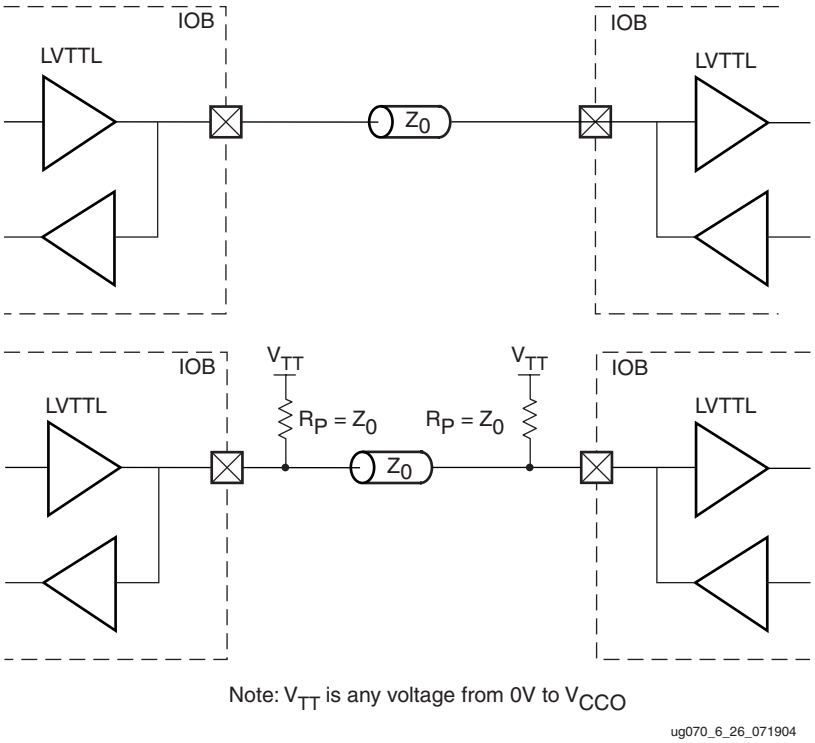


Figure 6-26: **LVTTL Bidirectional Termination**

Table 6-1 lists the LVTTL DC voltage specifications.

Table 6-1: **LVTTL DC Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.45
V_{REF}	–	–	–
V_{TT}	–	–	–
V_{IH}	2.0	–	3.45
V_{IL}	–0.2	–	0.8
V_{OH}	2.4	–	–
V_{OL}	–	–	0.4
I_{OH} at V_{OH} (mA)	Note (2)	–	–
I_{OL} at V_{OL} (mA)	Note (2)	-	–

Notes:

1. V_{OL} and V_{OH} for lower drive currents are sample tested.
2. Supported DRIVE strengths are 2/4/6/8/12 /16/24 mA.

Table 6-2 details the allowed attributes that can be applied to the LVTTL I/O standard.

Table 6-2: **Allowed Attributes for the LVTTL I/O Standard**

Attributes	Primitives		
	IBUF/IBUFG	OBUF/OBUFT	IOBUF
IOSTANDARD	LVTTL	LVTTL	LVTTL
CAPACITANCE	LOW, NORMAL, DONT_CARE		
DRIVE	UNUSED	2, 4, 6, 8, 12, 16, 24	2, 4, 6, 8, 12, 16, 24
SLEW	UNUSED	{FAST, SLOW}	{FAST, SLOW}

LVCMOS (Low Voltage Complementary Metal Oxide Semiconductor)

LVCMOS is a widely used switching standard implemented in CMOS transistors. This standard is defined by JEDEC (JESD 8-5).

Sample circuits illustrating both unidirectional and bidirectional LVCMOS termination techniques are shown in Figure 6-27 and Figure 6-28.

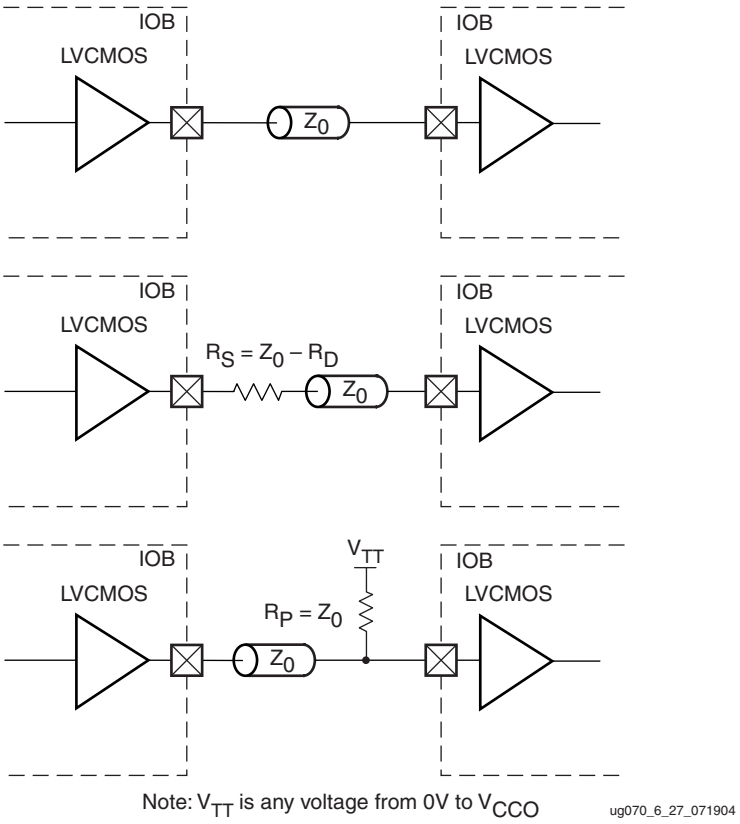


Figure 6-27: **LVCMOS Unidirectional Termination**

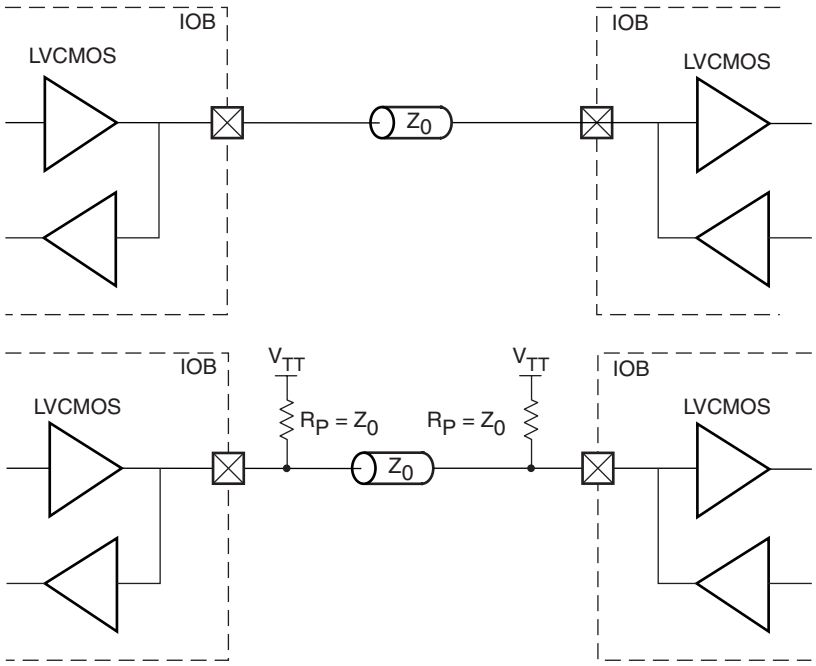


Figure 6-28: **LVCMOS Bidirectional Termination**

Table 6-3 details the allowed attributes that can be applied to the LVCMOS33 and LVCMOS25 I/O standards.

Table 6-3: **Allowed Attributes for the LVCMOS33 and LVCMOS25 I/O Standards**

Attributes	Primitives		
	IBUF/IBUFG	OBUF/OBUFT	IOBUF
IOSTANDARD	LVCMOS33 LVCMOS25	LVCMOS33 LVCMOS25	LVCMOS33 LVCMOS25
CAPACITANCE	LOW, NORMAL, DONT_CARE		
DRIVE	UNUSED	2, 4, 6, 8, 12, 16, 24	2, 4, 6, 8, 12, 16, 24
SLEW	UNUSED	{FAST, SLOW}	{FAST, SLOW}

Table 6-4 details the allowed attributes that can be applied to the LVCMOS18 and LVCMOS15 I/O standards.

Table 6-4: **Allowed Attributes for the LVCMOS18 and LVCMOS15 I/O Standard**

Attributes	Primitives		
	IBUF/IBUFG	OBUF/OBUFT	IOBUF
IOSTANDARD	LVCMOS18 LVCMOS15	LVCMOS18 LVCMOS15	LVCMOS18 LVCMOS15
CAPACITANCE	LOW, NORMAL, DONT_CARE		
DRIVE	UNUSED	2, 4, 6, 8, 12, 16	2, 4, 6, 8, 12, 16
SLEW	UNUSED	{FAST, SLOW}	{FAST, SLOW}

LVDCI (Low Voltage Digitally Controlled Impedance)

Using these I/O buffers configures the outputs as controlled impedance drivers. The receiver of LVDCI is identical to a LVCMOS receiver. Some I/O standards, such as LVTTL, LVCMOS, etc., must have a drive impedance that matches the characteristic impedance of the driven line. Virtex-4 devices provide a controlled impedance output driver to provide series termination without external source termination resistors. The impedance is set by the common external reference resistors, with resistance equal to the trace characteristic impedance, Z_0.

Sample circuits illustrating both unidirectional and bidirectional termination techniques for a controlled impedance driver are shown in Figure 6-29 and Figure 6-30. The DCI I/O standards supporting a controlled impedance driver are: LVDCI_15, LVDCI_18, LVDCI_25, and LVDCI_33.

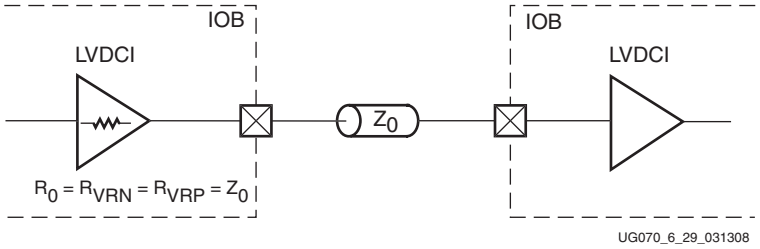


Figure 6-29: **Controlled Impedance Driver with Unidirectional Termination**

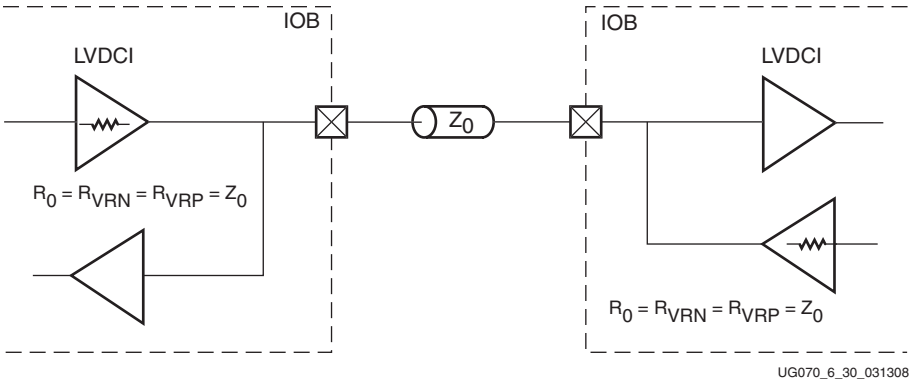


Figure 6-30: **Controlled Impedance Driver with Bidirectional Termination**

LVDCI_DV2

A controlled impedance driver with half impedance (source termination) can also provide drivers with one half of the impedance of the reference resistors. The I/O standards supporting a controlled impedance driver with half impedance are: LVDCI_DV2_15, LVDCI_DV2_18, and LVDCI_DV2_25. Figure 6-31 and Figure 6-32 illustrate a controlled driver with half impedance unidirectional and bidirectional termination.

To match the drive impedance to Z_0 when using a driver with half impedance, the reference resistor R must be twice Z_0.

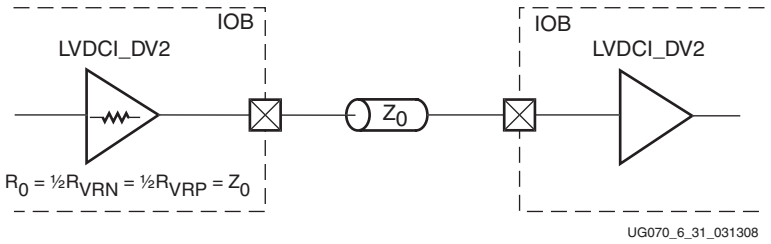


Figure 6-31: **Controlled Impedance Driver with Half Impedance Unidirectional Termination**

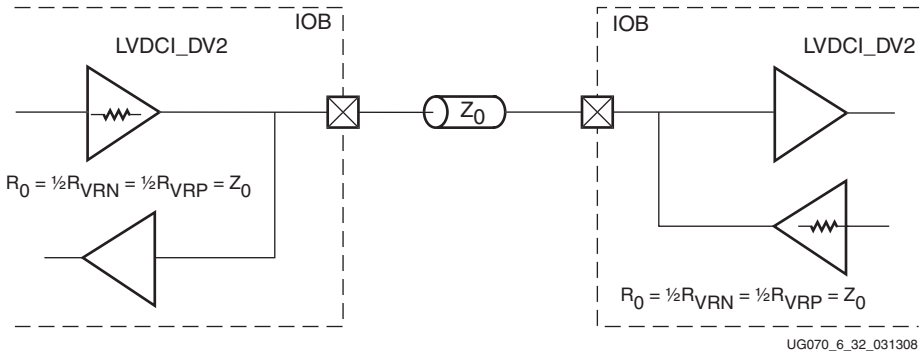


Figure 6-32: **Controlled Impedance Driver with Half Impedance Bidirectional Termination**

There are no drive strength settings for LVDCI drivers. When the driver impedance is one-half of the VRN/VRP reference resistors, it is indicated by the addition of DV2 to the attribute name.

Table 6-5 lists the LVCMOS, LVDCI, and LVDCI_DV2 voltage specifications.

Table 6-5: **LVCMOS, LVDCI, and LVDCI_DV2 DC Voltage Specifications at Various Voltage References**

Standard	+3.3V			+2.5V			+1.8V			+1.5V		
	Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	Min	Typ	Max
V_{CCO} [V]	3.0	3.3	3.6	2.3	2.5	2.7	1.7	1.8	1.9	1.43	1.5	1.57
V_{IH} [V]	2.0	–	3.6	1.7	–	2.7	1.19	–	1.95	1.05	–	1.65
V_{IL} [V]	–0.5	–	0.8	–0.5	–	0.7	–0.5	–	0.4	–0.5	–	0.3
V_{OH} [V]	2.6	–	–	1.9	–	–	1.3	–	–	–	1.05	–
V_{OL} [V]	–	–	0.4	–	–	0.4	–	–	0.4	–	–	0.4
I_{IN} [µA]	–	± 5	–	–	± 5	–	–	± 5	–	–	± 5	–

Notes: V_{OL} and V_{OH} for lower drive currents are sample tested.

HSLVDCI (High-Speed Low Voltage Digitally Controlled Impedance)

The HSLVDCI standard is intended for bidirectional use. The driver is identical to LVDCI, while the input is identical to HSTL and SSTL. By using a V_{REF}-referenced input, HSLVDCI allows greater input sensitivity at the receiver than when using a single-ended LVCMOS-type receiver.

Sample circuits illustrating both unidirectional and bidirectional termination techniques for an HSLVDCI controlled impedance driver are shown in Figure 6-29 and Figure 6-30. The DCI I/O standards supporting a controlled impedance driver with a V_{REF} referenced input are: HSLVDCI_15, HSLVDCI_18, HSLVDCI_25, and HSLVDCI_33.

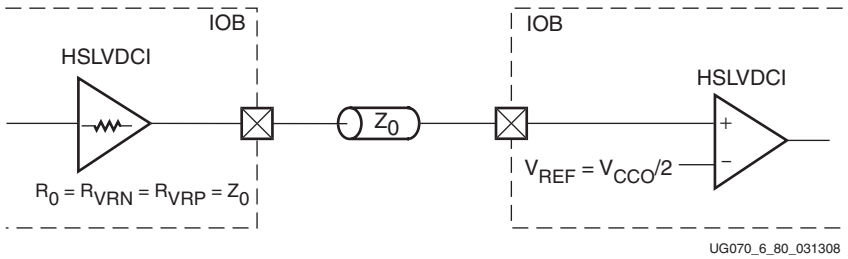


Figure 6-33: **HSLVDCI Controlled Impedance Driver with Unidirectional Termination**

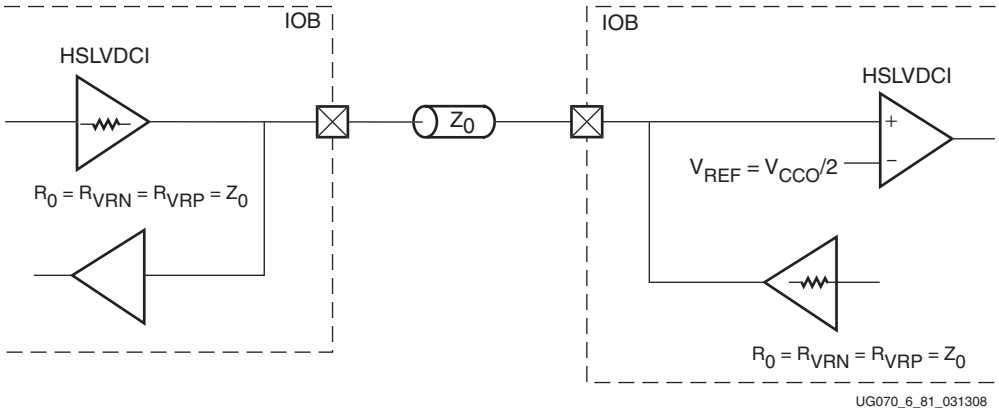


Figure 6-34: **HSLVDCI Controlled Impedance Driver with Bidirectional Termination**

For output DC voltage specifications, refer to the LVDCI V_{OH} and V_{OL} entries in Table 6-5. Table 6-6 lists the input DC voltage specifications when using HSLVDCI. Valid values of V_{CCO} are 1.5V, 1.8V, 2.5V, and 3.3V. Select V_{REF} to provide the optimum noise margin in specific use conditions.

Table 6-6: **HSLVDCI Input DC Voltage Specifications**

Standard	Min	Typ	Max
V_{REF}	–	$V_{CCO}/2$	–
V_{IH}	V_{REF} + 0.1	–	–
V_{IL}	–	–	V_{REF} – 0.1

Table 6-7 details the allowed attributes that can be applied to the LVDCI, HSLVDCI, and LVDCI_DV2 I/O standards.

Table 6-7: **Allowed Attributes of the LVDCI, HSLVDCI, and LVDCI_DV2 I/O Standards**

Attributes	Primitives		
	IBUF/IBUFG	OBUF/OBUFT	IOBUF
IOSTANDARD	LVDCI_15, LVDCI_18, LVDCI_25, LVDCI_33 LVDCI_DV2_15, LVDCI_DV2_18, LVDCI_DV2_25, HSLVDCI_15, HSLVDCI_18, HSLVDCI_25, HSLVDCI_33		
CAPACITANCE	LOW, NORMAL, DONT_CARE		

PCIX, PCI33, PCI66 (Peripheral Component Interface)

The PCI standard specifies support for 33 MHz, 66 MHz, and 133 MHz PCI bus applications. It uses an LVTTL input buffer and a push-pull output buffer. This standard does not require the use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}). However, it does require 3.3V input/output source voltage (V_{CCO}).

A PCI undershoot/overshoot specification could require V_{CCO} to be regulated at 3.0V as discussed in "Regulating VCCO at 3.0V," page 306. This is not necessary if overshoot and undershoot are controlled by careful design.

Table 6-8 lists the DC voltage specifications.

Table 6-8: **PCI33_3, PCI66_3, and PCIX DC Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.5
V_{REF}	–	–	–
V_{TT}	–	–	–
$V_{IH} = 0.5 \times V_{CCO}$	1.5	1.65	$V_{CCO} + 0.5$
$V_{IL} = 0.3 \times V_{CCO}$	–0.5	0.99	1.08
$V_{OH} = 0.9 \times V_{CCO}$	2.7	–	–
$V_{OL} = 0.1 \times V_{CCO}$	–	–	0.36
I_{OH} at V_{OH} (mA)	(Note 1)	–	–
I_{OL} at V_{OL} (mA)	(Note 1)	–	–

Notes:

1. Tested according to the relevant specification.

Table 6-9 details the allowed attributes that can also be applied to the PCI33_3, PCI66_3, and PCIX I/O standards.

Table 6-9: **Allowed Attributes of the PCI33_3, PCI66_3, and PCIX I/O Standards**

Attributes	Primitives		
	IBUF/IBUFG	OBUF/OBUFT	IOBUF
IOSTANDARD	PCI33_3, PCI66_3, and PCIX		
CAPACITANCE	LOW, NORMAL, DONT_CARE		

GTL (Gunning Transceiver Logic)

The Gunning Transceiver Logic (GTL) standard is a high-speed bus standard (JESD8.3) invented by Xerox. Xilinx has implemented the terminated variation for this standard. This standard requires a differential amplifier input buffer and an open-drain output buffer. The negative terminal of the differential input buffer is referenced to the V_{REF} pin.

A sample circuit illustrating a valid termination technique for GTL with external parallel termination and unconnected V_{CCO} is shown in Figure 6-35.

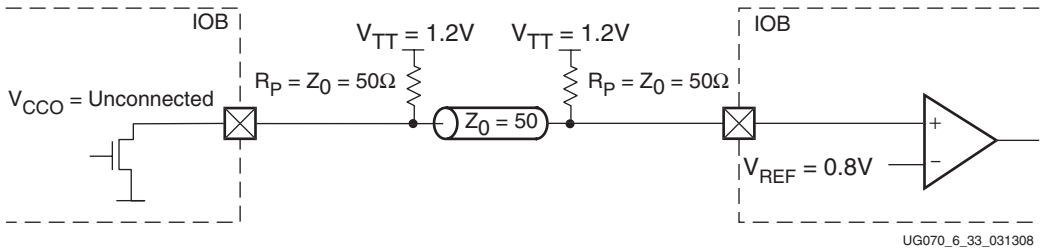


Figure 6-35: **GTL with External Parallel Termination and Unconnected V_{CCO}**

GTL_DCI Usage

GTL does not require a V_{CCO} voltage. However, for GTL_DCI, V_{CCO} must be connected to 1.2V. GTL_DCI provides single termination to V_{CCO} for inputs or outputs.

A sample circuit illustrating a valid termination technique for GTL_DCI with internal parallel driver and receiver termination is shown in Figure 6-36.

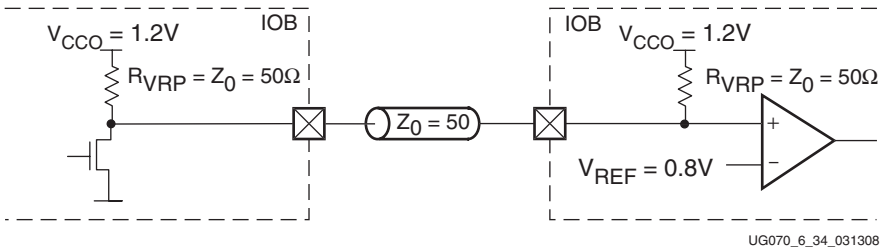


Figure 6-36: **GTL_DCI with Internal Parallel Driver and Receiver Termination**

Table 6-10 lists the GTL DC voltage specifications.

Table 6-10: **GTL DC Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	–	N/A	-
$V_{REF} = N \times V_{TT}$ (1)	0.74	0.8	0.86
V_{TT}	1.14	1.2	1.26
$V_{IH} = V_{REF} + 0.05$	0.79	0.85	–
$V_{IL} = V_{REF} - 0.05$	–	0.75	0.81
V_{OH}	–	–	–
V_{OL}	–	0.2	0.4
I_{OH} at V_{OH} (mA)	–	–	–
I_{OL} at V_{OL} (mA) at 0.4V	32	–	–

Table 6-10: **GTL DC Voltage Specifications *(Continued)***

Parameter	Min	Typ	Max
I_{OL} at V_{OL} (mA) at 0.2V	-	–	40

Notes:

1. N must be greater than or equal to 0.653 and less than or equal to 0.68.

Table 6-11 details the allowed attributes that can also be applied to the GTL I/O standards.

Table 6-11: **Allowed Attributes of the GTL I/O Standards**

Attributes	Primitives		
	IBUF/IBUFG	OBUF/OBUFT	IOBUF
IOSTANDARD	GTL and GTL_DCI		
CAPACITANCE	LOW, NORMAL, DONT_CARE		

GTLP (Gunning Transceiver Logic Plus)

The Gunning Transceiver Logic Plus or GTL+ standard is a high-speed bus standard (JESD8.3) first used by the Pentium Pro Processor. This standard requires a differential amplifier input buffer and a open-drain output buffer. The negative terminal of the differential input buffer is referenced to the V_{REF} pin.

A sample circuit illustrating a valid termination technique for GTL+ with external parallel termination and unconnected V_{CCO} is shown in Figure 6-37.

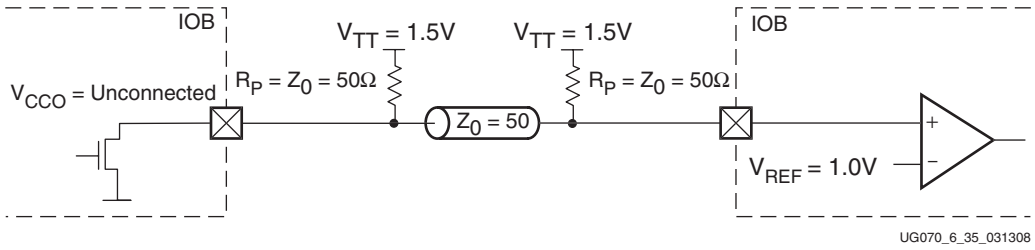


Figure 6-37: **GTL+ with External Parallel Termination and Unconnected V_{CCO}**

GTLP_DCI Usage

GTL+ does not require a V_{CCO} voltage. However, for GTLP_DCI, V_{CCO} must be connected to 1.5V. GTLP_DCI provides single termination to V_{CCO} for inputs or outputs.

A sample circuit illustrating a valid termination technique for GTLP_DCI with internal parallel driver and receiver termination is shown in Figure 6-38.

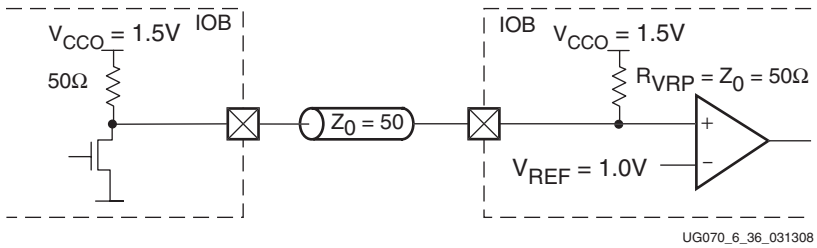


Figure 6-38: **GTLP_DCI Internal Parallel Driver and Receiver Termination**

Table 6-12 lists the GTLP DC voltage specifications.

Table 6-12: **GTLP DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	–	–	–
$V_{REF} = N \times V_{TT}$ (1)	0.88	1.0	1.12
V_{TT}	1.35	1.5	1.65
$V_{IH} = V_{REF} + 0.1$	0.98	1.1	–
$V_{IL} = V_{REF} - 0.1$	–	0.9	1.02
V_{OH}	–	–	–
V_{OL}	0.3	0.45	0.6
I_{OH} at V_{OH} (mA)	–	–	–
I_{OL} at V_{OL} (mA) at 0.6V	36	–	–
I_{OL} at V_{OL} (mA) at 0.3V	–	–	48

Notes:

1. N must be greater than or equal to 0.653 and less than or equal to 0.68.

Table 6-13 details the allowed attributes that can be applied to the GTLP I/O standards.

Table 6-13: **Allowed Attributes of the GTLP I/O Standards**

Attributes	Input	Output	Bidirectional
IOSTANDARD	GTLP and GTLP_DCI		
CAPACITANCE	LOW, NORMAL, DONT_CARE		

HSTL (High-Speed Transceiver Logic)

The High-Speed Transceiver Logic (HSTL) standard is a general-purpose high-speed, 1.5V or 1.8V bus standard sponsored by IBM (EIA/JESD8-6). This standard has four variations or classes. To support clocking high-speed memory interfaces, a CSE differential version of this standard was added. Virtex-4 FPGA I/O supports all four classes and the differential version. This standard requires a differential amplifier input buffer and a push-pull output buffer.

HSTL_ I, HSTL_ III, HSTL_ I_18, HSTL_ III_18 Usage

HSTL_I uses $V_{CCO}/2$ as a parallel termination voltage (V_{TT}). HSTL_III uses V_{CCO} as a parallel termination voltage (V_{TT}). HSTL_I and HSTL_III are intended to be used in unidirectional links.

HSTL_ I_DCI, HSTL_ III_DCI, HSTL_ I_DCI_18, HSTL_ III_DCI_18 Usage

HSTL_I_DCI provides on-chip split thevenin termination powered from V_{CCO}, creating an equivalent parallel termination voltage (V_{TT}) of $V_{CCO}/2$. HSTL_I_DCI and HSTL_III_DCI are intended to be used in unidirectional links.

HSTL_ II, HSTL_ IV, HSTL_ II_18, HSTL_ IV_18 Usage

HSTL_II uses $V_{CCO}/2$ as a parallel termination voltage (V_{TT}). HSTL_IV uses V_{CCO} as a parallel termination voltage (V_{TT}). HSTL_II and HSTL_IV are intended to be used in bidirectional links.

HSTL_ II_DCI, HSTL_ IV_DCI, HSTL_ II_DCI_18, HSTL_ IV_DCI_18 Usage

HSTL_II_DCI provides on-chip split thevenin termination powered from V_{CCO}, creating an equivalent termination voltage of $V_{CCO}/2$. HSTL_IV_ DCI provides single termination to V_{CCO} (V_{TT}). HSTL_II_DCI and HSTL_IV_ DCI are intended to be used in bidirectional links.

DIFF_HSTL_ II, DIFF_HSTL_II_18

Differential HSTL class II pairs complimentary single-ended HSTL_II type drivers with a differential receiver. Differential HSTL Class II is intended to be used in bidirectional links. Differential HSTL can also be used for differential clock and DQS signals in memory interface designs.

DIFF_HSTL_II_DCI, DIFF_HSTL_II_DCI_18

Differential HSTL class II pairs complimentary single-ended HSTL_II type drivers with a differential receiver, including on-chip differential termination. Differential HSTL Class II is intended to be used in bidirectional links. Differential HSTL can also be used for differential clock and DQS signals in memory interface designs.

HSTL Class I

Figure 6-39 shows a sample circuit illustrating a valid termination technique for HSTL Class I.

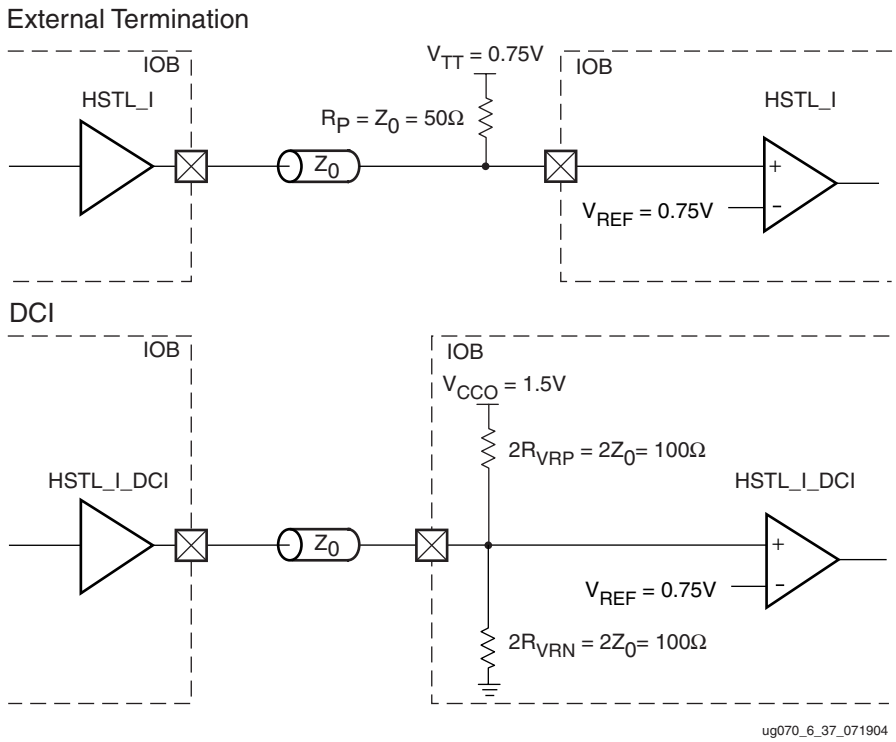


Figure 6-39: **HSTL Class I Termination**

Table 6-14 lists the HSTL Class I DC voltage specifications.

Table 6-14: **HSTL Class I DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.40	1.50	1.60
V_{REF} (2)	0.68	0.75	0.90
V_{TT}	–	$V_{CCO} \times 0.5$	–
V_{IH}	V_{REF} + 0.1	–	–
V_{IL}	–	–	V_{REF} – 0.1
V_{OH}	V_{CCO} – 0.4	–	–
V_{OL}	–	–	0.4
I_{OH} at V_{OH} (mA)(1)	–8	–	–
I_{OL} at V_{OL} (mA)(1)	8	–	–

Notes:

1. V_{OL} and V_{OH} for lower drive currents are sample tested.
2. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

HSTL Class II

Figure 6-40 shows a sample circuit illustrating a valid termination technique for HSTL Class II (1.5V) with unidirectional termination.

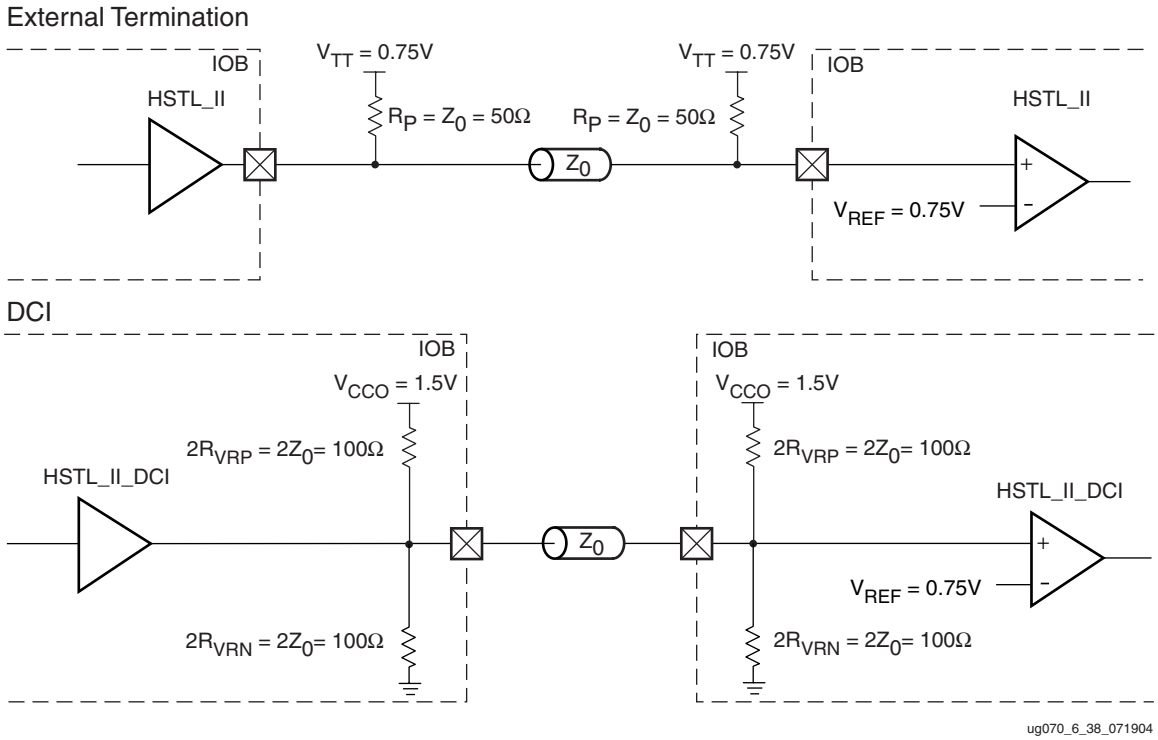


Figure 6-40: **HSTL (1.5V) Class II Unidirectional Termination**

Figure 6-41 shows a sample circuit illustrating a valid termination technique for HSTL Class II (1.5V) with bidirectional termination.

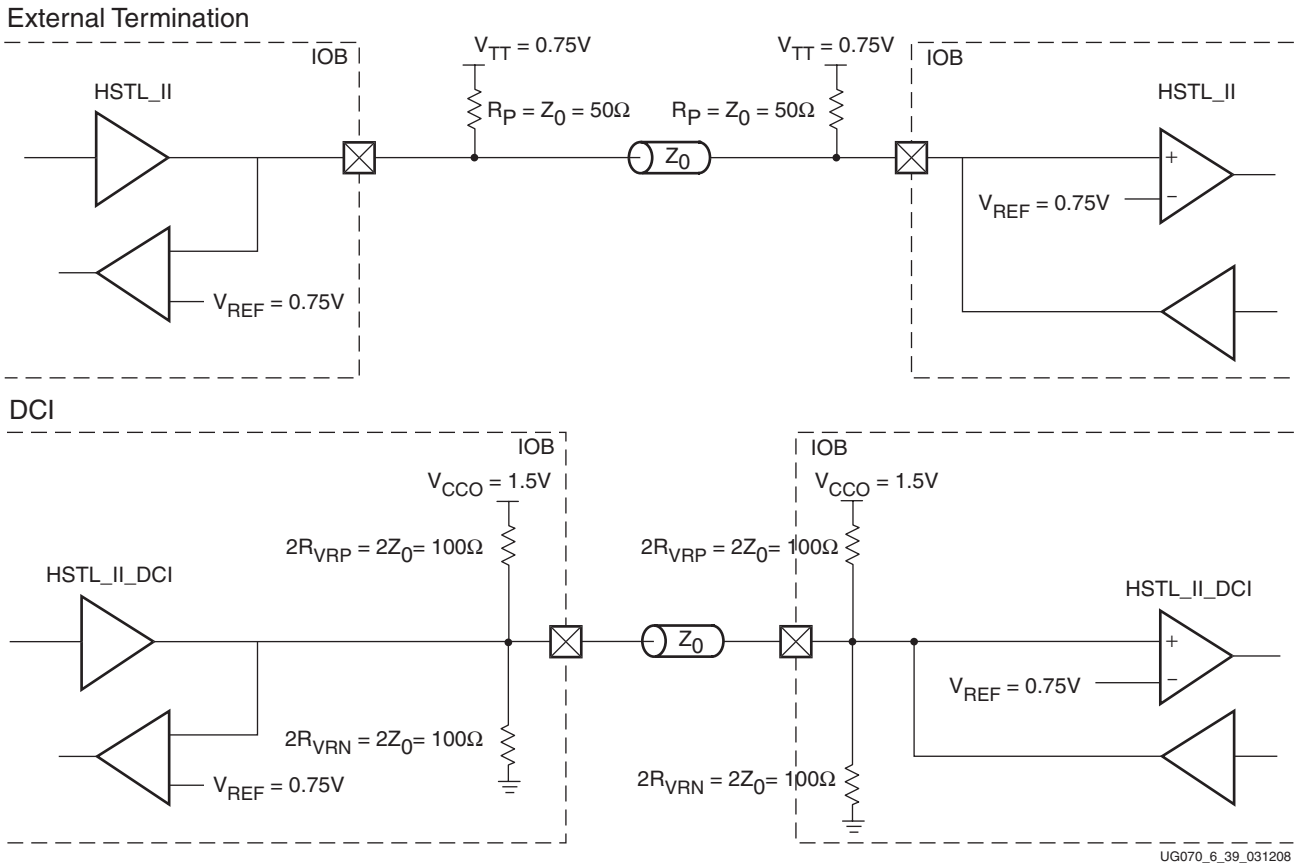


Figure 6-41: **HSTL (1.5V) Class II Bidirectional Termination**

Table 6-15 lists the HSTL (1.5V) Class II DC voltage specifications.

Table 6-15: **HSTL (1.5V) Class II DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.40	1.50	1.60
V_{REF} (2)	0.68	0.75	0.90
V_{TT}	–	$V_{CCO} \times 0.5$	–
V_{IH}	$V_{REF} + 0.1$	–	–
V_{IL}	–	–	$V_{REF} - 0.1$
V_{OH}	$V_{CCO} - 0.4$	–	–
V_{OL}	–	–	0.4
I_{OH} at V_{OH} (mA)(1)	–16	–	–
I_{OL} at V_{OL} (mA)(1)	16	–	–

Notes:

1. V_{OL} and V_{OH} for lower drive currents are sample tested.
2. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

Complementary Single-Ended (CSE) Differential HSTL Class II

Figure 6-42 shows a sample circuit illustrating a valid termination technique for differential HSTL Class II (1.5V) with unidirectional termination.

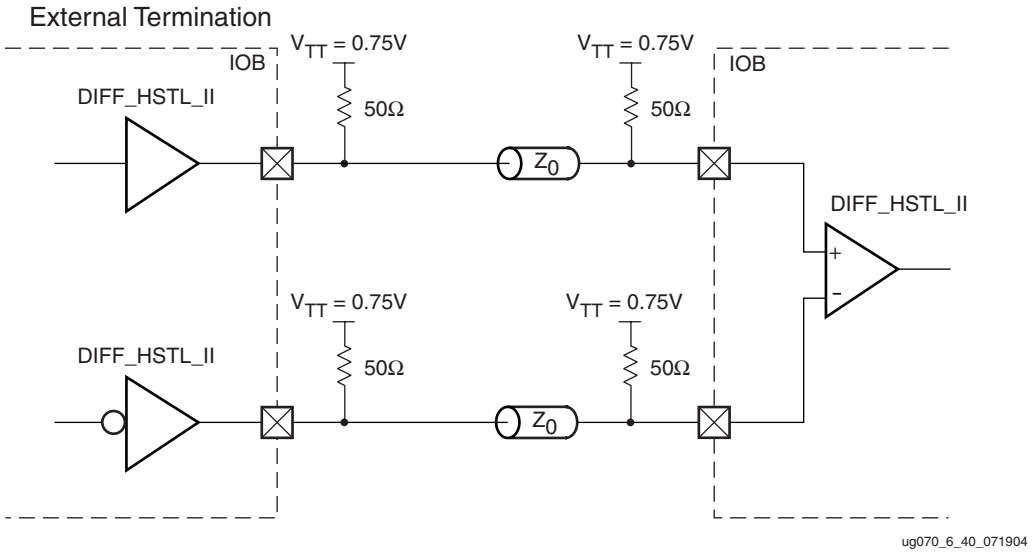


Figure 6-42: **Differential HSTL (1.5V) Class II Unidirectional Termination**

Figure 6-43 shows a sample circuit illustrating a valid termination technique for differential HSTL Class II (1.5V) with unidirectional DCI termination.

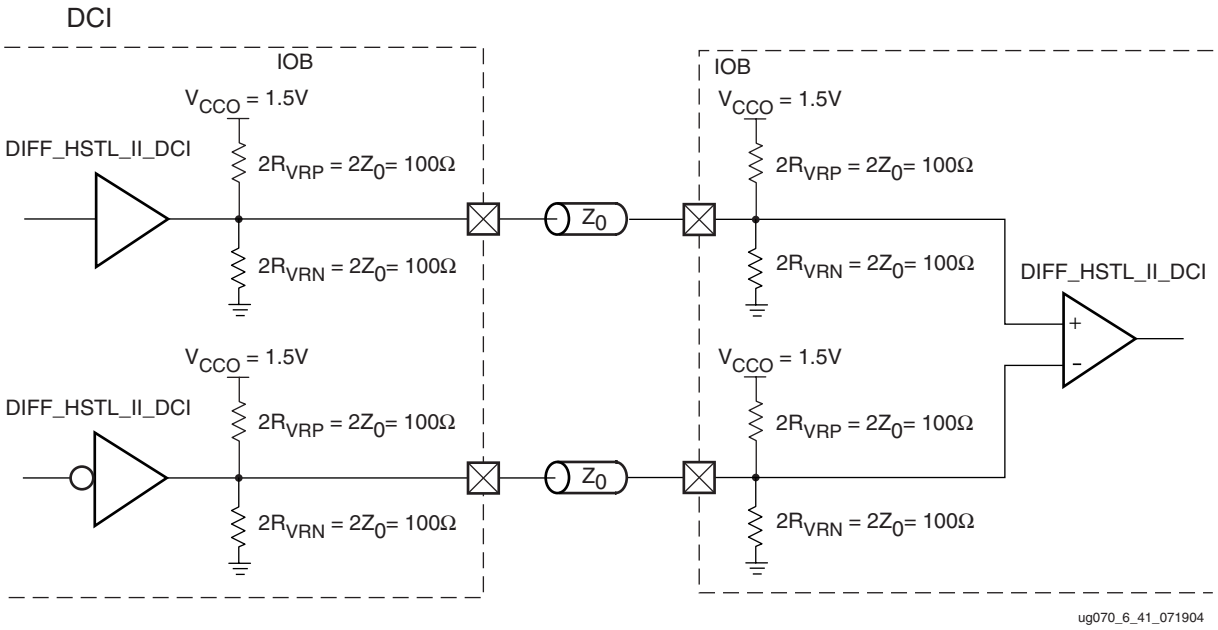


Figure 6-43: **Differential HSTL (1.5V) Class II DCI Unidirectional Termination**

Figure 6-44 shows a sample circuit illustrating a valid termination technique for differential HSTL Class II (1.5V) with bidirectional termination.

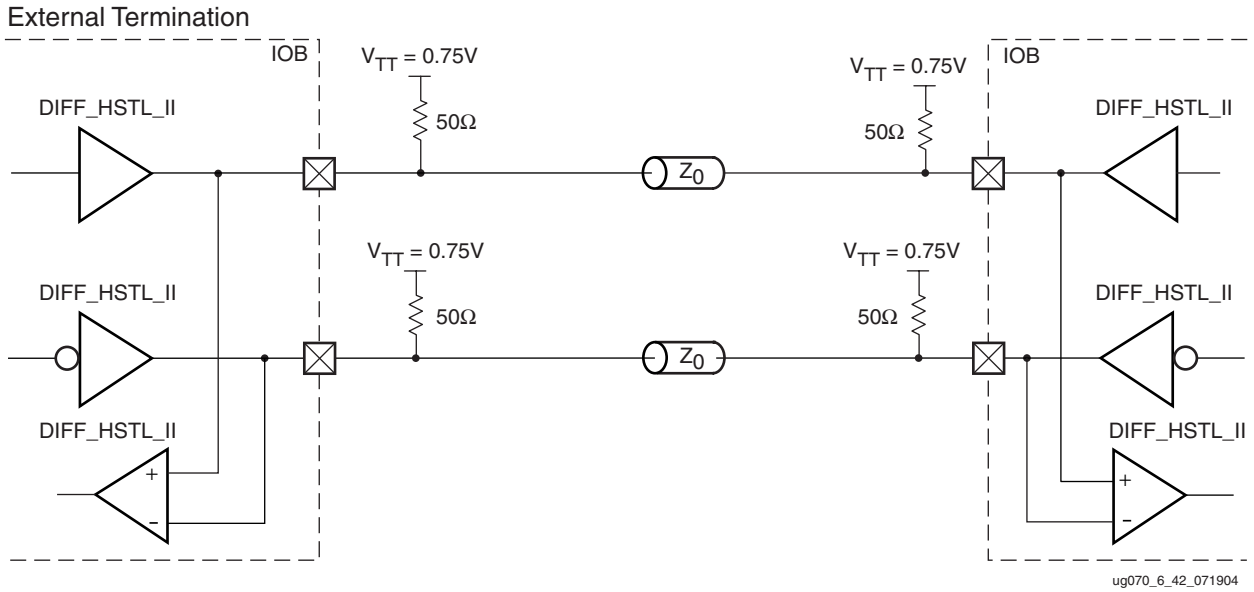


Figure 6-44: **Differential HSTL (1.5V) Class II Bidirectional Termination**

Figure 6-45 shows a sample circuit illustrating a valid termination technique for differential HSTL Class II (1.5V) with bidirectional DCI termination.

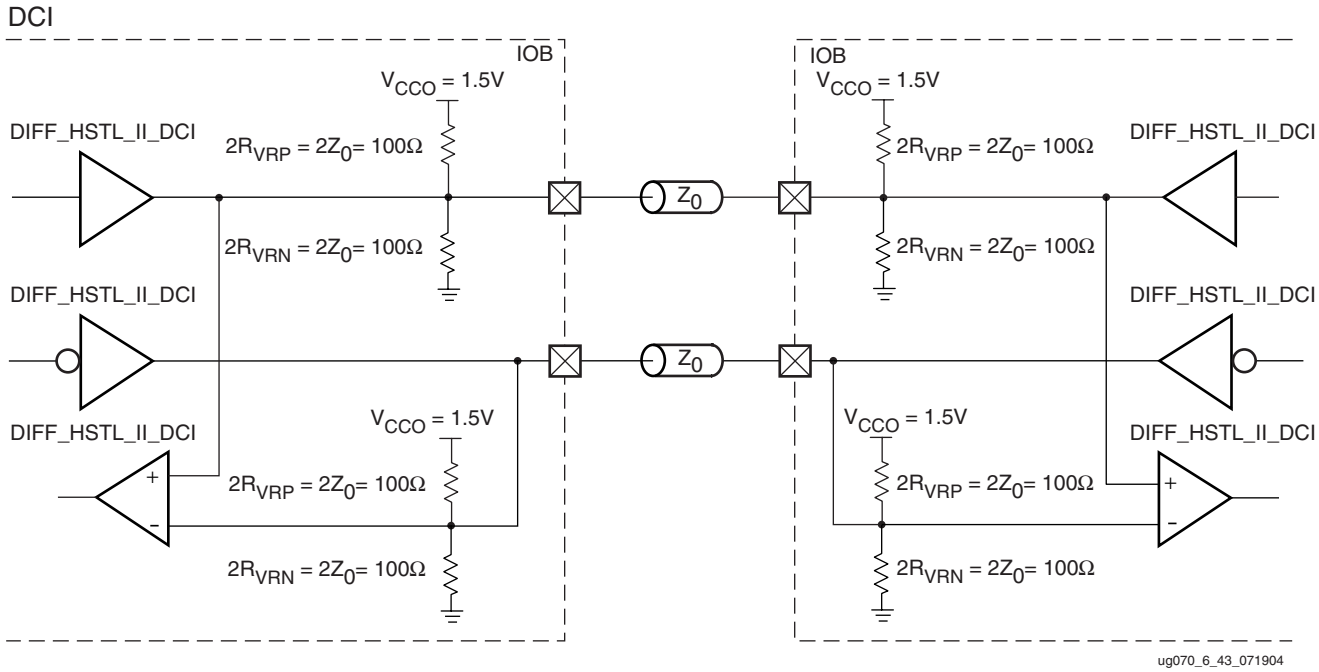


Figure 6-45: **Differential HSTL (1.5V) Class II DCI Bidirectional Termination**

Table 6-16 lists the differential HSTL Class II DC voltage specifications.

Table 6-16: **Differential HSTL Class II DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.40	1.50	1.60
V_{TT}	–	$V_{CCO} \times 0.5$	–
V_{IN} (DC)	–0.30	–	$V_{CCO} + 0.30$
V_{DIFF} (DC)	0.20	–	$V_{CCO} + 0.60$
V_{CM} (DC)(1)	0.68	–	0.90
V_{DIFF} (AC)	0.40	–	$V_{CCO} + 0.60$
V_X (Crossover)(2)	0.68	–	0.90

Notes:

1. Common mode voltage: $V_{CM} = V_P - ((V_P - V_N)/2$
2. Crossover point: V_X where $V_P - V_N = 0$ (AC coupled)

HSTL Class III

Figure 6-46 shows a sample circuit illustrating a valid termination technique for HSTL Class III.

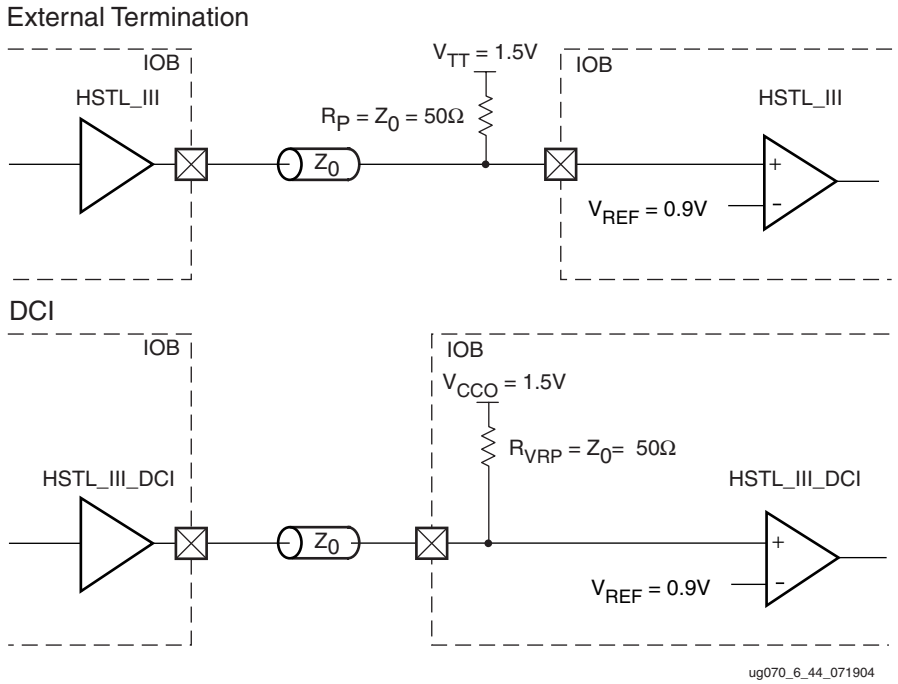


Figure 6-46: **HSTL Class III Termination**

Table 6-17 lists the HSTL Class III DC voltage specifications.

Table 6-17: **HSTL Class III DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.40	1.50	1.60
V_{REF} (2)	–	0.90	–
V_{TT}	–	V_{CCO}	–
V_{IH}	V_{REF} + 0.1	–	–
V_{IL}	–	–	V_{REF} – 0.1
V_{OH}	V_{CCO} – 0.4	–	–
V_{OL}	–	–	0.4
I_{OH} at V_{OH} (mA)(1)	–8	–	–
I_{OL} at V_{OL} (mA)(1)	24	–	–

Notes:

1. V_{OL} and V_{OH} for lower drive currents are sample tested.
2. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

HSTL Class IV

Figure 6-47 shows a sample circuit illustrating a valid unidirectional termination technique for HSTL Class IV.

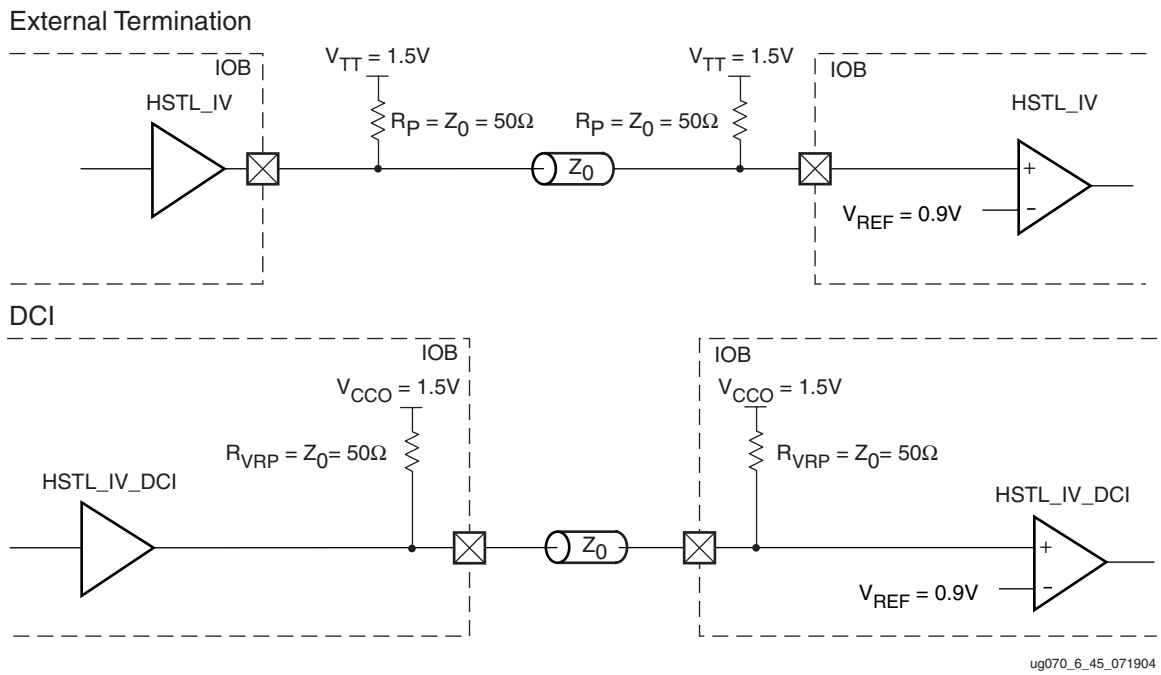


Figure 6-47: **HSTL Class IV Unidirectional Termination**

Figure 6-48 shows a sample circuit illustrating a valid bidirectional termination technique for HSTL Class IV.

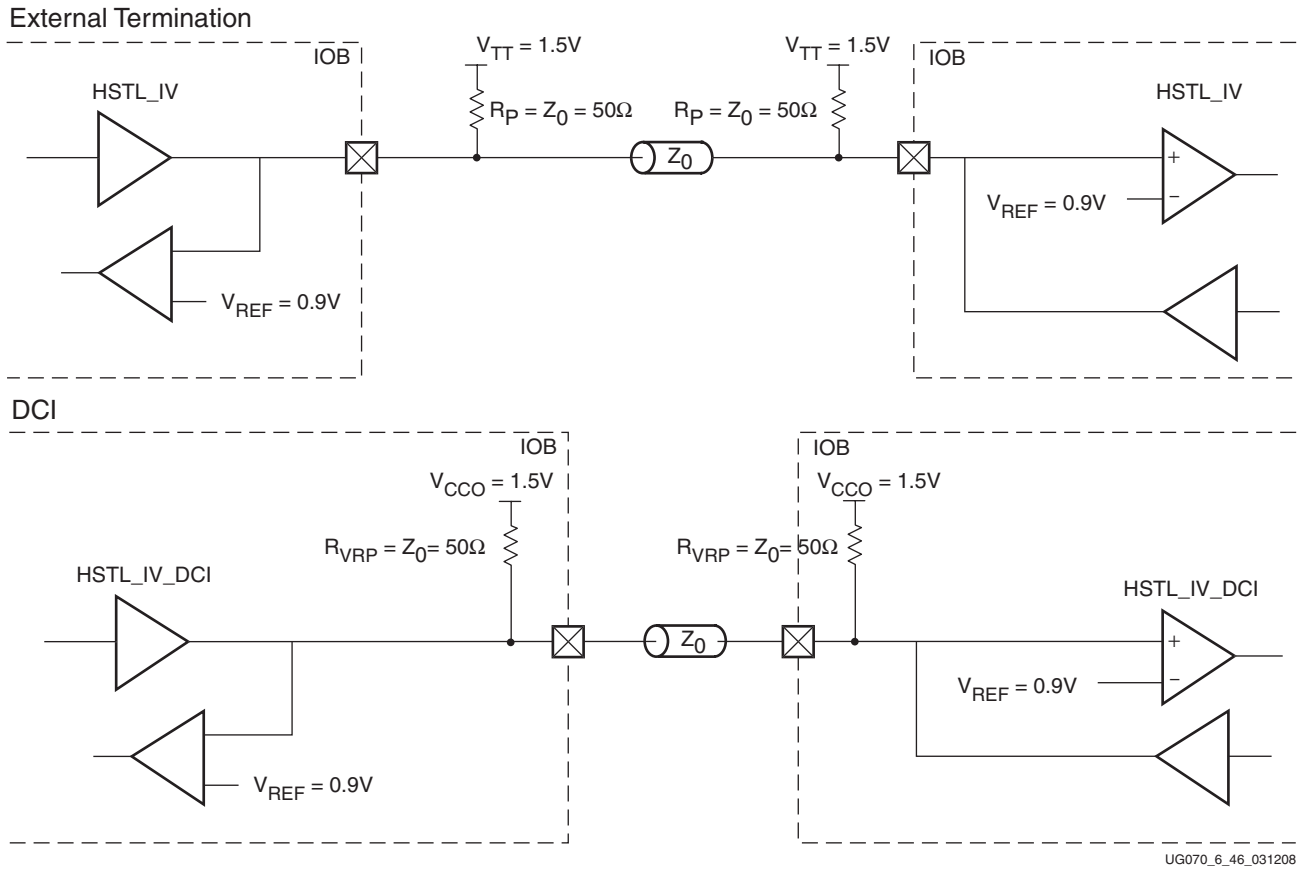


Figure 6-48: **HSTL Class IV Bidirectional Termination**

Table 6-18 lists the HSTL Class IV DC voltage specifications.

Table 6-18: **HSTL Class IV DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.40	1.50	1.60
V_{REF} (2)	–	0.90	–
V_{TT}	–	V_{CCO}	–
V_{IH}	V_{REF} + 0.1	–	–
V_{IL}	–	–	V_{REF} – 0.1
V_{OH}	V_{CCO} – 0.4	–	–
V_{OL}	–	–	0.4
I_{OH} at V_{OH} (mA)(1)	–8	–	–
I_{OL} at V_{OL} (mA)(1)	48	–	–

Notes:

1. V_{OL} and V_{OH} for lower drive currents are sample tested.
2. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

HSTL Class I (1.8V)

Figure 6-49 shows a sample circuit illustrating a valid termination technique for HSTL Class I (1.8V).

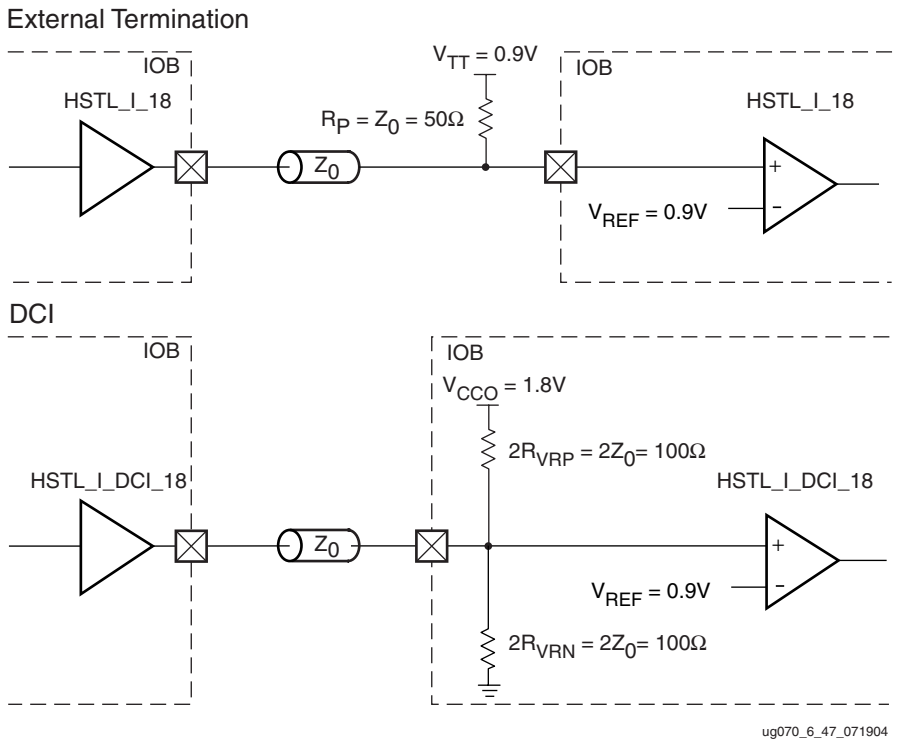


Figure 6-49: **HSTL Class I (1.8V) Termination**

Table 6-19 lists the HSTL Class I (1.8V) DC voltage specifications.

Table 6-19: **HSTL Class I (1.8V) DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.7	1.8	1.9
V_{REF} (2)	0.8	0.9	1.1
V_{TT}	–	$V_{CCO} \times 0.5$	–
V_{IH}	V_{REF} + 0.1	–	–
V_{IL}	–	–	V_{REF} – 0.1
V_{OH}	V_{CCO} – .0.4	–	–
V_{OL}	–	–	0.4
I_{OH} at V_{OH} (mA)(1)	–8	–	–
I_{OL} at V_{OL} (mA)(1)	8	–	–

Notes:

1. V_{OL} and V_{OH} for lower drive currents are sample tested.
2. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

HSTL Class II (1.8V)

Figure 6-50 shows a sample circuit illustrating a valid termination technique for HSTL Class II (1.8V) with unidirectional termination.

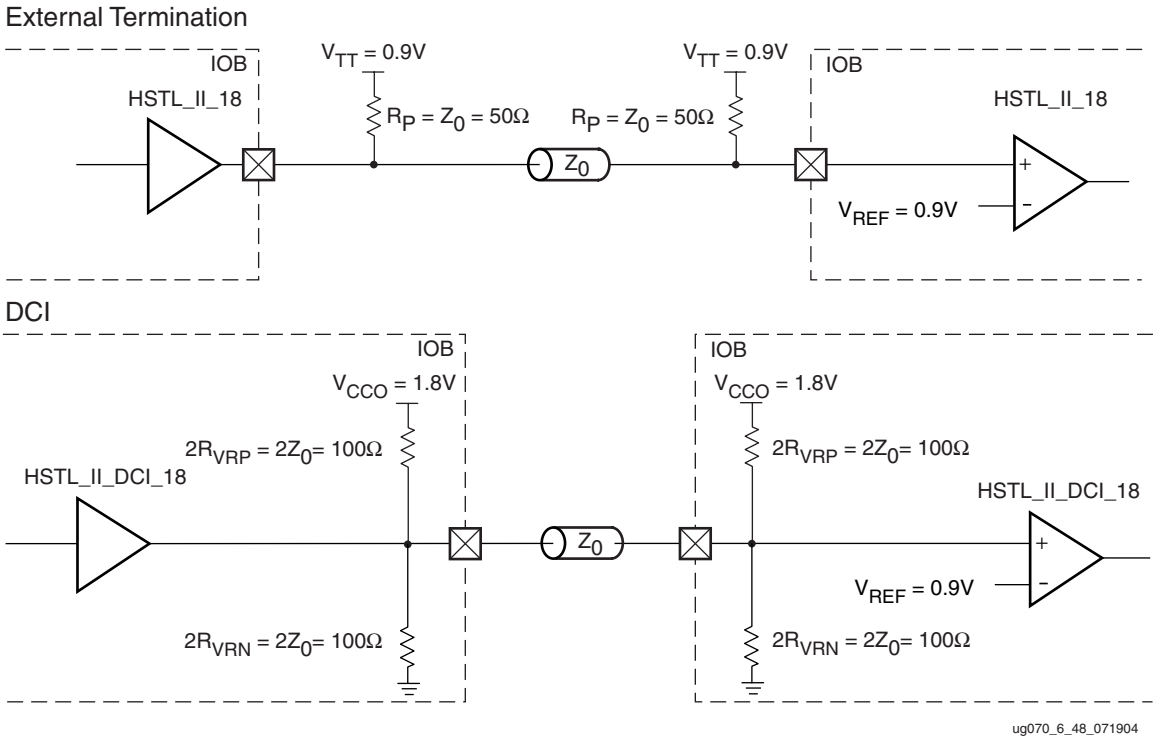


Figure 6-50: **HSTL Class II (1.8V) with Unidirectional Termination**

Figure 6-51 shows a sample circuit illustrating a valid termination technique for HSTL Class II (1.8V) with bidirectional termination.

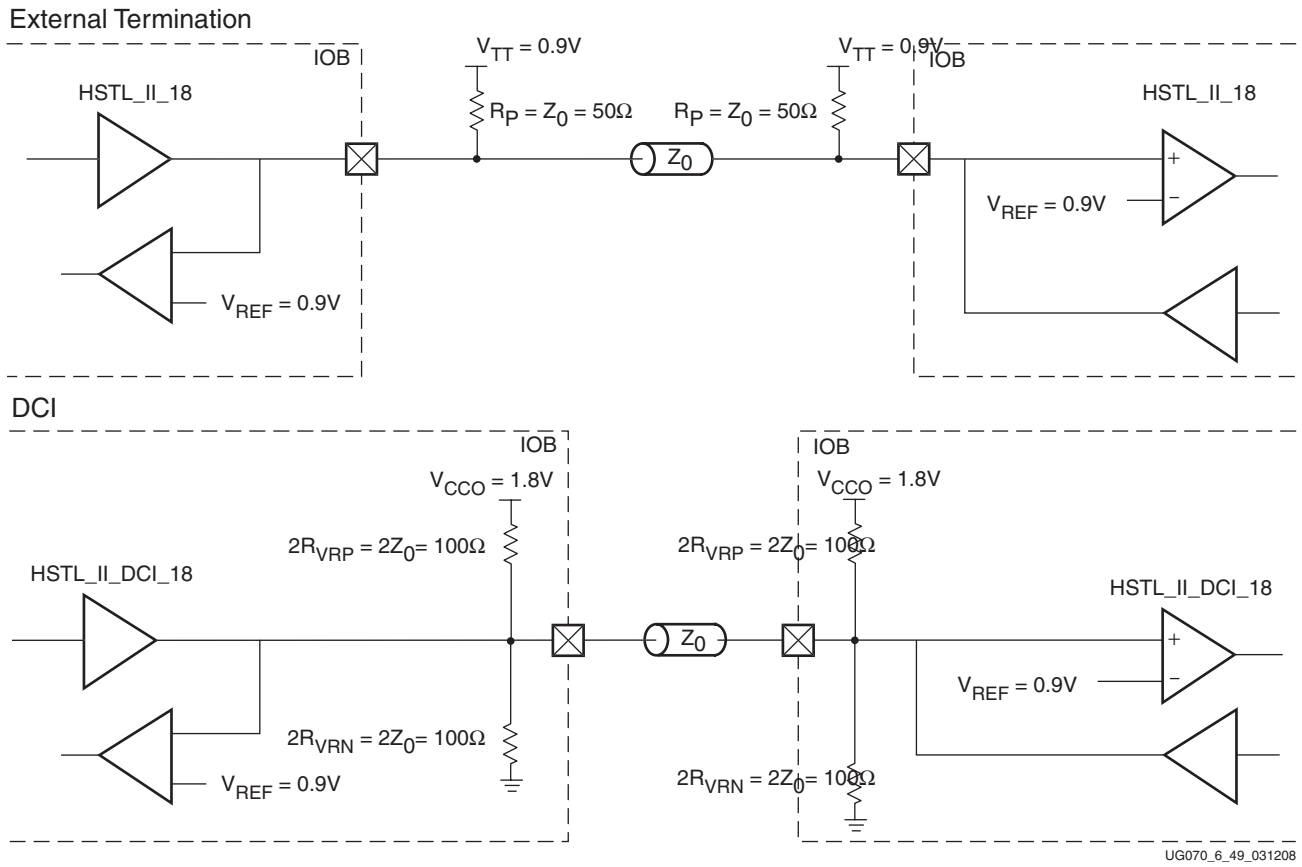


Figure 6-51: **HSTL Class II (1.8V) with Bidirectional Termination**

Table 6-20 lists the HSTL Class II (1.8V) DC voltage specifications.

Table 6-20: **HSTL Class II (1.8V) DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.7	1.8	1.9
V_{REF} (2)	–	0.9	–
V_{TT}	–	$V_{CCO} \times 0.5$	–
V_{IH}	$V_{REF} + 0.1$	–	–
V_{IL}	–	–	$V_{REF} - 0.1$
V_{OH}	$V_{CCO} - 0.4$	–	–
V_{OL}	–	–	0.4
I_{OH} at V_{OH} (mA)(1)	–16	–	–
I_{OL} at V_{OL} (mA)(1)	16	–	–

Notes:

1. V_{OL} and V_{OH} for lower drive currents are sample tested.
2. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

Complementary Single-Ended (CSE) Differential HSTL Class II (1.8V)

Figure 6-52 shows a sample circuit illustrating a valid termination technique for differential HSTL Class II (1.8V) with unidirectional termination.

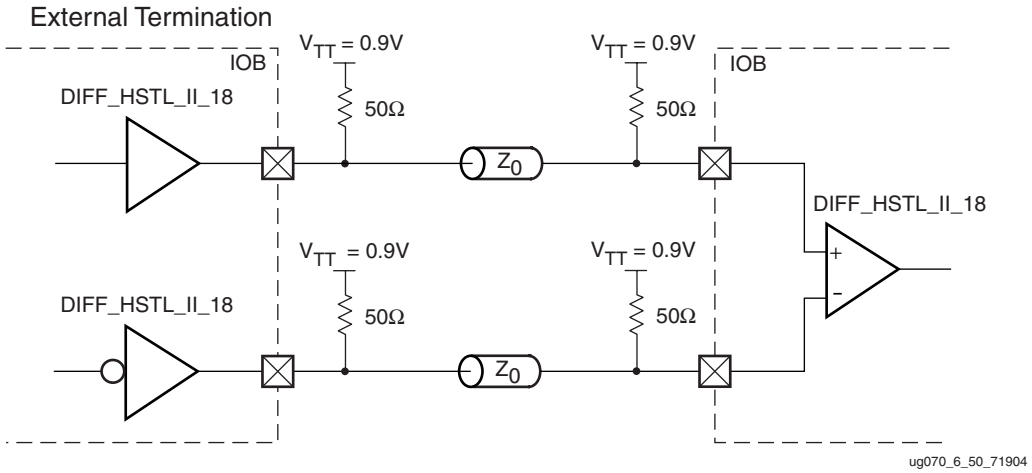


Figure 6-52: **Differential HSTL (1.8V) Class II Unidirectional Termination**

Figure 6-53 shows a sample circuit illustrating a valid termination technique for differential HSTL Class II (1.8V) with unidirectional DCI termination.

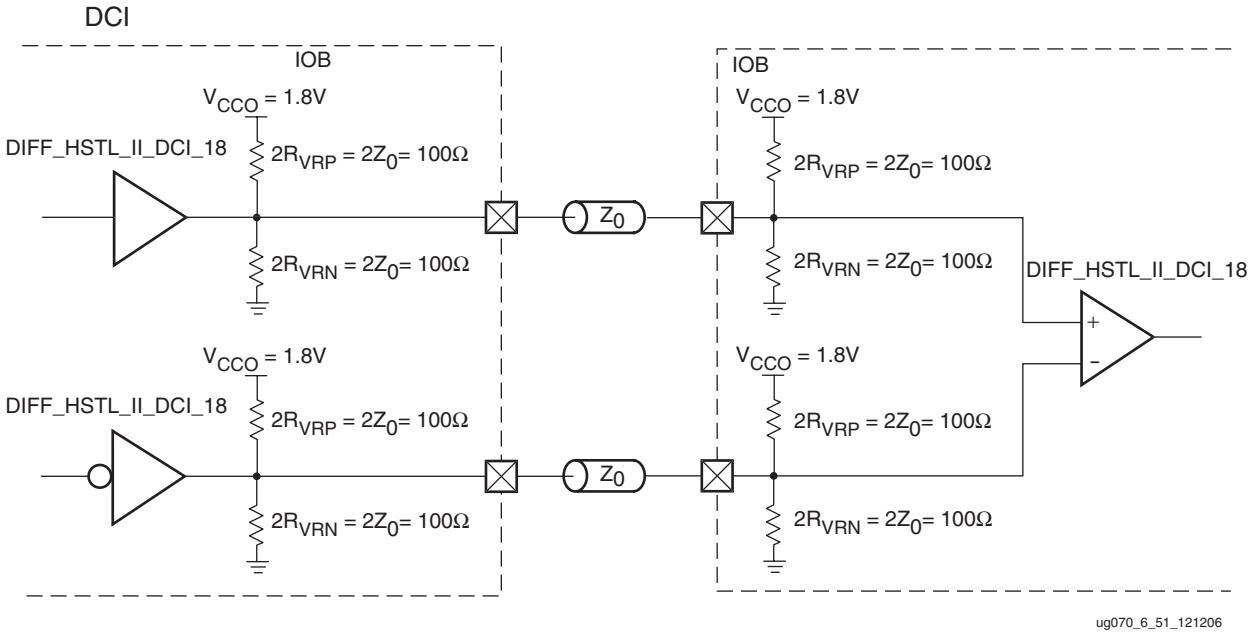


Figure 6-53: **Differential HSTL (1.8V) Class II DCI Unidirectional Termination**

Figure 6-54 shows a sample circuit illustrating a valid termination technique for differential HSTL Class II (1.8V) with bidirectional termination.

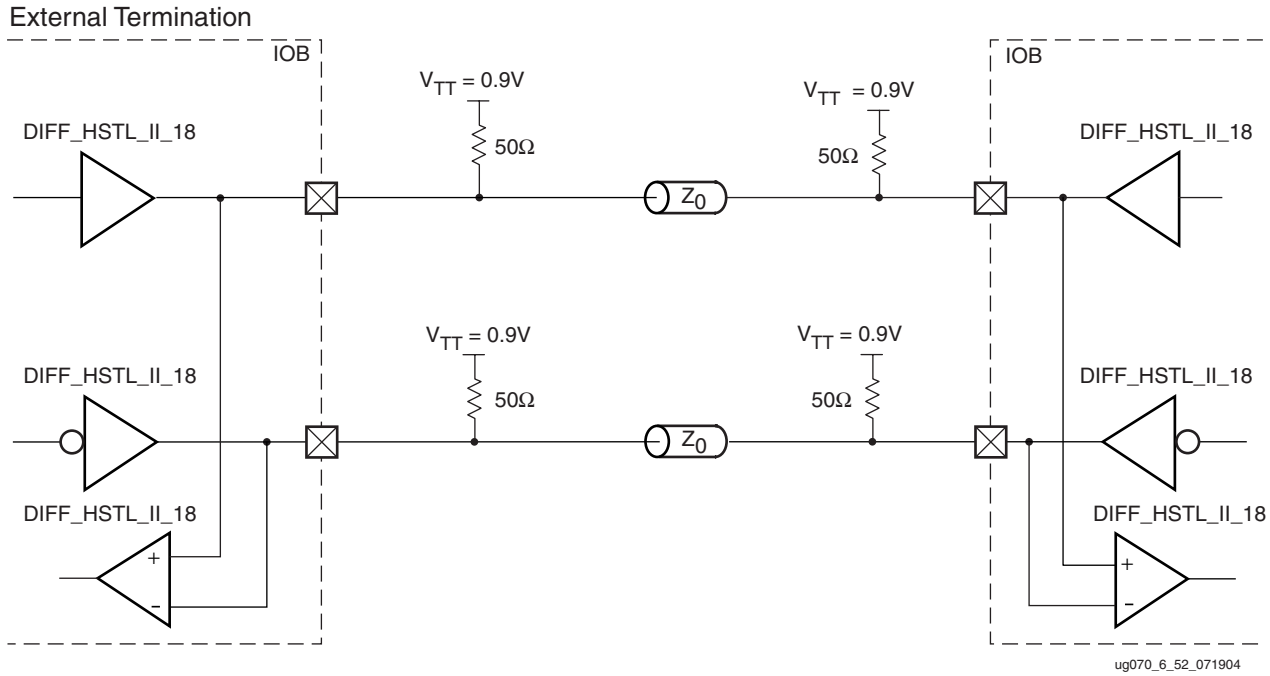


Figure 6-54: **Differential HSTL (1.8V) Class II Bidirectional Termination**

Figure 6-55 shows a sample circuit illustrating a valid termination technique for differential HSTL Class II (1.8V) with bidirectional DCI termination.

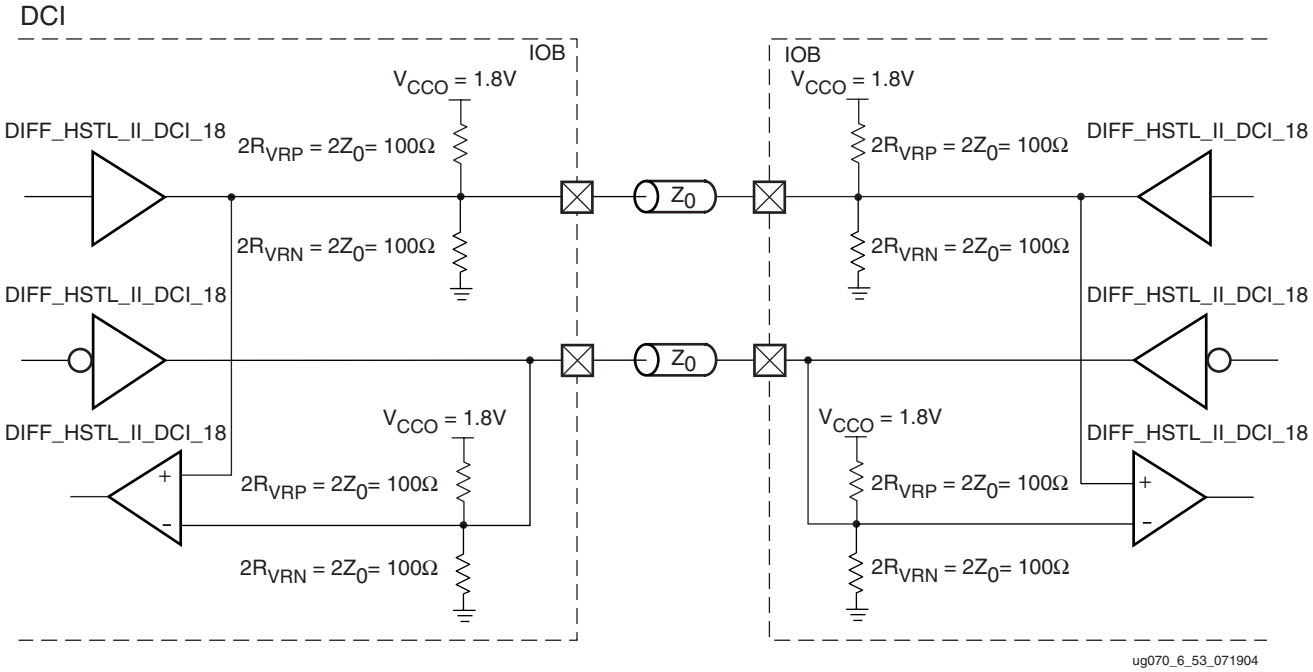


Figure 6-55: **Differential HSTL (1.8V) Class II DCI Bidirectional Termination**

Table 6-21 lists the differential HSTL Class II (1.8V) DC voltage specifications.

Table 6-21: **Differential HSTL Class II (1.8V) DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.7	1.8	1.9
V_{TT}	–	$V_{CCO} \times 0.5$	–
V_{IN} (DC)	–0.30	–	$V_{CCO} + 0.30$
V_{DIFF} (DC)	0.20	–	$V_{CCO} + 0.60$
V_{CM} (DC)(1)	0.78	–	1.12
V_{DIFF} (AC)	0.40	–	$V_{CCO} + 0.60$
V_X (Crossover)(2)	0.78	–	1.12

Notes:

1. Common mode voltage: $V_{CM} = V_P - ((V_P - V_N)/2$
2. Crossover point: V_X where $V_P - V_N = 0$ (AC coupled)

HSTL Class III (1.8V)

Figure 6-56 shows a sample circuit illustrating a valid termination technique for HSTL Class III (1.8V).

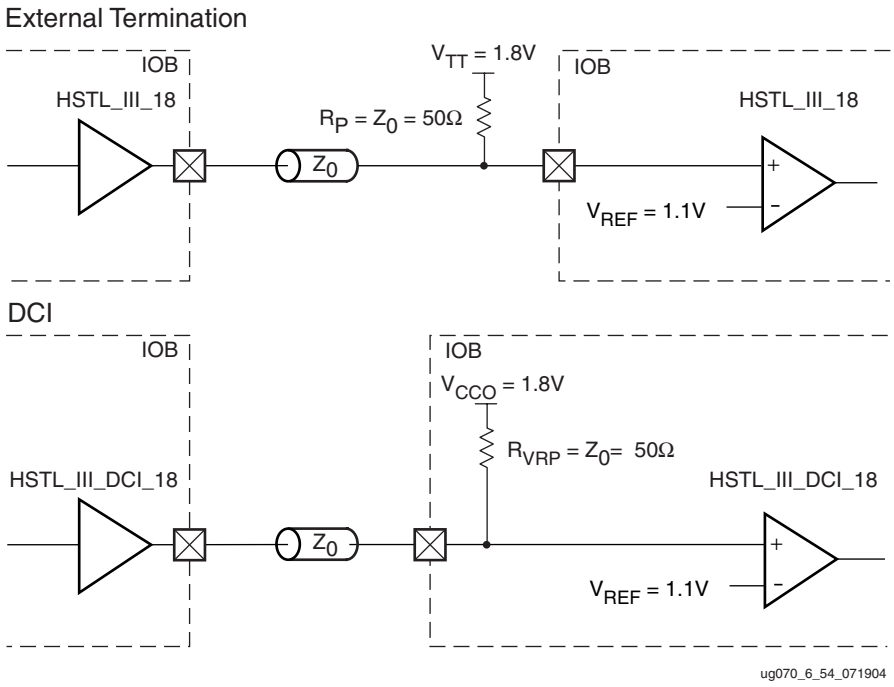


Figure 6-56: **HSTL Class III (1.8V) Termination**

Table 6-22 lists the HSTL Class III (1.8V) DC voltage specifications.

Table 6-22: **HSTL Class III (1.8V) DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.7	1.8	1.9
V_{REF} (2)	–	1.1	–
V_{TT}	–	V_{CCO}	–
V_{IH}	V_{REF} + 0.1	–	–
V_{IL}	–	–	V_{REF} – 0.1
V_{OH}	V_{CCO} – 0.4	–	–
V_{OL}	–	–	0.4
I_{OH} at V_{OH} (mA)(1)	–8	–	–
I_{OL} at V_{OL} (mA)(1)	24	–	–

Notes:

1. V_{OL} and V_{OH} for lower drive currents are sample tested.
2. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

HSTL Class IV (1.8V)

Figure 6-57 shows a sample circuit illustrating a valid unidirectional termination technique for HSTL Class IV (1.8V).

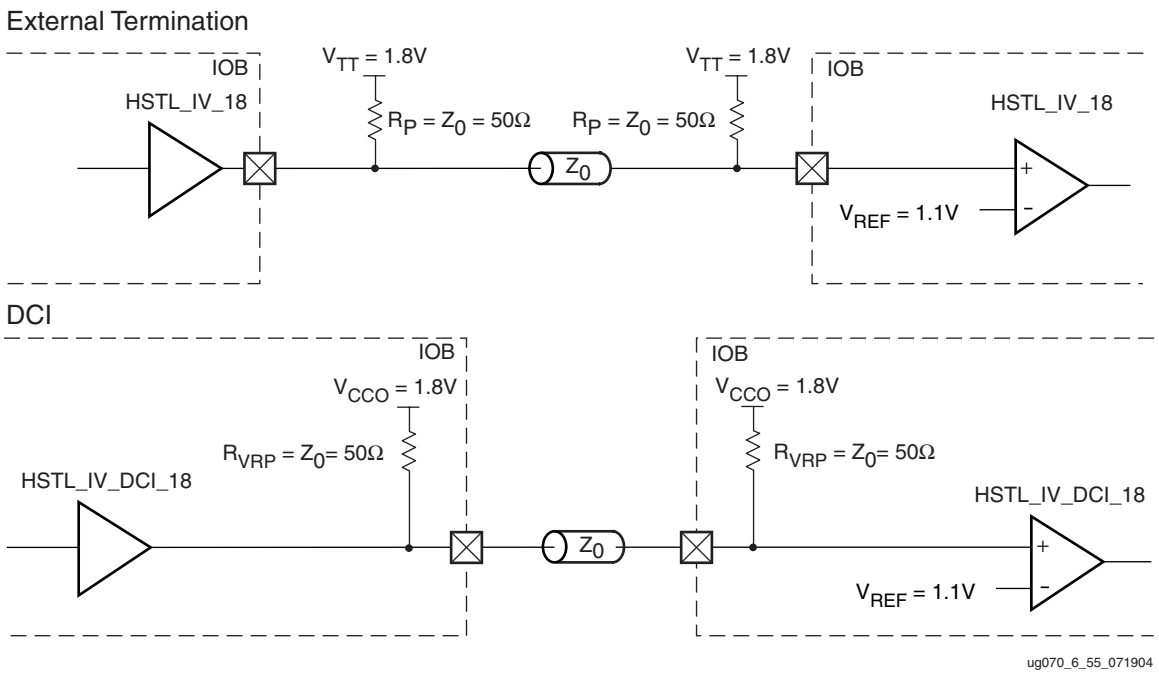


Figure 6-57: **HSTL Class IV (1.8V) with Unidirectional Termination**

Figure 6-58 shows a sample circuit illustrating a valid bidirectional termination technique for HSTL Class IV (1.8V).

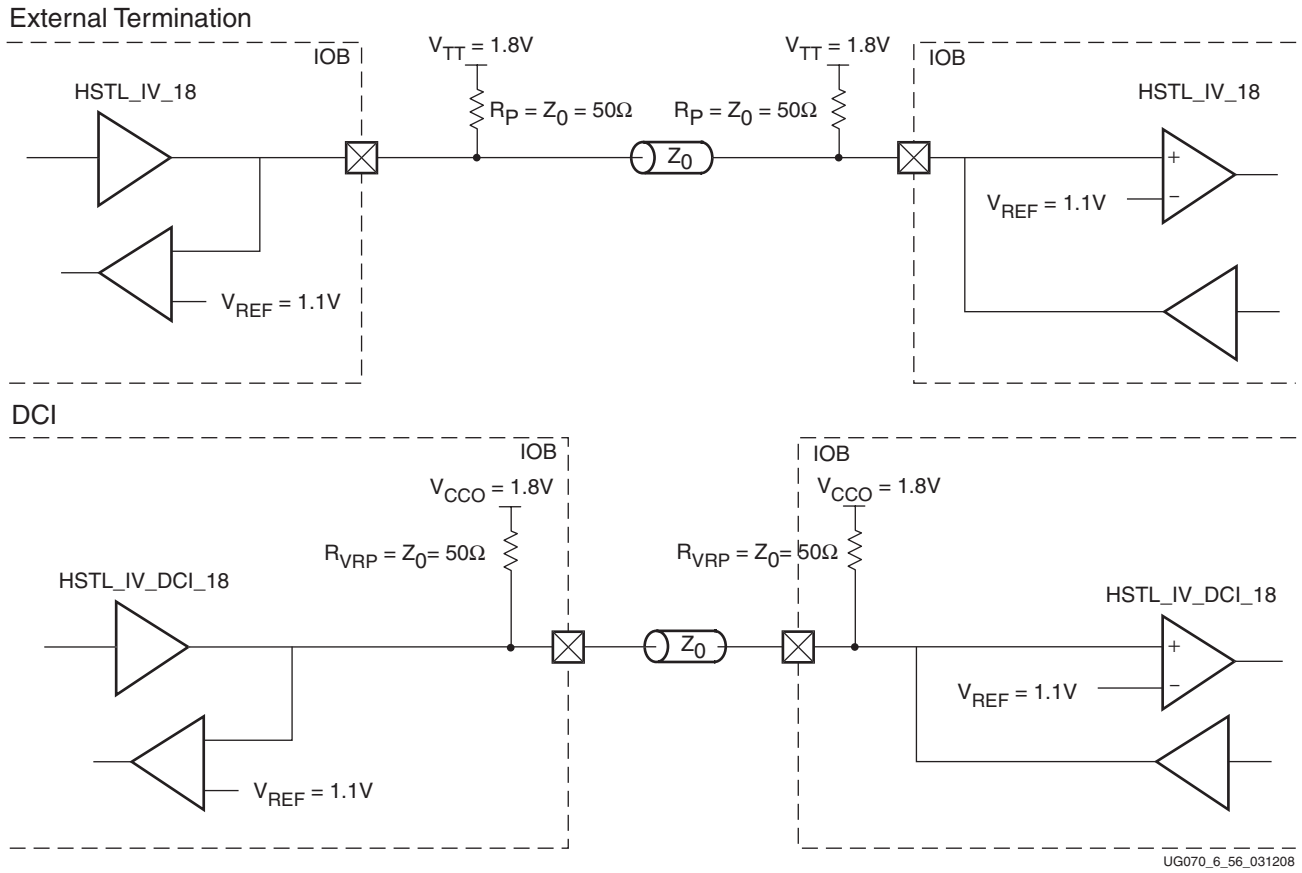


Figure 6-58: **HSTL Class IV (1.8V) with Bidirectional Termination**

Table 6-23 lists the HSTL Class IV (1.8V) DC voltage specifications.

Table 6-23: **HSTL Class IV (1.8V) DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.7	1.8	1.9
V_{REF} (2)	–	1.1	–
V_{TT}	–	V_{CCO}	–
V_{IH}	V_{REF} + 0.1	–	–
V_{IL}	–	–	V_{REF} – 0.1
V_{OH}	V_{CCO} – 0.4	–	–
V_{OL}	–	–	0.4
I_{OH} at V_{OH} (mA)(1)	–8	–	–
I_{OL} at V_{OL} (mA)(1)	48	–	–

Notes:

1. V_{OL} and V_{OH} for lower drive currents are sample tested.
2. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

Table 6-24 details the allowed attributes that can be applied to the HSTL I/O standards.

Table 6-24: **Allowed Attributes of the HSTL I/O Standards**

Attributes	Primitives		
	IBUF/IBUFG	OBUF/OBUFT	IOBUF
IOSTANDARD	All possible HSTL standards		
CAPACITANCE	LOW, NORMAL, DONT_CARE		

Table 6-25 details the allowed attributes that can be applied to the DIFF_HSTL I/O standards.

Table 6-25: **Allowed Attributes of the DIFF_HSTL I/O Standards**

Attributes	Primitives		
	IBUFDS/IBUFGDS	OBUFDS/OBUFTDS	IOBUFDS
IOSTANDARD	All possible DIFF_HSTL standards		
CAPACITANCE	LOW, NORMAL, DONT_CARE		

SSTL (Stub-Series Terminated Logic)

The Stub-Series Terminated Logic (SSTL) for 2.5V (SSTL2) and 1.8V (SSTL18) is a standard for a general purpose memory bus. These standards are sponsored by Hitachi, IBM, and are defined in the JEDEC JESD8-15 documents. The standard has two classes; Class I is for unidirectional and Class II is for bidirectional signaling. Virtex-4 FPGA I/O supports both standards for single-ended signaling and Class II only for differential signaling. This standard requires a differential amplifier input buffer and a push-pull output buffer.

SSTL2_I, SSTL18_I Usage

Class I signaling uses V_{TT} ($V_{CCO}/2$) as a parallel termination voltage to a 50Ω resistor at the receiver. A series resistor (25Ω at 2.5V, 20Ω at 1.8V) must be connected to the transmitter output.

SSTL2_I_DCI, SSTL18_I_DCI Usage

The DCI transmitter provides the internal series resistance (25Ω at 2.5V, 20Ω at 1.8V). The DCI receiver has an internal split thevenin termination powered from V_{CCO} creating an equivalent V_{TT} voltage and termination impedance.

SSTL2_II, SSTL18_II Usage

Class II signaling uses V_{TT} ($V_{CCO}/2$) as a parallel termination voltage to a 50Ω resistor at the receiver and transmitter respectively. A series resistor (25Ω at 2.5V, 20Ω at 1.8V) must be connected to the transmitter output for a unidirectional link. For a bidirectional link, 25Ω series resistors must connected the transmitters of the transceivers.

SSTL2_II_DCI, SSTL18_II_DCI Usage

The DCI circuits have a split thevenin termination powered from V_{CCO} and an internal series resistor (25Ω at 2.5V, 20Ω at 1.8V). For a unidirectional link the series resistance is supplied only for the transmitter. A bidirectional link has the series resistor for both transmitters.

DIFF_SSTL2_II, DIFF_SSTL18_II Usage

Differential SSTL 2.5V and 1.8V Class II pairs complementary single-ended SSTL_II type drivers with a differential receiver. For a bidirectional link, a series resistor must be connected to both transmitters.

DIFF_SSTL2_II_DCI, DIFF_SSTL18_II_DCI Usage

Differential SSTL 2.5V and 1.8V Class II pairs complementary single-ended SSTL_II type drivers with a differential receiver, including on-chip termination. DCI can be used for unidirectional and bidirectional links.

SSTL2 Class I (2.5V)

Figure 6-59 shows a sample circuit illustrating a valid termination technique for SSTL2 Class I.

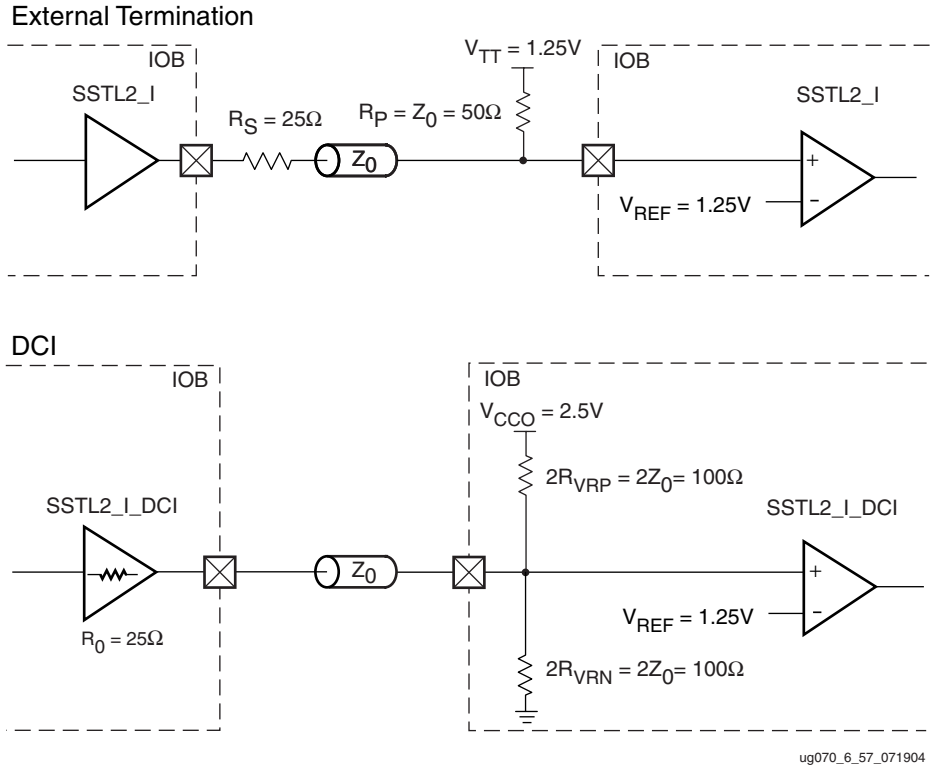


Figure 6-59: **SSTL2 Class I Termination**

Table 6-26 lists the SSTL2 DC voltage specifications for Class I.

Table 6-26: **SSTL2 DC Voltage Specifications Class I**

	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
$V_{REF} = 0.5 \times V_{CCO}$	1.13	1.25	1.38
$V_{TT} = V_{REF} + N^{(1)}$	1.09	1.25	1.42
$V_{IH} \geq V_{REF} + 0.15$	1.28	1.4	$3.0^{(2)}$

Table 6-26: **SSTL2 DC Voltage Specifications Class I *(Continued)***

	Min	Typ	Max
$V_{IL} \leq V_{REF} - 0.15$	–0.3(3)	1.1	1.23
$V_{OH} \geq V_{REF} + 0.61$	1.74	1.84	1.94
$V_{OL} \leq V_{REF} - 0.61$(4)	0.56	0.66	0.76
I_{OH} at V_{OH} (mA)	–8.1	–	–
I_{OL} at V_{OL} (mA)	8.1	–	–

Notes:

1. N must be greater than or equal to –0.04 and less than or equal to 0.04.
2. V_{IH} maximum is V_{CCO} +0.3.
3. V_{IL} minimum does not conform to the formula.
4. Because SSTL2_I_DCI uses a controlled-impedance driver, V_{OH} and V_{OL} are different.

SSTL2 Class II (2.5V)

Figure 6-60 shows a sample circuit illustrating a valid unidirectional termination technique for SSTL2 Class II.

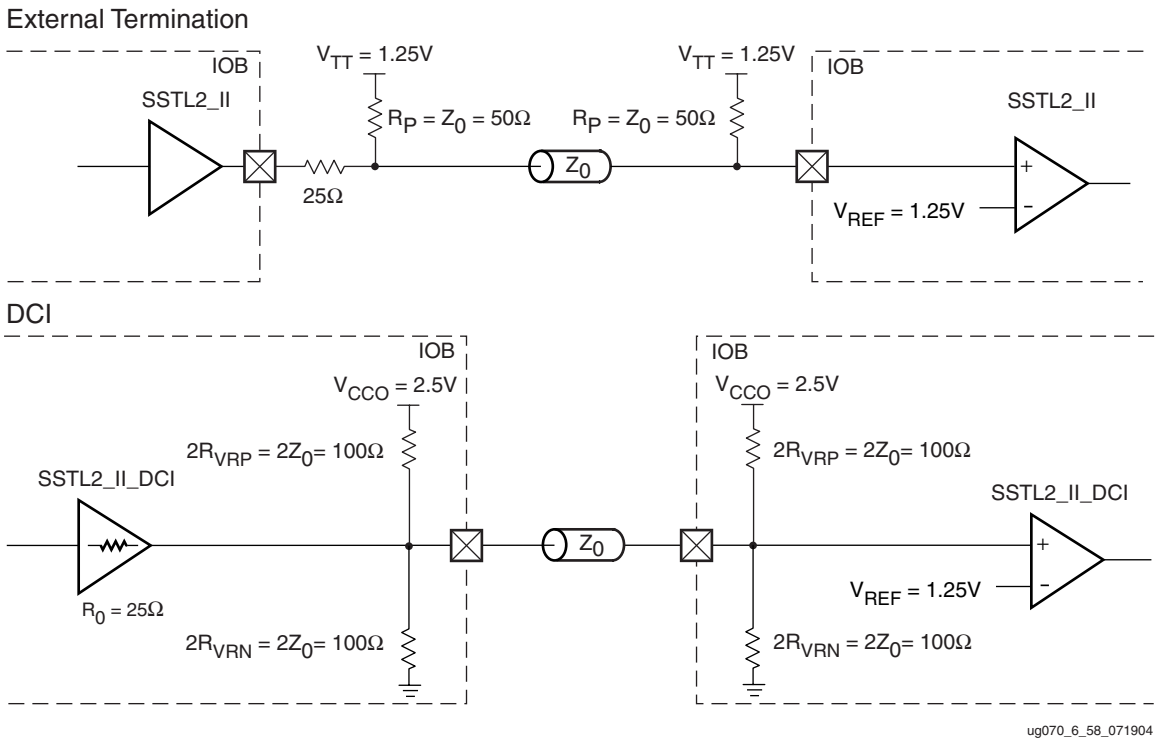


Figure 6-60: **SSTL2 Class II with Unidirectional Termination**

Figure 6-61 shows a sample circuit illustrating a valid bidirectional termination technique for SSTL2 Class II.

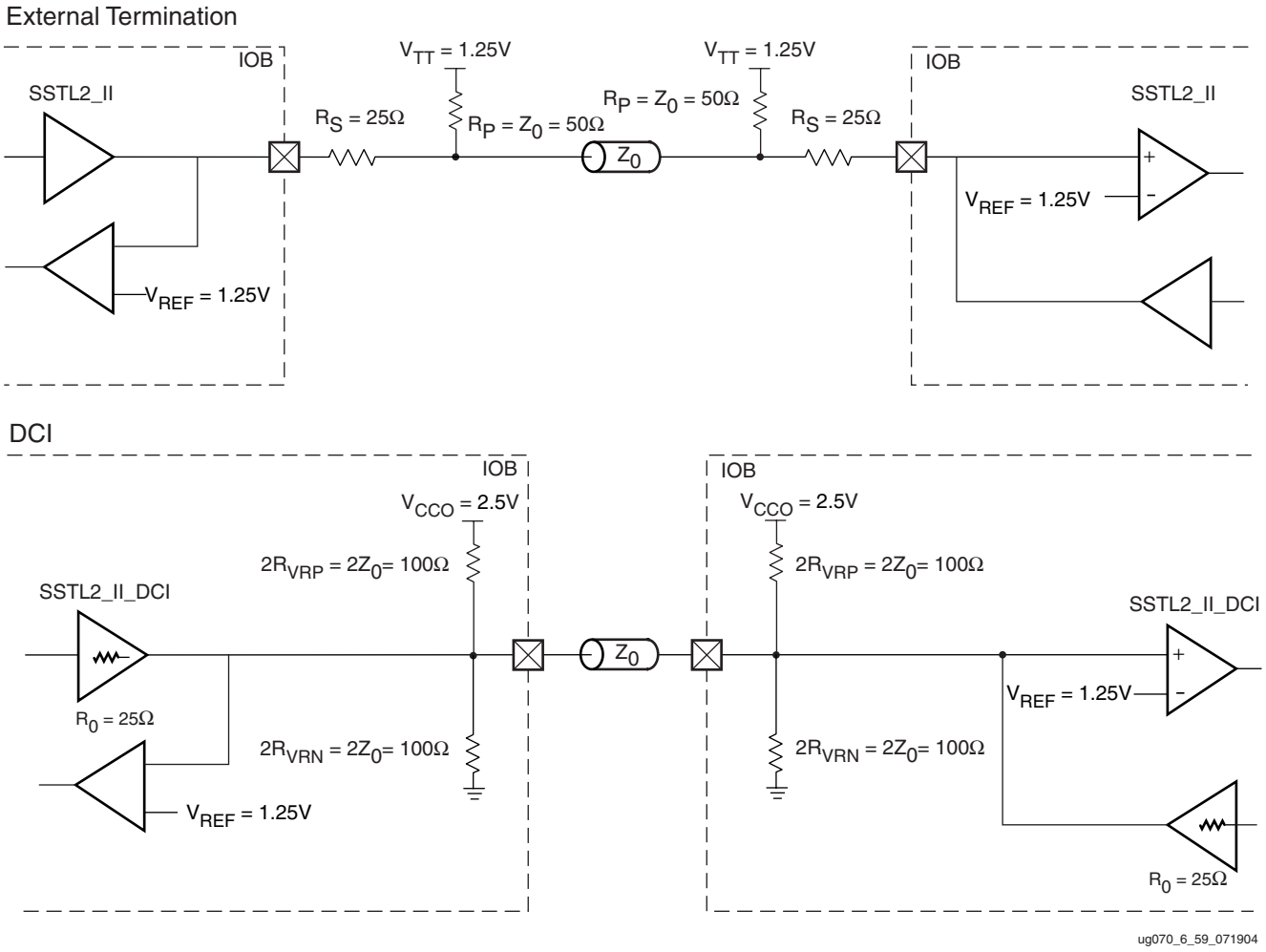


Figure 6-61: **SSTL2 Class II with Bidirectional Termination**

Table 6-27 lists the SSTL2 DC voltage specifications for Class II.

Table 6-27: **SSTL2 DC Voltage Specifications Class II**

	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
$V_{REF} = 0.5 \times V_{CCO}$	1.13	1.25	1.38
$V_{TT} = V_{REF} + N^{(1)}$	1.09	1.25	1.42
$V_{IH} \geq V_{REF} + 0.15$	1.28	1.40	3.0(2)
$V_{IL} \leq V_{REF} - 0.15$	–0.3(3)	1.1	1.27
$V_{OH} \geq V_{REF} + 0.8$	1.93	2.03	2.13
$V_{OL} \leq V_{REF} - 0.8^{(4)}$	0.36	0.46	0.55
I_{OH} at V_{OH} (mA)	–16.2	–	–

Table 6-27: **SSTL2 DC Voltage Specifications Class II *(Continued)***

	Min	Typ	Max
I_{OL} at V_{OL} (mA)	16.2	–	–

Notes:

1. N must be greater than or equal to –0.04 and less than or equal to 0.04.
2. V_{IH} maximum is V_{CCO} +0.3.
3. V_{IL} minimum does not conform to the formula.
4. Because SSTL2_I_DCI uses a controlled-impedance driver, V_{OH} and V_{OL} are different.

Complementary Single-Ended (CSE) Differential SSTL2 Class II (2.5V)

Figure 6-62 shows a sample circuit illustrating a valid termination technique for CSE differential SSTL2 Class II (2.5V) with unidirectional termination.

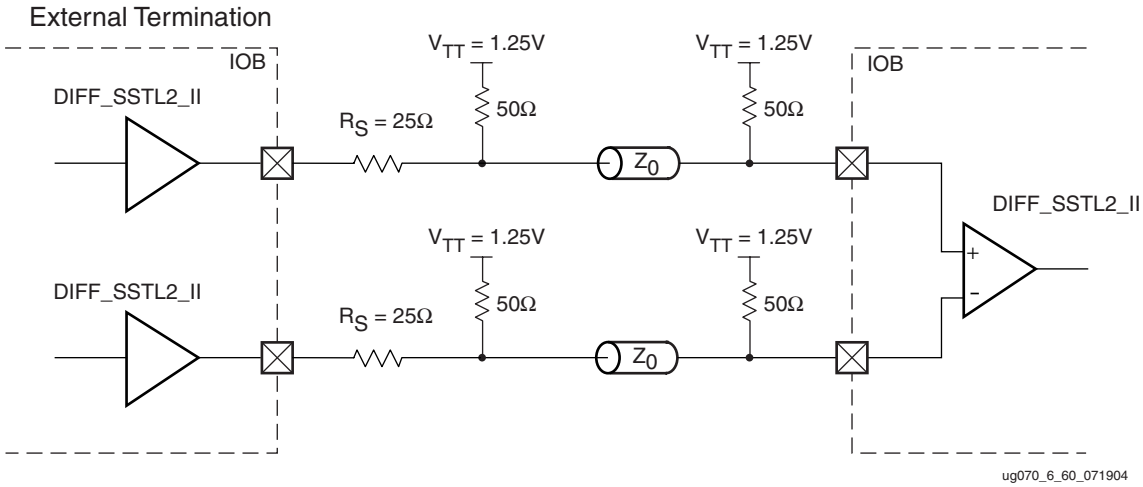


Figure 6-62: **Differential SSTL2 Class II Unidirectional Termination**

Figure 6-63 shows a sample circuit illustrating a valid termination technique for CSE differential SSTL2 Class II (2.5V) with unidirectional DCI termination.

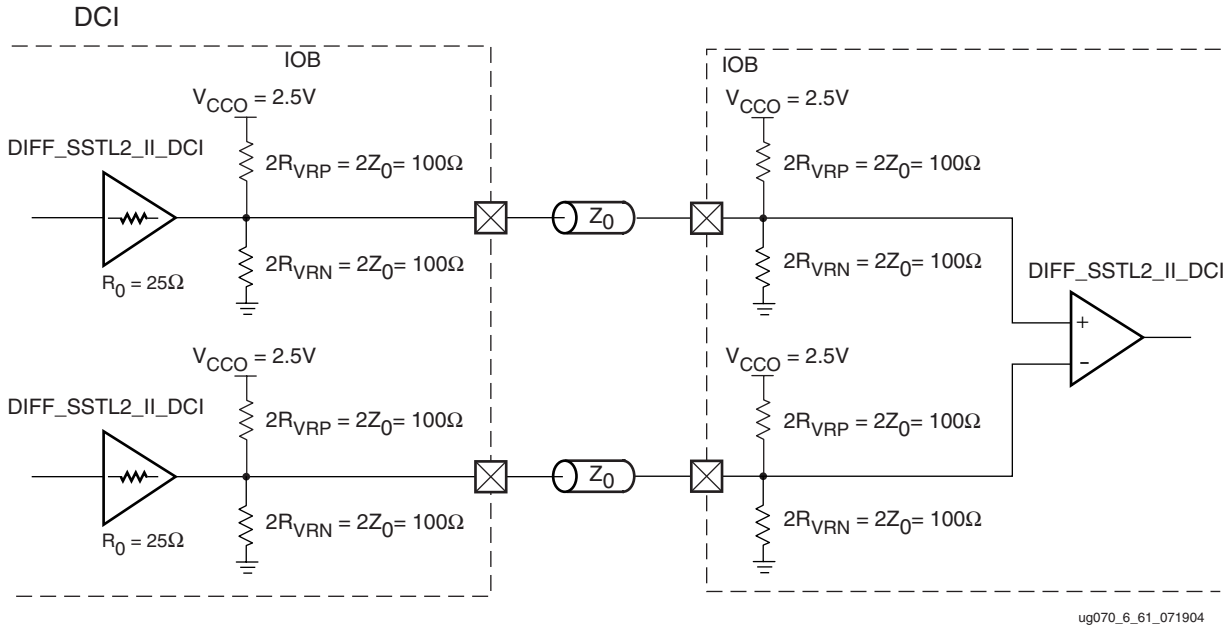


Figure 6-63: **Differential SSTL2 (2.5V) Class II Unidirectional DCI Termination**

Figure 6-64 shows a sample circuit illustrating a valid termination technique for CSE differential SSTL2 Class II (2.5V) with bidirectional termination.

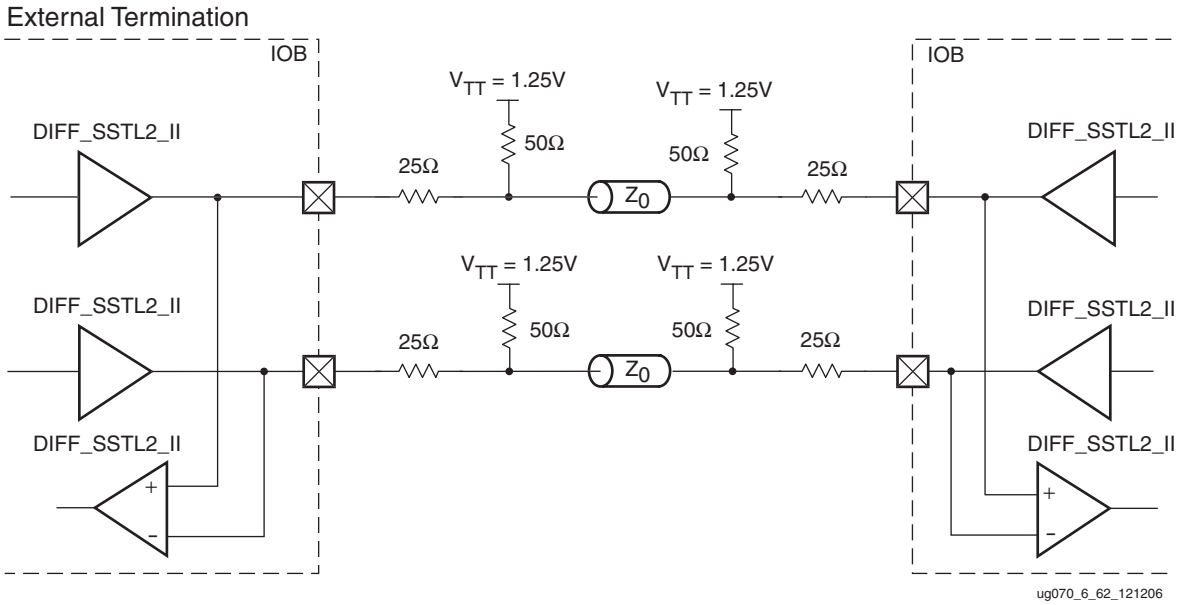


Figure 6-64: **Differential SSTL2 (2.5V) Class II with Bidirectional Termination**

Figure 6-65 shows a sample circuit illustrating a valid termination technique for CSE differential SSTL2 Class II (2.5V) with bidirectional DCI termination.

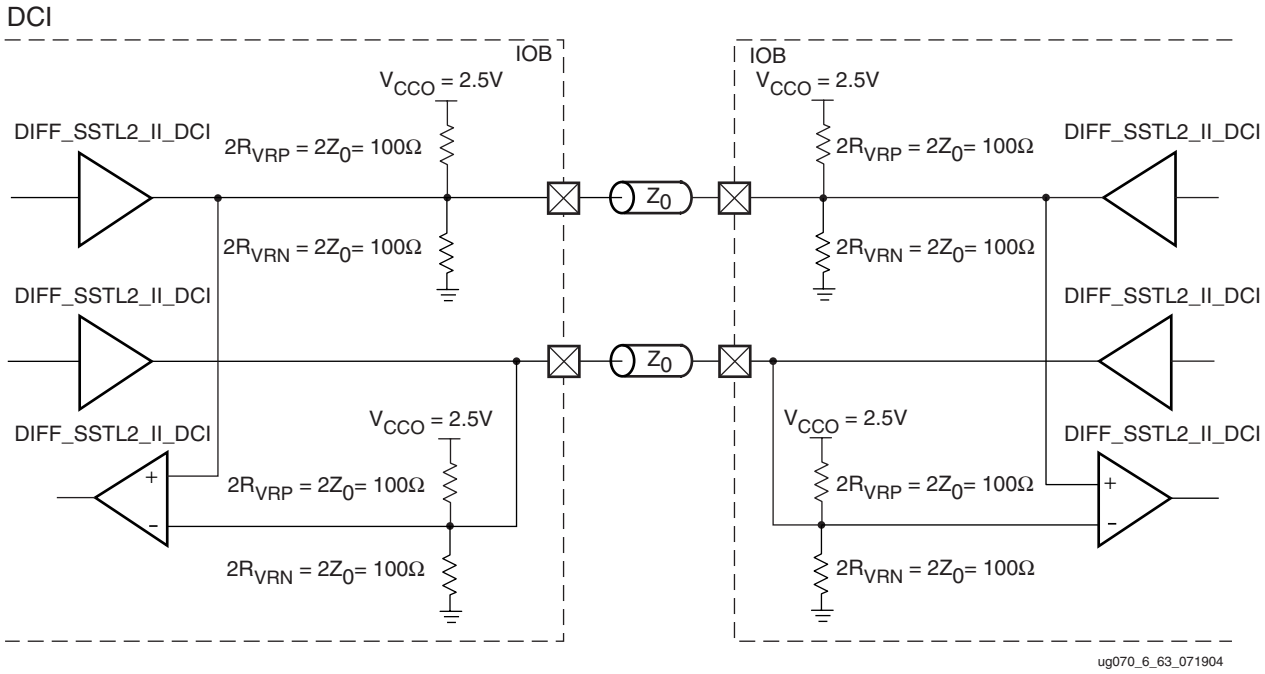


Figure 6-65: **Differential SSTL2 (2.5V) Class II with DCI Bidirectional Termination**

Table 6-28 lists the differential SSTL2 Class II DC voltage specifications.

Table 6-28: **Differential SSTL2 Class II DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
Input Parameters			
V_{TT}	–	$V_{CCO} \times 0.5$	–
V_{IN} (DC)(1)	–0.30	–	V_{CCO} + 0.30
V_{ID} (DC)(2)	0.3	–	V_{CCO} + 0.60
V_{ID} (AC)	0.62	–	V_{CCO} + 0.60
V_{IX} (AC)(3)	0.95	–	1.55
Output Parameters			
V_{OX} (AC)(4)	1.0	–	1.5

Notes:

1. V_{IN} (DC) specifies the allowable DC excursion of each differential input.
2. V_{ID} (DC) specifies the input differential voltage required for switching.
3. V_{IX} (AC) indicates the voltage where the differential input signals must cross.
4. V_{OX} (AC) indicates the voltage where the differential output signals must cross.

SSTL18 Class I (1.8V)

Figure 6-66 shows a sample circuit illustrating a valid termination technique for SSTL Class I (1.8V).

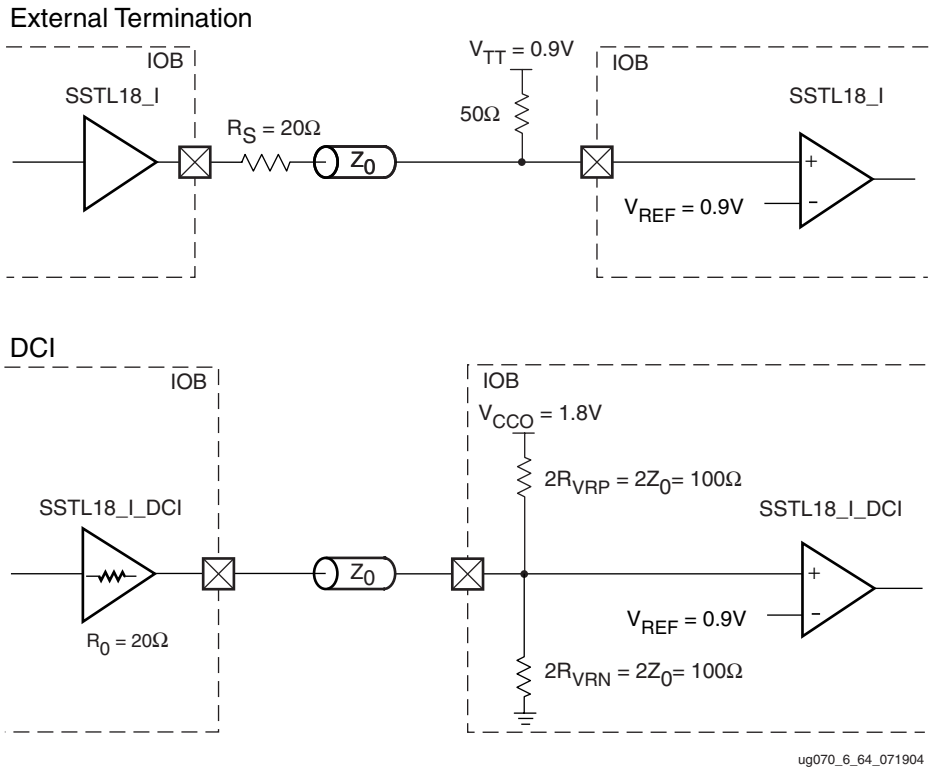


Figure 6-66: **SSTL18 (1.8V) Class I Termination**

SSTL18 Class II (1.8V)

Figure 6-67 shows a sample circuit illustrating a valid unidirectional termination technique for SSTL Class II (1.8V).

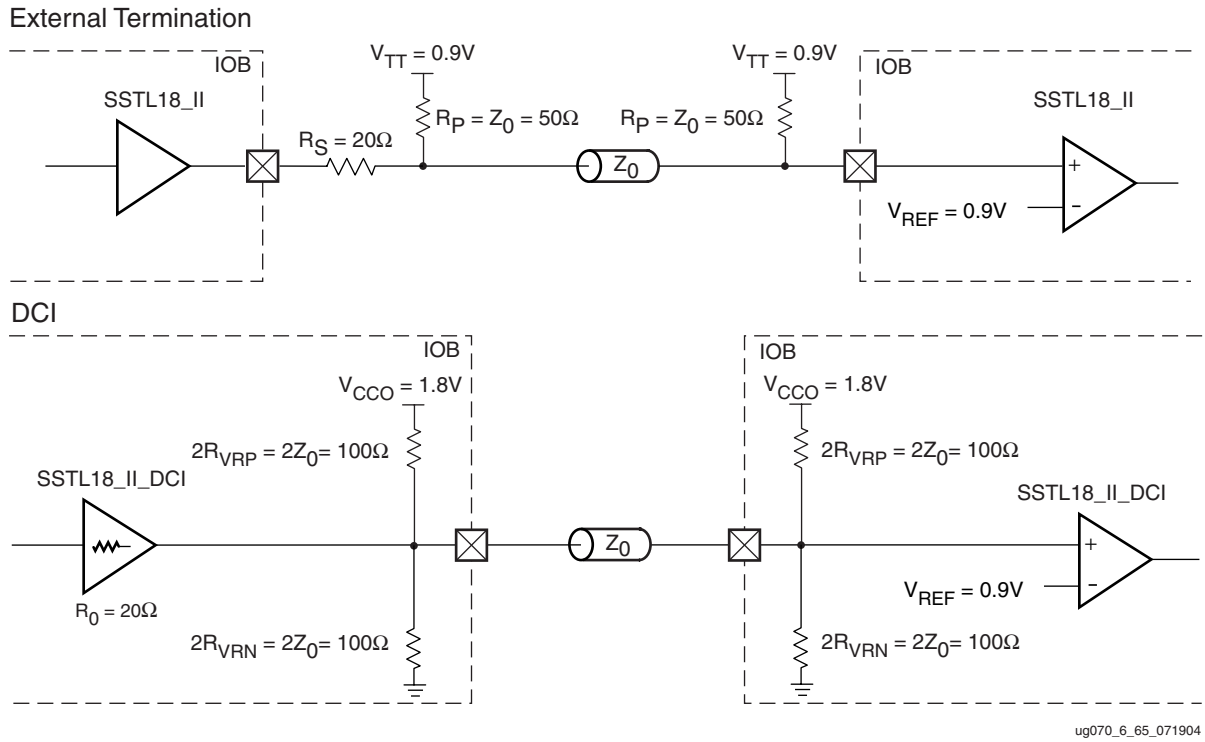


Figure 6-67: **SSTL18 (1.8V) Class II Unidirectional Termination**

Figure 6-68 shows a sample circuit illustrating a valid bidirectional termination technique for SSTL (1.8V) Class II.

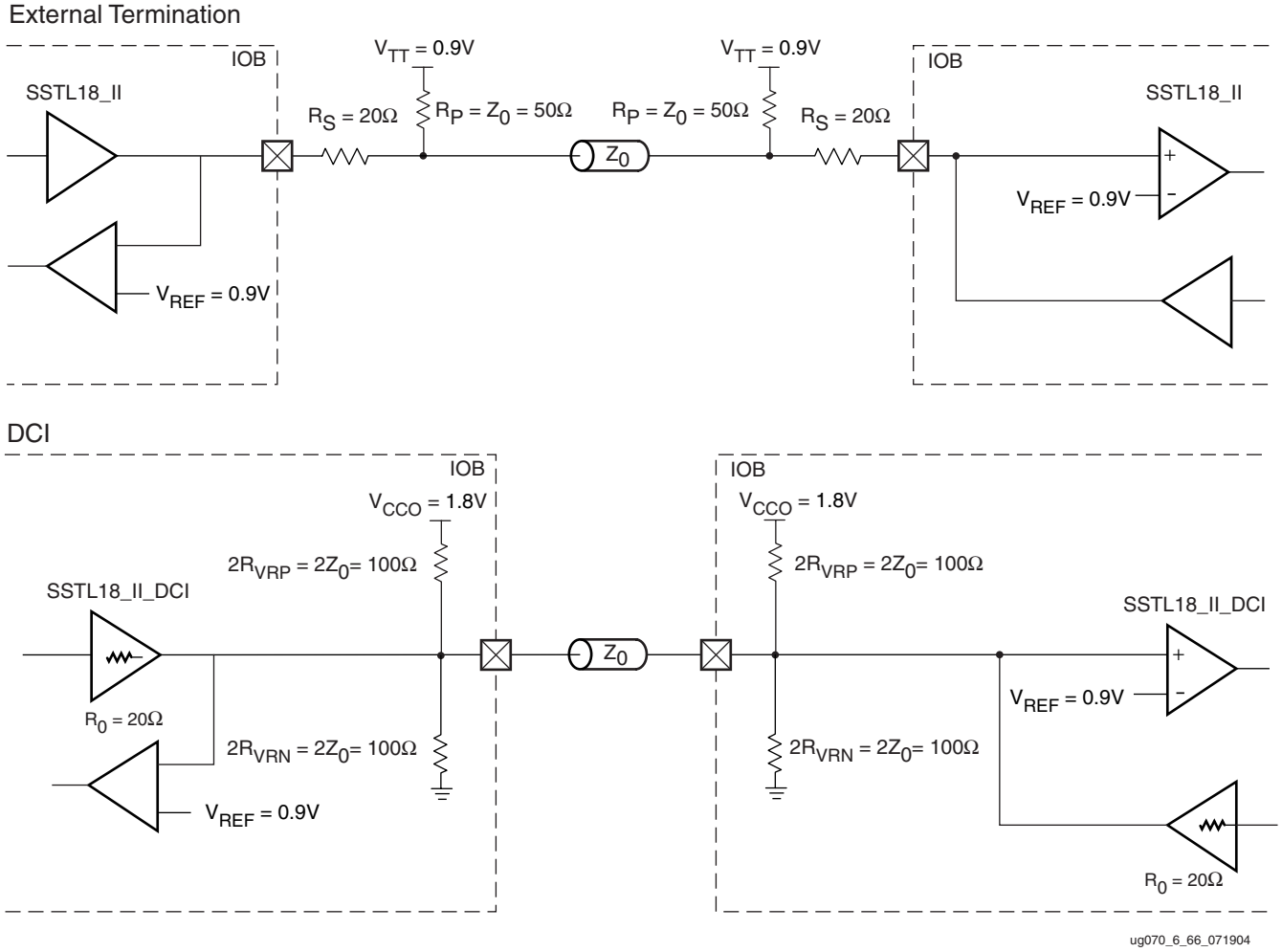


Figure 6-68: **SSTL (1.8V) Class II Termination**

Table 6-29 lists the SSTL (1.8V) DC voltage specifications.

Table 6-29: **SSTL (1.8V) DC Voltage Specifications**

	Class I			Class II		
	Min	**Typ**	**Max**	**Min**	**Typ**	**Max**
V_{CCO}	1.7	1.8	1.9	1.7	1.8	1.9
V_{REF} = 0.5 × V_{CCO}	0.833	0.9	0.969	0.833	0.9	0.969
V_{TT} = V_{REF} + N(1)	0.793	0.9	1.009	0.793	0.9	1.009
V_{IH} ≥ V_{REF} + 0.125	0.958	–	2.2(2)	0.958	–	2.2(2)
V_{IL} ≤ V_{REF} – 0.125	–0.3(3)	–	0.844	–0.3(3)	–	0.844
V_{OH} ≥ V_{TT} + 0.603(4)	1.396	–	–	1.396	–	–
V_{OL} ≤ V_{TT} – 0.603(4)	–	–	0.406	–	–	0.406

Table 6-29: **SSTL (1.8V) DC Voltage Specifications *(Continued)***

	Class I			Class II		
	Min	Typ	Max	Min	Typ	Max
I_{OH} at V_{OH} (mA)	–6.7	–	–	–13.4	–	–
I_{OL} at V_{OL} (mA)	6.7	–	–	13.4	–	–

Notes:

1. N must be greater than or equal to –0.04 and less than or equal to 0.04.
2. V_{IH} maximum is V_{CCO} +0.3.
3. V_{IL} minimum does not conform to the formula.
4. Because SSTL_I_DCI uses a controlled-impedance driver, V_{OH} and V_{OL} are different.

Complementary Single-Ended (CSE) Differential SSTL Class II (1.8V)

Figure 6-69 shows a sample circuit illustrating a valid termination technique for differential SSTL Class II (1.8V) with unidirectional termination.

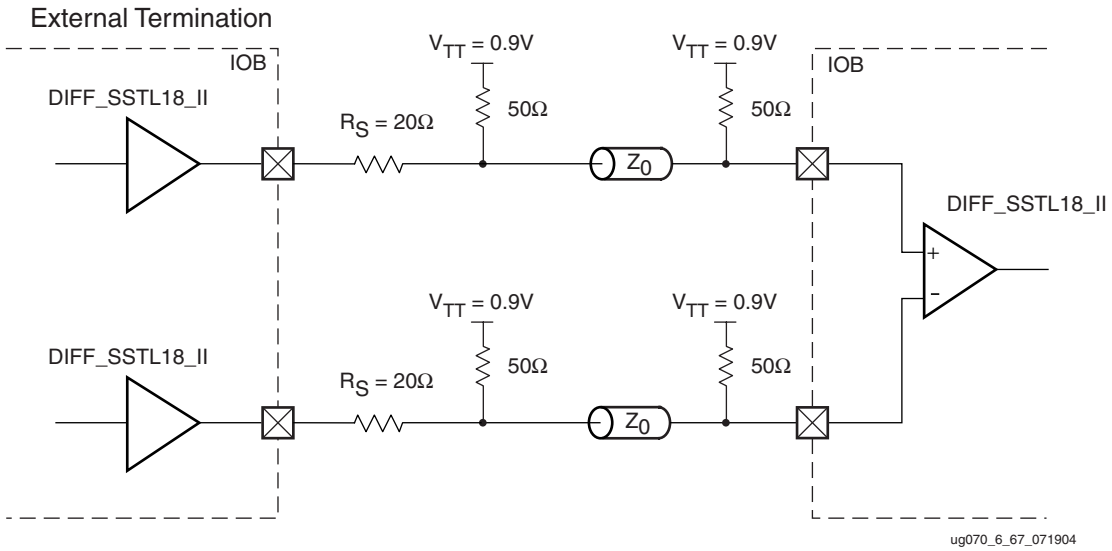


Figure 6-69: **Differential SSTL (1.8V) Class II Unidirectional Termination**

Figure 6-70 shows a sample circuit illustrating a valid termination technique for CSE differential SSTL Class II (1.8V) with unidirectional DCI termination.

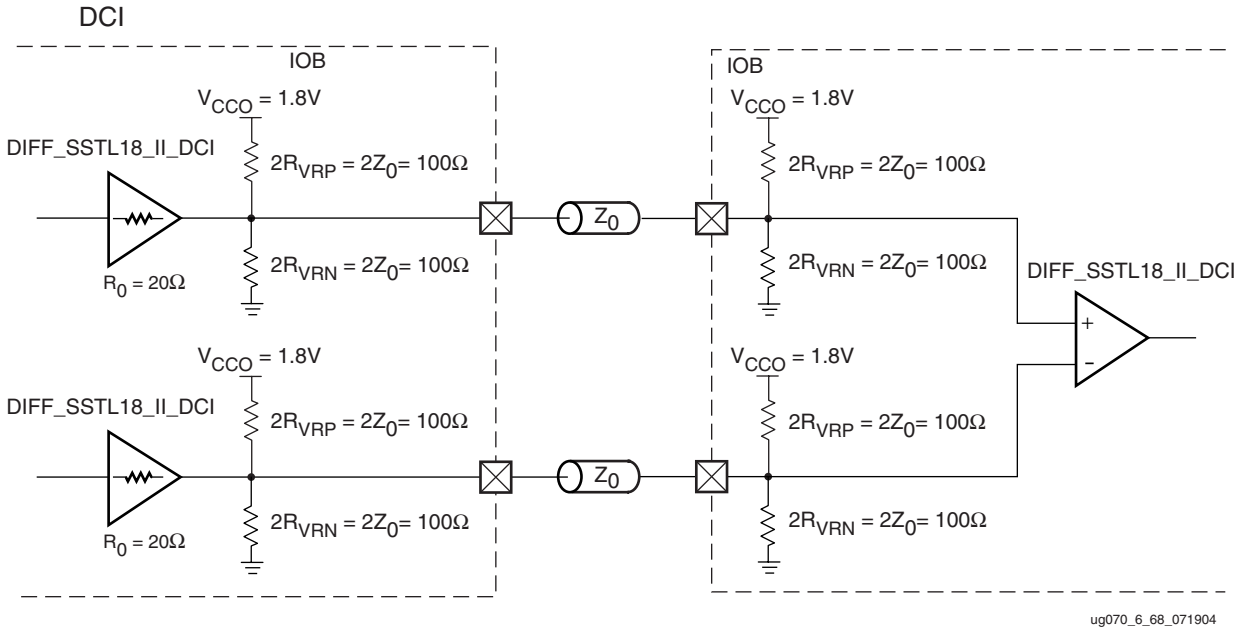


Figure 6-70: **Differential SSTL (1.8V) Class II Unidirectional DCI Termination**

Figure 6-71 shows a sample circuit illustrating a valid termination technique for CSE differential SSTL Class II (1.8V) with bidirectional termination.

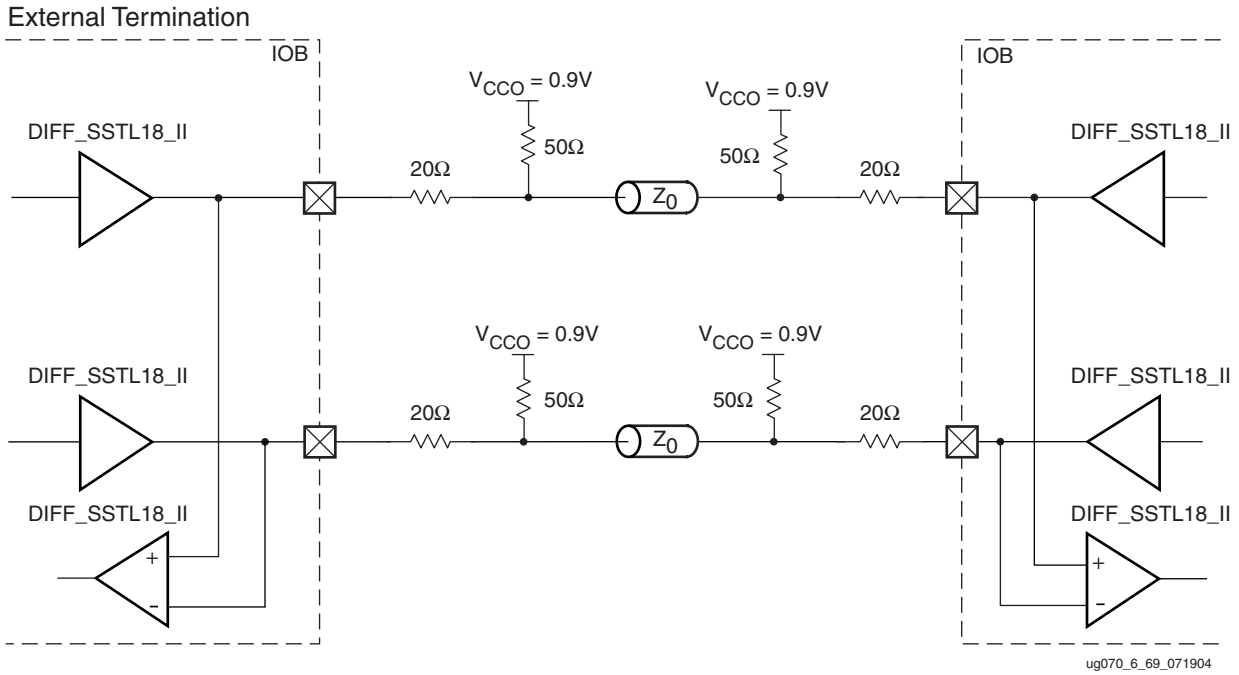


Figure 6-71: **Differential SSTL (1.8V) Class II with Bidirectional Termination**

Figure 6-72 shows a sample circuit illustrating a valid termination technique for CSE differential SSTL Class II (1.8V) with bidirectional DCI termination.

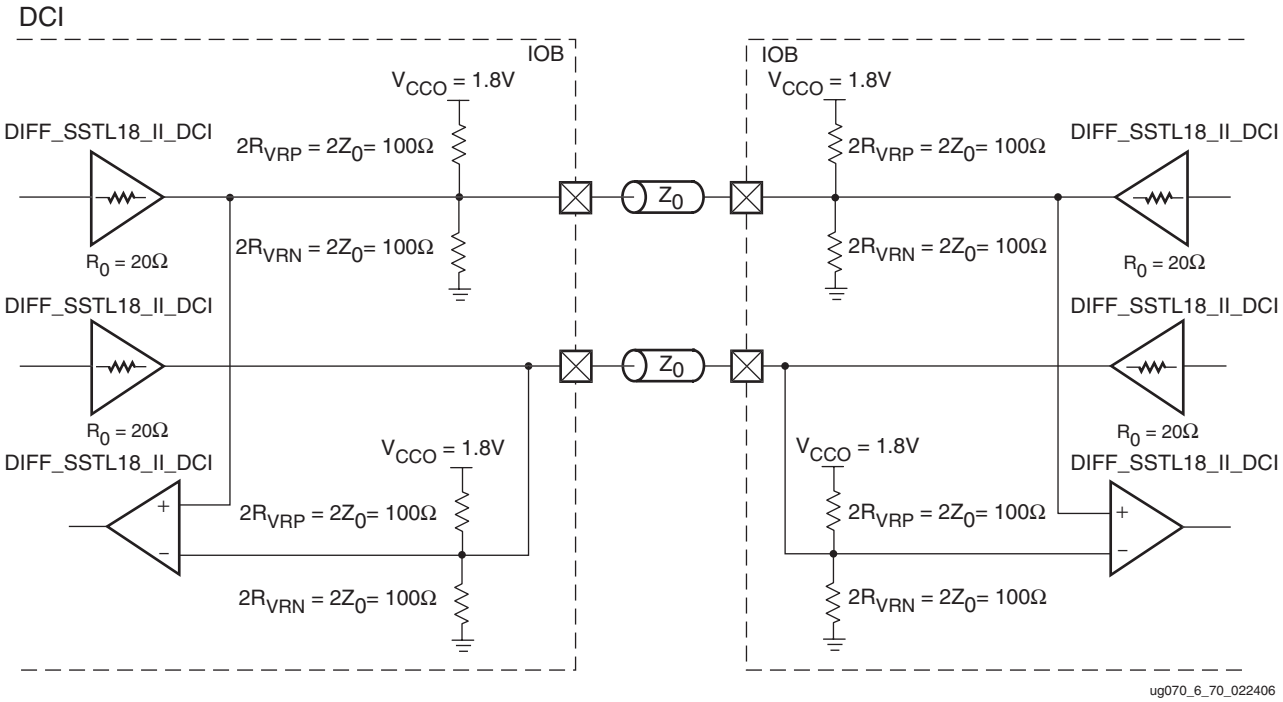


Figure 6-72: **Differential SSTL (1.8V) Class II with DCI Bidirectional Termination**

Table 6-30 lists the differential SSTL (1.8V) Class II DC voltage specifications.

Table 6-30: **Differential SSTL (1.8V) Class II DC Voltage Specifications**

	Min	Typ	Max
V_{CCO}	1.7	1.8	1.9
Input Parameters			
V_{TT}	–	$V_{CCO} \times 0.5$	–
V_{IN} (DC)(1)	–0.30	–	V_{CCO} + 0.30
V_{ID} (DC)(3)	0.25	–	V_{CCO} + 0.60
V_{ID} (AC)	0.50	–	V_{CCO} + 0.60
V_{IX} (AC)(4)	0.675	–	1.125
Output Parameters			
V_{OX} (AC)(5)	0.725	–	1.075

Notes:

1. V_{IN} (DC) specifies the allowable DC excursion of each differential input.
2. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."
3. V_{ID} (DC) specifies the input differential voltage required for switching.
4. V_{IX} (AC) indicates the voltage where the differential input signals must cross.
5. V_{OX} (AC) indicates the voltage where the differential output signals must cross.

Table 6-31 details the allowed attributes that can be applied to the SSTL I/O standards.

Table 6-31: **Allowed Attributes for the SSTL I/O Standards**

Attributes	Primitives		
	IBUFDS/IBUFGDS	OBUFDS/OBUFTDS	IOBUFDS
IOSTANDARD	All possible SSTL standards		
CAPACITANCE	LOW, NORMAL, DONT_CARE		

Table 6-32: **Allowed Attributes for the DIFF_SSTL I/O Standards**

Attributes	Primitives		
	IBUFDS/IBUFGDS	OBUFDS/OBUFTDS	IOBUFDS
IOSTANDARD	All possible DIFF_SSTL standards		
CAPACITANCE	LOW, NORMAL, DONT_CARE		

Differential Termination: DIFF_TERM Attribute

Virtex-4 FPGA IOBs provide a 100Ω differential termination across the input differential receiver terminals. This attribute is used in conjunction with LVDS_25, LVDSEXT_25, LDT_25, and ULVDS_25. It replaces the Virtex-II Pro FPGA LVDS_25_DT, LVDSEXT_25_DT, LDT_25_DT, and ULVDS_25_DT.

The on-chip input differential termination in Virtex-4 devices provides major advantages over the external resistor by removing the stub at the receiver completely and therefore greatly improving signal integrity:

- Consumes less power than DCI termination
- Does not use VRP/VRN pins (DCI)
- Supports LDT and ULVDS (not supported by DCI termination)

The V_{CCO} of the I/O bank must be connected to 2.5V ±5% to provide 100Ω of effective differential termination. DIFF_TERM is only available for inputs and can *only* be used with a bank voltage of V_{CCO} = 2.5V. The "Differential Termination Attribute" (DIFF_TERM) section outlines using this feature.

LVDS and Extended LVDS (Low Voltage Differential Signaling)

Low Voltage Differential Signaling (LVDS) is a very popular and powerful high-speed interface in many system applications. Virtex-4 FPGA I/Os are designed to comply with the EIA/TIA electrical specifications for LVDS to make system and board design easier. With the use of an LVDS current-mode driver in the IOBs, the need for external source termination in point-to-point applications is eliminated, and with the choice of an extended mode, Virtex-4 devices provide the most flexible solution for doing an LVDS design in an FPGA.

Extended LVDS provides a higher drive capability and voltage swing (350 - 750 mV), making it ideal for long-distance or cable LVDS links. The output AC characteristics of the LVDS extended mode driver are not within the EIA/TIA specifications. The LVDS extended mode driver is intended for situations requiring higher drive capabilities to produce an LVDS signal within the EIA/TIA specification at the receiver.

Transmitter Termination

The Virtex-4 FPGA LVDS transmitter does not require any external termination. Table 6-33 lists the allowed attributes corresponding to the Virtex-4 FPGA LVDS current-mode drivers. Virtex-4 FPGA LVDS current-mode drivers are a true current source and produce the proper (EIA/TIA compliant) LVDS signal.

Receiver Termination

LVDS_25_DCI, LVDSEXT_25_DCI Usage

LVDS_25_DCI and LVDSEXT_25_DCI provide split termination for the P and N inputs only. VRP and VRN should connect to 50Ω resistors.

Equivalently, it provides 100Ω differential impedance between the LVDS inputs.

Figure 6-73 and Figure 6-74 are examples of differential termination for an LVDS receiver on a board with 50Ω transmission lines.

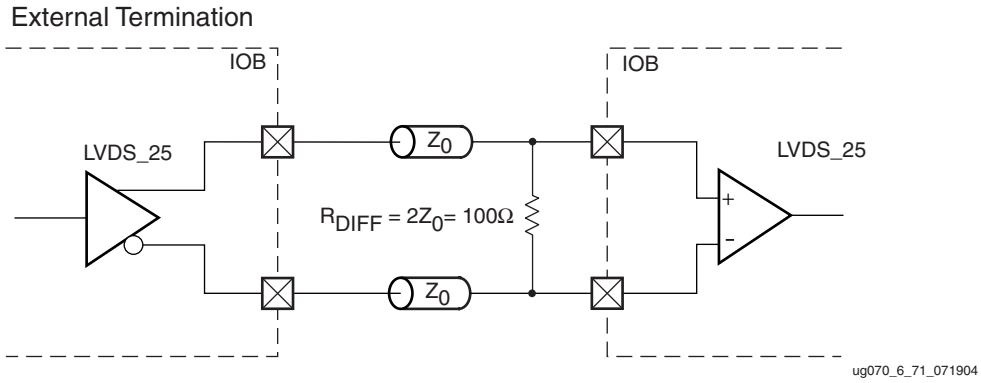


Figure 6-73: **LVDS_25 Receiver Termination**

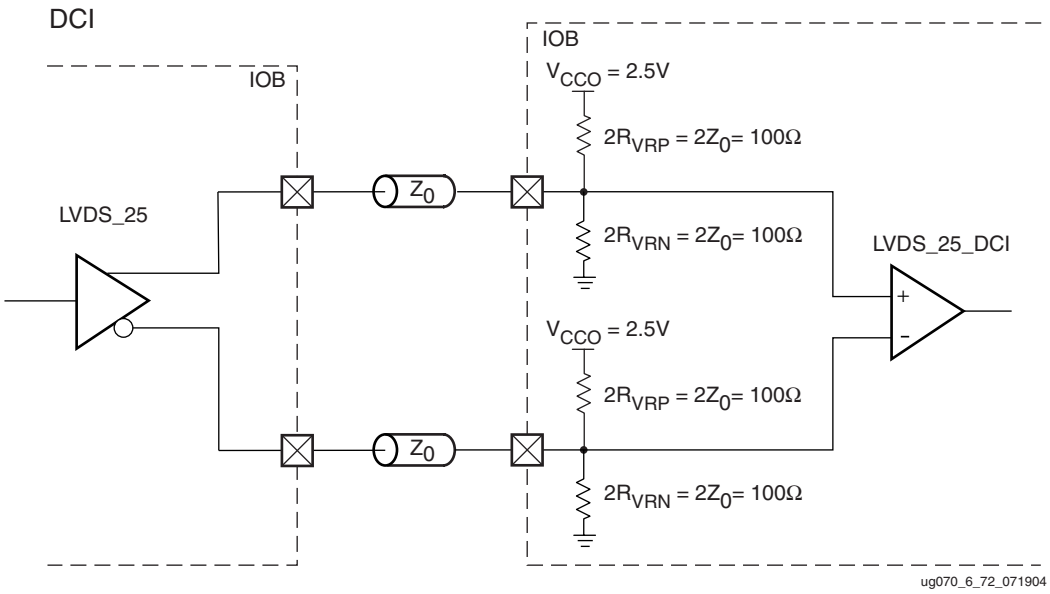


Figure 6-74: **LVDS_25_DCI Receiver Termination**

Figure 6-75 is an example of a differential termination for an LVDS receiver on a board with 50Ω transmission lines.

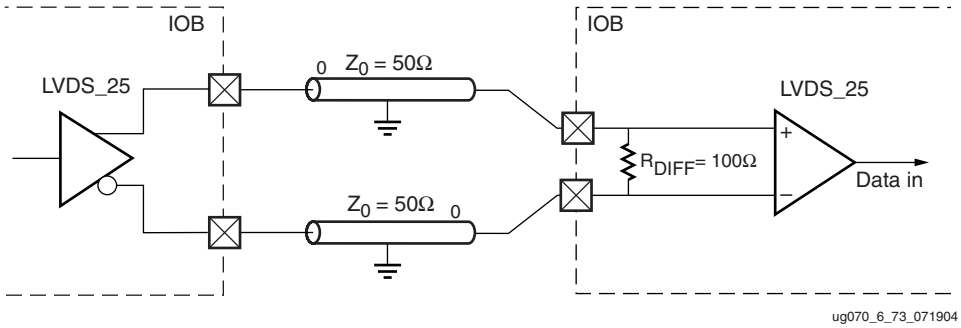


Figure 6-75: **LVDS_25 With DIFF_TERM Receiver Termination**

Table 6-33 lists the available Virtex-4 FPGA LVDS I/O standards and attributes supported.

Table 6-33: **Allowed Attributes of the LVDS I/O Standard**

Attributes	Primitives	
	IBUFDS/IBUFGDS	OBUFDS/OBUFTDS
IOSTANDARD	LVDS_25, LVDSEXT_25, ULVDS_25	
CAPACITANCE	LOW, NORMAL, DONT CARE	NORMAL
DIFF_TERM	TRUE, FALSE	Unused

Table 6-34 lists the available Virtex-4 FPGA LVDS DCI I/O standards and attributes supported.

Table 6-34: **Allowed Attributes of the LVDS DCI I/O Standard**

Attributes	Primitives	
	IBUFDS/IBUFGDS	OBUFDS/OBUFTDS
IOSTANDARD	LVDS_25_DCI LVDSEXT_25_DCI	Unused
CAPACITANCE	LOW, NORMAL, DONT CARE	Unused

HyperTransport Protocol (LDT)

The HyperTransport™ protocol or formally known as Lightning Data Transport (LDT) is a low-voltage standard for high-speed interfaces. Its differential signaling based interface is very similar to LVDS. Virtex-4 FPGA IOBs are equipped with LDT buffers. Table 6-35 summarizes all the possible LDT I/O standards and attributes supported.

Table 6-35: **Allowed Attributes of the LDT I/O Standard**

Attributes	Primitives	
	IBUFDS/IBUFGDS	OBUFDS/OBUFTDS
IOSTANDARD	LDT_25	
CAPACITANCE	LOW, NORMAL, DONT CARE	NORMAL
DIFF_TERM	TRUE, FALSE	Unused

BLVDS (Bus LVDS)

Since LVDS is intended for point-to-point applications, BLVDS is not an EIA/TIA standard implementation and requires careful adaptation of I/O and PCB layout design rules. The primitive supplied in the software library for bidirectional LVDS does not use the Virtex-4 FPGA LVDS current-mode driver. Therefore, source termination is required. Figure 6-76 shows the BLVDS transmitter termination.

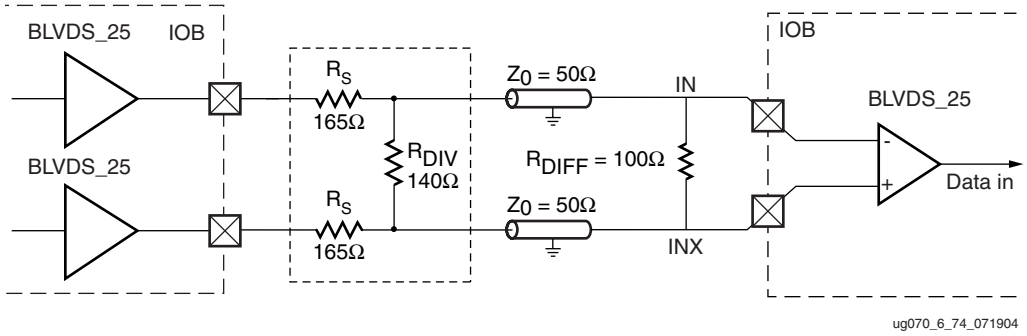


Figure 6-76: **BLVDS Transmitter Termination**

Table 6-36 summarizes all the possible BLVDS I/O standards and attributes supported.

Table 6-36: **Available BLVDS Primitives**

Attributes	Primitives		
	IBUFDS/IBUFGDS	OBUFDS/OBUFTDS	IOBUFDS
IOSTANDARD	BLVDS_25		
CAPACITANCE	LOW, NORMAL, DONT_CARE	NORMAL	LOW, NORMAL, DONT_CARE

CSE Differential LVPECL (Low-Voltage Positive Emitter-Coupled Logic)

LVPECL is a very popular and powerful high-speed interface in many system applications. Virtex-4 FPGA I/Os are designed to comply with the EIA/TIA electrical specifications for 2.5V LVPECL to make system and board design easier.

LVPECL Transceiver Termination

The Virtex-4 FPGA LVPECL transmitter and receiver requires the termination shown in Figure 6-77, illustrating a Virtex-4 FPGA LVPECL transmitter and receiver on a board with 50Ω transmission lines. The LVPECL driver is composed of two LVCMOS drivers that

when combined with the three resistor output termination circuit form a compliant LVPECL output.

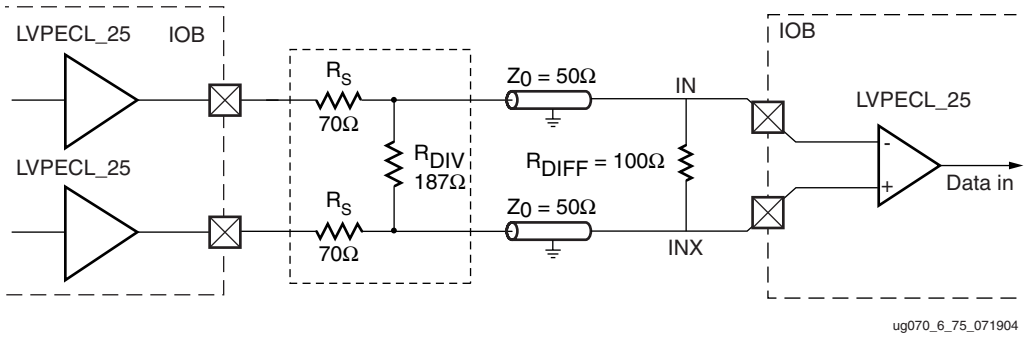


Figure 6-77: **LVPECL Transmitter Termination**

Table 6-37 summarizes all the possible LVPECL I/O standards and attributes supported.

Table 6-37: **Available LVPECL Primitives**

Attributes	Primitives		
	IBUFDS/IBUFGDS	**OBUFDS/OBUFTDS**	**IOBUFDS**
IOSTANDARD	LVPECL		
CAPACITANCE	LOW, NORMAL, DONT_CARE	NORMAL	LOW, NORMAL, DONT_CARE

I/O Standards Compatibility

Table 6-38 summarizes the Virtex-4 FPGA supported I/O standards.

Table 6-38: **I/O Compatibility**

I/O Standard	V_{CCO}		V_{REF}	Termination Type		Lower Capacitance IOB	
	Output	Input	Input	Output	Input	Output	Input
LVTTL(1)	3.3	3.3	N/R	N/R	N/R	Yes	Yes
LVCMOS33(1)			N/R	N/R	N/R	Yes	Yes
LVDCI_33(1)			N/R	Series	N/R	Yes	Yes
HSLVDCI_33(1)			$V_{CCO}/2$	Series	N/R	Yes	Yes
PCIX(1)			N/R	N/R	N/R	Yes	Yes
PCI33_3(1)			N/R	N/R	N/R	Yes	Yes
PCI66_3(1)			N/R	N/R	N/R	Yes	Yes
LVDS_25	2.5	Note (2)	N/R	N/R	N/R	No	Yes
LVDSEXT_25			N/R	N/R	N/R	No	Yes
LDT_25			N/R	N/R	N/R	No	Yes
ULVDS_25			N/R	N/R	N/R	No	Yes
RSDS_25(4)			N/R	N/R	N/R	No	Yes
BLVDS_25			N/R	N/R	N/R	Yes	Yes
LVPECL_25			N/R	N/R	N/R	Yes	Yes
SSTL2_I			1.25	N/R	N/R	Yes	Yes
SSTL2_II			1.25	N/R	N/R	Yes	Yes
DIFF_SSTL2_II			N/R	N/R	N/R	Yes	Yes
LVCMOS25		2.5	N/R	N/R	N/R	Yes	Yes
LVDCI_25			N/R	Series	N/R	Yes	Yes
HSLVDCI_25			$V_{CCO}/2$	Series	N/R	Yes	Yes
LVDCI_DV2_25			N/R	Series	N/R	Yes	Yes
LVDS_25_DCI			N/R	N/R	Split	No	Yes
LVDSEXT_25_DCI			N/R	N/R	Split	No	Yes
SSTL2_I_DCI			1.25	N/R	Split	Yes	Yes
SSTL2_II_DCI			1.25	Split	Split	Yes	Yes
DIFF_SSTL2_II_DCI			N/R	Split	Split	Yes	Yes

Table 6-38: **I/O Compatibility *(Continued)***

I/O Standard	V_{CCO}		V_{REF}	Termination Type		Lower Capacitance IOB	
	Output	Input	Input	Output	Input	Output	Input
HSTL_III_18	1.8	Note (2)	1.1	N/R	N/R	Yes	Yes
HSTL_IV_18			1.1	N/R	N/R	Yes	Yes
HSTL_I_18			0.9	N/R	N/R	Yes	Yes
HSTL_II_18			0.9	N/R	N/R	Yes	Yes
DIFF_HSTL_II_18			N/R	N/R	N/R	Yes	Yes
SSTL18_I			0.9	N/R	N/R	Yes	Yes
SSTL18_II			0.9	N/R	N/R	Yes	Yes
DIFF_SSTL18_II			N/R	N/R	N/R	Yes	Yes
LVCMOS18		1.8	N/R	N/R	N/R	Yes	Yes
LVDCI_18			N/R	Series	N/R	Yes	Yes
HSLVDCI_18			$V_{CCO}/2$	Series	N/R	Yes	Yes
LVDCI_DV2_18			N/R	Series	N/R	Yes	Yes
HSTL_III_18_DCI			1.1	N/R	Single	Yes	Yes
HSTL_IV_18_DCI			1.1	Single	Single	Yes	Yes
HSTL_I_18_DCI			0.9	N/R	Split	Yes	Yes
HSTL_II_18_DCI			0.9	Split	Split	Yes	Yes
DIFF_HSTL_II_18_DCI			N/R	Split	Split	Yes	Yes
SSTL18_I_DCI			0.9	N/R	Split	Yes	Yes
SSTL18_II_DCI			0.9	Split	Split	Yes	Yes
DIFF_SSTL18_II_DCI			N/R	Split	Split	Yes	Yes

Table 6-38: **I/O Compatibility *(Continued)***

I/O Standard	V_{CCO}		V_{REF}	Termination Type		Lower Capacitance IOB	
	Output	Input	Input	Output	Input	Output	Input
HSTL_III	1.5	Note (2)	0.9	N/R	N/R	Yes	Yes
HSTL_IV			0.9	N/R	N/R	Yes	Yes
HSTL_I			0.75	N/R	N/R	Yes	Yes
HSTL_II			0.75	N/R	N/R	Yes	Yes
DIFF_HSTL_II			N/R	N/R	N/R	Yes	Yes
LVCMOS15		1.5	N/R	N/R	N/R	Yes	Yes
LVDCI_15			N/R	Series	N/R	Yes	Yes
HSLVDCI_15			$V_{CCO}/2$	Series	N/R	Yes	Yes
LVDCI_DV2_15			N/R	Series	N/R	Yes	Yes
GTLP_DCI			1	Single	Single	Yes	Yes
HSTL_III_DCI			0.9	N/R	Single	Yes	Yes
HSTL_IV_DCI			0.9	Single	Single	Yes	Yes
HSTL_I_DCI			0.75	N/R	Split	Yes	Yes
HSTL_II_DCI			0.75	Split	Split	Yes	Yes
DIFF_HSTL_II_DCI			N/R	Split	Split	Yes	Yes
GTL_DCI	1.2	1.2	0.8	Single	Single	Yes	Yes
GTLP	N/R	Note (2)	1	N/R	N/R	Yes	Yes
GTL			0.8	N/R	N/R	Yes	Yes

Notes:

1. See "3.3V I/O Design Guidelines" for more detailed information
2. Differential inputs and inputs using V_{REF} are powered from V_{CCAUX}. However, pin voltage must not exceed V_{CCO}, due to the presence of clamp diodes to V_{CCO}.
3. N/R = no requirement.
4. RSDS_25 has the same DC specifications as LVDS_25. All information pertaining to LVDS_25 is applicable to RSDS_25.
5. I/O standard is selected using the IOSTANDARD attribute.

I/O Standards Special Design Rules

Rules for Combining I/O Standards in the Same Bank

The following rules must be obeyed to combine different input, output, and bi-directional standards in the same bank:

1. **Combining output standards only.** Output standards with the same output V_{CCO} requirement can be combined in the same bank.

   Compatible example:

   SSTL2_I and LVDCI_25 outputs

   Incompatible example:

   SSTL2_I (output V_{CCO} = 2.5V) and
   LVCMOS33 (output V_{CCO} = 3.3V) outputs

2. **Combining input standards only.** Input standards with the same input V_{CCO} and input V_{REF} requirements can be combined in the same bank.

   Compatible example:

   LVCMOS15 and HSTL_IV inputs

   Incompatible example:

   LVCMOS15 (input V_{CCO} = 1.5V) and
   LVCMOS18 (input V_{CCO} = 1.8V) inputs

   Incompatible example:

   HSTL_I_DCI_18 (V_{REF} = 0.9V) and
   HSTL_IV_DCI_18 (V_{REF} = 1.1V) inputs

3. **Combining input standards and output standards.** Input standards and output standards with the same input V_{CCO} and output V_{CCO} requirement can be combined in the same bank.

   Compatible example:

   LVDS_25 output and HSTL_I input

   Incompatible example:

   LVDS_25 output (output V_{CCO} = 2.5V) and
   HSTL_I_DCI_18 input (input V_{CCO} = 1.8V)

4. **Combining bi-directional standards with input or output standards.** When combining bi-directional I/O with other standards, make sure the bi-directional standard can meet the first three rules.

5. **Additional rules for combining DCI I/O standards.**

   a. No more than one Single Termination type (input or output) is allowed in the same bank.

      Incompatible example:

      HSTL_IV_DCI input and HSTL_III_DCI input

   b. No more than one Split Termination type (input or output) is allowed in the same bank.

      Incompatible example:

      HSTL_I_DCI input and HSTL_II_DCI input

The implementation tools enforce the above design rules.

3.3V I/O Design Guidelines

To achieve maximum performance in Virtex-4 devices, several 3.3V I/O design guidelines and techniques are highlighted in this section. This includes managing overshoot/undershoot with termination techniques, regulating V_{CCO} at 3.0V with a voltage regulator, using external bus switches, reviewing configuration methods, and other design considerations.

I/O Standard Design Rules

Overshoot/Undershoot

Undershoot and overshoot voltages on I/Os operating at 3.3V should not exceed the absolute maximum ratings of –0.3V to 4.05V, respectively, when V_{CCO} is 3.75V. These absolute maximum limits are stated in the absolute maximum ratings table in Table 6-38 of the *Virtex-4 Data Sheet*. However, the maximum undershoot value is directly affected by the value of V_{CCO}. Table 6-38 describes the worst-case undershoot and overshoot at different V_{CCO} levels.

The voltage across the gate oxide at any time must not exceed 4.05V. Consider the case in which the I/O is either an input or a 3-stated buffer as shown in Figure 6-78. The gate of the output PMOS transistor P_0 and NMOS transistor N_0 is connected essentially to V_{CCO} and ground, respectively.

The amount of undershoot allowed without overstressing the PMOS transistor P_0 is the gate voltage minus the gate oxide limit, or V_{CCO} – 4.05V.

Similarly, the absolute maximum overshoot allowed without overstressing the NMOS transistor N_0 is the gate voltage plus the gate oxide limit, or Ground + 4.05V.

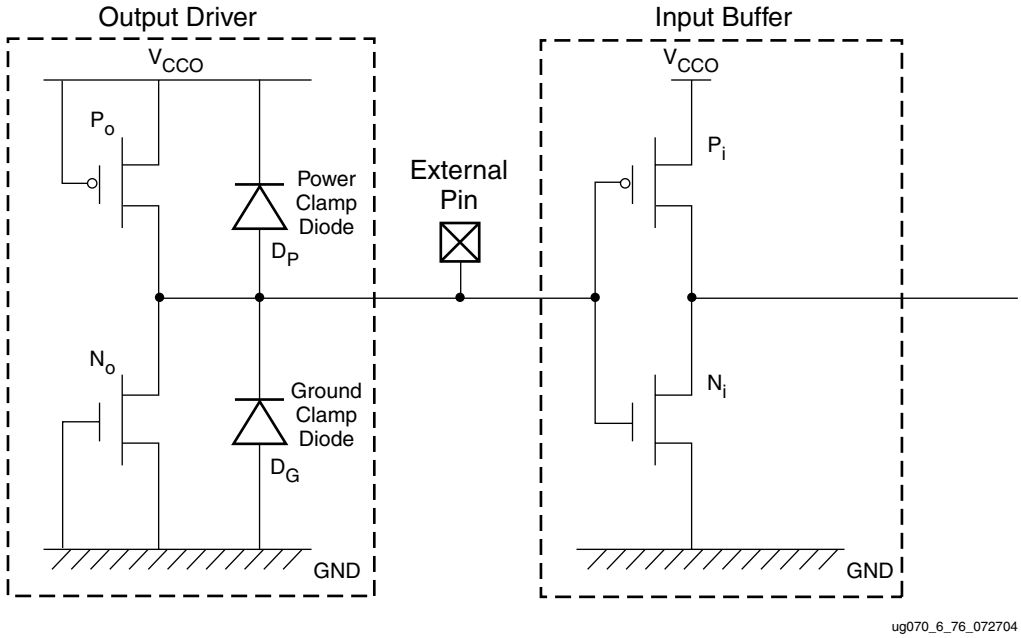


Figure 6-78: **Virtex-4 FPGA I/O: 3-State Output Driver**

Table 6-39: **Absolute Maximum Undershoot and Overshoot**

V_{CCO} (V)	Maximum Undershoot (V)	Maximum Overshoot (V)
3.75	–0.30	4.05
3.6	–0.45	4.05
3.45	–0.60	4.05
3.3	–0.75	4.05
3.0	–1.05	4.05

The clamp diodes offer protection against transient voltage beyond approximately V_{CCO} + 0.5V and Ground – 0.5V. The voltage across the diode increases proportionally to the current going through it. Therefore the clamped level is not fixed and can vary depending on the board design. The absolute maximum I/O limits might be exceeded even if the clamp diode is active.

The IBIS models contain the voltage-current characteristics of the I/O drivers and clamp diodes.

To verify overshoot and undershoot are within the I/O absolute maximum specifications, Xilinx recommends proper I/O termination and performing IBIS simulation.

Source Termination and LVDCI_33

In general, the I/O drivers should match the board trace impedance to within ±10% to minimize overshoot and undershoot. Source termination is often used for unidirectional interfaces. The DCI feature has built-in source termination on all user output pins. It compensates for impedance changes due to voltage and/or temperature fluctuations, and can match the reference resistor values. Assuming the reference resistor values are the same as the board trace impedance, the output impedance of the driver will closely match with the board trace.

The LVDCI_33 standard is used to enable the DCI features for 3.3V I/O operations. As shown in Figure 6-79, the OBUF_LVDCI_33 primitive is used to implement the source termination function in Virtex-4 FPGA output drivers. The pull-up resistor connected to VRN and the pull-down resistor connected to VRP determine the output impedance of all the output drivers in the same bank. The "Virtex-4 FPGA Digitally Controlled Impedance (DCI)" section has more details on using DCI.

Since the LVDCI_33 standard does not offer input termination, source termination must be implemented on the driver side. Figure 6-79 shows the recommended external source termination resistors to be incorporated on the external device side.

The total impedance of the LVTTL/LVCMOS driver added to the series termination resistor R_0 must match the board trace impedance ±10 percent to minimize overshoot and undershoot. An IBIS simulation is advised for calculating the exact value needed for R_0.

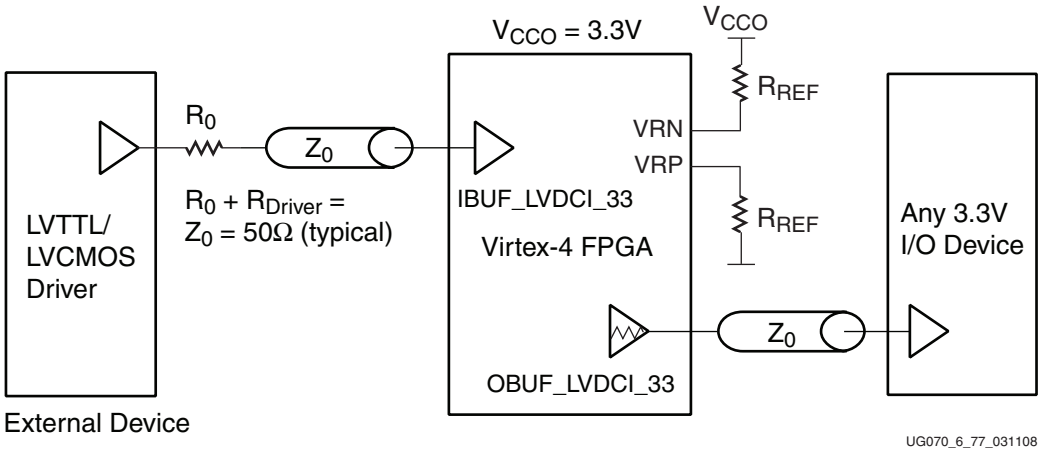


Figure 6-79: **Connecting LVTTL or LVCMOS Using the LVDCI_33 Standard**

The connection scheme shown in Figure 6-80 is for a bidirectional bus scenario. The signal performance may be degraded by R_0. Therefore, it is also recommended to verify the R_0 value and performance with an IBIS simulation.

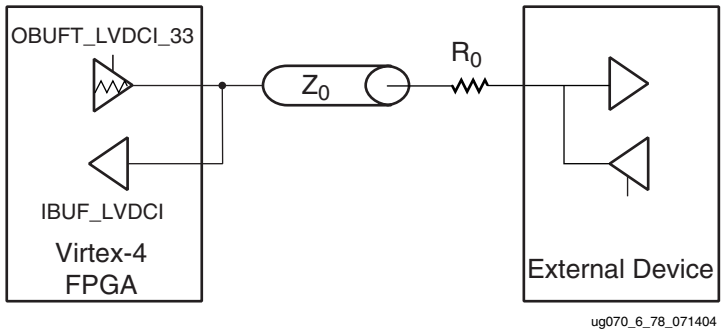


Figure 6-80: **3.3V I/O Configuration**

When designing with the LVDCI_33 standard:

- The output drive strength and slew rates are not programmable. The output impedance references the VRP and VRN resistors, and the output current is determined by the output impedance.
- If only LVDCI_33 inputs are used, it is not necessary to connect VRP and VRN to external reference resistors. The implementation pad report does not record VRP and VRN being used. External reference resistors are required only if LVDCI_33 outputs are present in a bank.
- LVDCI_33 is compatible with LVTTL and LVCMOS standards only.

In addition, changing the slew rate from fast to slow and/or reducing the current drive could significantly reduce overshoot and undershoot.

The *Virtex-4 PCB Designer's Guide* contains additional design information to assist PCB designers and signal integrity engineers.

Regulating V_{CCO} at 3.0V

The following section discusses alternatives for managing overshoot and undershoot for LVTTL, LVCMOS33, and PCI applications.

When V_{CCO} is lowered to 3.0V, the power clamp diode turns on at about 3.5V. Therefore it limits any overshoot higher than 3.5V before reaching the absolute maximum level of 4.05V. In addition, instead of –0.3V when V_{CCO} = 3.75V, the lower absolute maximum limit corresponding to V_{CCO} = 3.0V is –1.05V. In this case, the ground clamp diode clips undershoot before reaching the lower absolute maximum limit.

As a result, lowering V_{CCO} to 3.0V addresses the overshoot and undershoot specifications for all supported 3.3 V standards, including LVCMOS_33, LVTTL, LVDCI_33, and PCI.

Mixing Techniques

Either using LVDCI_33 standard or lowering the V_{CCO} to 3.0V is a good approach to address overshoot and undershoot. It is also acceptable to combine both methods. When V_{CCO} is lowered to 3.0V, it is not necessary to adjust the reference resistors VRP and VRN. The VRP and VRN values should always be the same as the board trace impedance.

Summary

Virtex-4 devices support 3.3V I/O standards (LVTTL, LVCMOS33, LVDCI33, PCI33/66, and PCI-X) when the following guidelines are met:

- Keep signal overshoot and undershoot within the absolute maximum FPGA device specifications.
  - Source termination using LVDCI_33
  - Slow slew rate and/or reduced drive current
  - Voltage regulation at 3.0V
  - External high-speed bus switches
- The absolute maximum junction temperature (T_J) is 125°C for 3.3V I/O operation.

Simultaneous Switching Output Limits

When multiple output drivers change state at the same time, power supply disturbance occurs. These disturbances can cause undesired transient behavior in output drivers, input receivers, or in internal logic. These disturbances are often referred to as Simultaneous-Switching Output (SSO) noise. The SSO limits govern the number and type of I/O output drivers that can be switched simultaneously while maintaining a safe level of SSO noise.

The Virtex-4 FPGA SSO limits are divided into two categories: Sparce Chevron (SC) and Non-Sparse Chevron (NSC), corresponding to package pinout style. SSO limits for SC packages are simpler and less restrictive than for NSC packages.

Sparse-Chevron Packages

Virtex-4 FPGA packaging falls into two categories according to pinout: sparse-chevron and non-sparse-chevron. The sparse-chevron pinout style is an improvement over previous designs, offering lower crosstalk and SSO noise. The pinout is designed to minimize PDS inductance and keep I/O signal return current paths very closely coupled to their associated I/O signal.

The maximum ratio of I/O to V_{CCO}/GND pin pairs in sparse-chevron packages is 8:1. For this reason, most of the SSO limits (those higher than eight per V_{CCO}/GND pair) are moot for sparse-chevron packages. The SSO limits table, Table 6-40, reflects this. Only I/O standards with limits less than eight (per V_{CCO}/GND pair) appear in the table. All the other I/O standards are designated "no limit" for the nominal PCB case.

For boards that do not meet the nominal PCB requirements listed in "Nominal PCB Specifications", the Virtex-4 FPGA SSO calculator is available, containing all SSO limit data for all I/O standards. For designs in nominal PCBs mixing limited and "no limit" I/O standards, the Virtex-4 FPGA SSO calculator must be used to ensure that I/O utilization does not exceed the limit. Information on the calculator is available under the "Full Device SSO Calculator" section.

Nominal PCB Specifications

The nominal SSO tables (Table 6-40 and Table 6-42) contain SSO limits for cases where the PCB parameters meet the following requirements. In cases where PCB parameters do not meet all requirements listed below, the Virtex-4 FPGA SSO calculator must be used to determine the SSO limit, according to the physical factors of the unique PCB.

PCB Construction

- V_{CCO} and GND vias should have a drill diameter no less than 11 mils (279 µ).
- Total board thickness must be no greater than 62 mils (1575 µ).

Signal Return Current Management

- Traces must be referenced to a plane on an adjacent PCB layer.
- The reference plane must be either GND or the V_{CCO} associated with the output driver.
- The reference layer must remain uninterrupted for its full length from device to device.

Load Traces

- All IOB output buffers must drive controlled impedance traces with characteristic impedance of 50Ω ±10%.
- Total capacitive loading at the far end of the trace (input capacitance of receiving device) must be no more than 10 pF.

Power Distribution System Design

- Designed according to Chapter 4 of the *Virtex-4 PCB Designer's Guide*.
  - At least one decoupling capacitor per V_{CCO} pin (see "Step 1: Determining Critical Parameters of the FPGA")
  - No less than one of each capacitor value present (see "Step 2: Designing the Generic Bypassing Network")
  - Capacitors mounted within a distance of λ/40 (see "Capacitor Placement")
  - Approved solder land patterns (see B, C, and D of Figure 4-6, "Example Capacitor Land and Mounting Geometries")
- V_{CCO} and GND planes cannot be separated by more than 5.0 mils (152 µ)

Nominal SSO Limit Table: Sparse Chevron

Table 6-40 provides the guidelines for the maximum number of simultaneously switching outputs allowed per output power/ground pair to avoid the effects of ground bounce. Table 6-40 omits all I/O standards that meet the no-limit criteria. Only I/O standards with nominal SSO limits of eight or less are listed. SSO limits for all I/O standards are listed in the Virtex-4 FPGA SSO calculator available on the Xilinx website at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=30163

Table 6-40: **Sparse Chevron Simultaneously Switching Output Limits per Equivalent V_{CCO}/GND Pair**(1)

Voltage	IOStandard	Sparse Chevron Limit FF672, FF676, FF1148, FF1152, FF1513, FF1517, FF1760
1.2V	HSTL_I_12	TBD
1.5V	HSTL_IV	5
	HSTL_IV_DCI	5
	LVCMOS15_16_fast	7
1.8V	HSTL_III_18	7
	HSTL_III_DCI_18	7
	HSTL_IV_18	4
	HSTL_IV_DCI_18	4
	LVDCI_DV2_18	6
2.5V	LVCMOS25_24_fast	6
	LVDCI_DV2_25 25 Ω	7
3.3V	LVCMOS33_24_fast	6
	LVTTL_24_slow	8
	GTL	5
	GTL_DCI	5
	GTLP	5
	GTLP_DCI	5
	LVTTL24_fast	6

Notes:

1. There are no SSO limits for LVDS outputs.

Table 6-41 lists the number of equivalent output V_{CCO}/GND pairs for each device, package, and I/O bank.

Equivalent V_{CCO}/GND Pairs: Sparse Chevron

Since ground pins and V_{CCO} pins are connected to common structures inside the package, the number of effective V_{CCO}/GND pin pairs in a bank can differ from the number of physical V_{CCO}/GND pin pairs. Table 6-41 shows the number of equivalent V_{CCO}/GND pin pairs in each bank of each sparse chevron package.

Table 6-41: **Equivalent V_{CCO}/GND Pairs per Bank: Sparse Chevron**

Package	Bank Number																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Virtex®-4 FPGA (LX Family)																	
FF676	4	2	2	2	2	8	8	8	8	8	8	---	---	---	---	---	---
FF1148	4	8	8	2	2	9	9	9	9	9	9	9	9	8	8	---	---
FF1513	3	14	14	2	2	10	9	9	10	9	10	10	9	9	9	9	9
Virtex-4 FPGA (SX Family)																	
FF676	4	2	2	2	2	8	8	8	8	8	8	---	---	---	---	---	---
FF1148	4	8	8	2	2	9	9	9	9	9	9	9	9	8	8	---	---
Virtex-4 FPGA (FX Family)																	
FF672	2	2	2	2	2	8	8	8	8	2	2	---	---	---	---	---	---
FF676	4	2	2	2	2	8	8	8	8	8	8	---	---	---	---	---	---
FF1152	4	2	2	2	2	9	9	9	9	8	8	8	8	---	---	---	---
FF1517	3	8	8	2	2	9	9	9	8	9	9	9	9	9	9	---	---

Notes:

1. These numbers are based on the package files and device pinout. Most of the limitations are based on the availability of GND pins in the vicinity of the bank. There are a few instances where the limitation is due to V_{CCO} pins.
2. Bank 0 in all devices contains no user I/O. Therefore, SSO analysis is unnecessary for Bank 0.

Nominal SSO Limit Tables: Non-Sparse Chevron

Table 6-42 provides the guidelines for the maximum number of simultaneously switching outputs allowed per output power/ground pair to avoid the effects of ground bounce. Refer to Table 6-43 for the number of equivalent output V_{CCO}/GND pairs for each device, package, and I/O bank.

Table 6-42: **Non-Sparse Chevron Simultaneously Switching Output Limits per Equivalent V_{CCO}/GND Pair**

Voltage	IOStandard	Non-Sparse Chevron Limit SF363 and FF668 Packages
1.5V	LVCMOS15_2_slow	51
	LVCMOS15_4_slow	31
	LVCMOS15_6_slow	22
	LVCMOS15_8_slow	17
	LVCMOS15_12_slow	11
	LVCMOS15_16_slow	8
	LVCMOS15_2_fast	30
	LVCMOS15_4_fast	18
	LVCMOS15_6_fast	13
	LVCMOS15_8_fast	10
	LVCMOS15_12_fast	8
	LVCMOS15_16_fast	6
	LVDCI_15 50 Ω	10
	LVDCI_DV2_15 25 Ω	5
	HSLVDCI_15 50 Ω	10
	HSTL_I	20
	HSTL_I_DCI	20
	HSTL_II	10
	HSTL_II_DCI	10
	HSTL_III	8
	HSTL_III_DCI	8
	HSTL_IV	4
	HSTL_IV_DCI	4
	DIFF_HSTL_II	10
	DIFF_HSTL_II_DCI	10

Table 6-42: **Non-Sparse Chevron Simultaneously Switching Output Limits per Equivalent V_{CCO}/GND Pair *(Continued)***

Voltage	IOStandard	Non-Sparse Chevron Limit SF363 and FF668 Packages
1.8V	LVCMOS18_2_slow	58
	LVCMOS18_4_slow	35
	LVCMOS18_6_slow	25
	LVCMOS18_8_slow	19
	LVCMOS18_12_slow	13
	LVCMOS18_16_slow	10
	LVCMOS18_2_fast	34
	LVCMOS18_4_fast	20
	LVCMOS18_6_fast	15
	LVCMOS18_8_fast	11
	LVCMOS18_12_fast	9
	LVCMOS18_16_fast	7
	LVDCI_18 50 Ω	11
	LVDCI_DV2_18 25 Ω	5
	HSLVDCI_18 50 Ω	11
	HSTL_I_18	16
	HSTL_I_DCI_18	16
	HSTL_II_18	8
	HSTL_II_DCI_18	8
	HSTL_III_18	6
	HSTL_III_DCI_18	6
	HSTL_IV_18	3
	HSTL_IV_DCI_18	3
	SSTL18_I	20
	SSTL18_I_DCI	20
	SSTL18_II	13
	SSTL18_II_DCI	13
	DIFF_HSTL_II_18	8
	DIFF_HSTL_II_DCI_18	8
	DIFF_SSTL18_II	12
	DIFF_SSTL18_II_DCI	12

Table 6-42: **Non-Sparse Chevron Simultaneously Switching Output Limits per Equivalent V_{CCO}/GND Pair *(Continued)***

Voltage	IOStandard	Non-Sparse Chevron Limit SF363 and FF668 Packages
2.5V	LVCMOS25_2_slow	68
	LVCMOS25_4_slow	41
	LVCMOS25_6_slow	29
	LVCMOS25_8_slow	22
	LVCMOS25_12_slow	15
	LVCMOS25_16_slow	11
	LVCMOS25_24_slow	7
	LVCMOS25_2_fast	40
	LVCMOS25_4_fast	24
	LVCMOS25_6_fast	17
	LVCMOS25_8_fast	13
	LVCMOS25_12_fast	10
	LVCMOS25_16_fast	8
	LVCMOS25_24_fast	5
	LVDCI_25 50 Ω	13
	LVDCI_DV2_25 25 Ω	6
	HSLVDCI_25 50 Ω	13
	SSTL2_I	15
	SSTL2_I_DCI	15
	SSTL2_II	10
	SSTL2_II_DCI	10
	DIFF_SSTL2_II	10
	DIFF_SSTL2_II_DCI	10
	LVPECL_25	8
	BLVDS_25	8

Table 6-42: **Non-Sparse Chevron Simultaneously Switching Output Limits per Equivalent V_{CCO}/GND Pair *(Continued)***

Voltage	IOStandard	Non-Sparse Chevron Limit SF363 and FF668 Packages
3.3V	LVCMOS33_2_slow	68
	LVCMOS33_4_slow	41
	LVCMOS33_6_slow	29
	LVCMOS33_8_slow	22
	LVCMOS33_12_slow	15
	LVCMOS33_16_slow	11
	LVCMOS33_24_slow	7
	LVCMOS33_2_fast	40
	LVCMOS33_4_fast	24
	LVCMOS33_6_fast	17
	LVCMOS33_8_fast	13
	LVCMOS33_12_fast	10
	LVCMOS33_16_fast	8
	LVCMOS33_24_fast	5
	LVDCI_33 50 Ω	13
	HSLVDCI_33 50 Ω	13
	LVTTL2_slow	68
	LVTTL4_slow	41
	LVTTL6_slow	29
	LVTTL8_slow	22
	LVTTL12_slow	15
	LVTTL16_slow	11
	LVTTL24_slow	7
	LVTTL2_fast	40
	LVTTL4_fast	24
	LVTTL6_fast	17
	LVTTL8_fast	13
	LVTTL12_fast	10
	LVTTL16_fast	8
	LVTTL24_fast	5
	PCI33_3/PCI66_3/PCIX	9

Table 6-42: **Non-Sparse Chevron Simultaneously Switching Output Limits per Equivalent V_{CCO}/GND Pair *(Continued)***

Voltage	IOStandard	Non-Sparse Chevron Limit SF363 and FF668 Packages
3.3V	GTL	4
	GTL_DCI	4
	GTLP	4
	GTLP_DCI	4

Equivalent V_{CCO}/GND Pairs: Non-Sparse Chevron

Since ground pins and V_{CCO} pins are connected to common structures inside the package, the number of effective V_{CCO}/GND pin pairs in a bank can differ from the number of physical V_{CCO}/GND pin pairs. Table 6-43 shows the number of equivalent V_{CCO}/GND pin pairs in each bank of each non-sparse chevron package. Some of the numbers are not integers as these banks share GND pins with other resources.

Table 6-43: **Equivalent V_{CCO}/GND Pairs per Bank: Non-Sparse Chevron**

Package	Bank Number																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Virtex-4 FPGA (LX/FX Families)																	
SF363	1	0.5	0.5	1	1	4.5	4.5	3	3	---	---	---	---	---	---	---	---
FF668	2	2	2	2	2	4.5	5	5	4	3(3)	3(3)	---	---	---	---	---	---
Virtex-4 FPGA (SX Family)																	
FF668	2	2	2	2	2	4.5	5	5	4	3	3	---	---	---	---	---	---

Notes:

1. These numbers are based on the package files and device pinout. Some of the numbers are not integers as these banks share their GND pin with other banks. Most of the limitations are based on the availability of GND pins in the vicinity of the bank. There are a few instances where the limitation is due to V_{CCO} pins.
2. Bank 0 in all devices contains no user I/O. Therefore, SSO analysis is unnecessary for Bank 0.
3. Banks 9 and 10 are not available in the XC4VFX12 device.

Actual SSO Limits versus Nominal SSO Limits

The Virtex-4 FPGA SSO limits are defined in for a set of nominal system conditions in Table 6-40 and Table 6-42. To compute the actual limits for a specific user's system, the automated "Parasitic Factors Derating Method (PFDM)" must be used. The PFDM allows the user to account for differences between actual and nominal PCB power systems, receiver capacitive loading, and maximum allowable ground bounce or V_{CC} bounce. A spreadsheet calculator, "Full Device SSO Calculator", automates this process.

Electrical Basis of SSO Noise

Power supply disturbance can take the form of ground bounce or V_{CC} bounce, and is usually a combination of the two. This bounce is a deviation of the die supply voltage (die GND rail or die V_{CC} rail) with respect to the voltage of the associated PCB supply (PCB GND rail or PCB V_{CC} rail). The deviation of die supplies from PCB supplies comes from

the voltage induced across power system parasitics by supply current transients. One cause of current transients is output driver switching events. Numerous output switching events occurring at the same time lead to bigger current transients, and therefore bigger induced voltages (ground bounce, V_{CC} bounce, or rail collapse). Relevant transient current paths exist in the die, package, and PCB, therefore, parasitics from all three must be considered. The larger the value of these parasitics, the larger the voltage induced by a current transient (power-supply disturbance).

V_{CC} bounce affects stable high outputs. Ground bounce affects stable low outputs. Ground bounce also affects inputs configured as certain I/O standards because they interpret incoming signals by comparing them to a threshold referenced to the die ground (as opposed to I/O standards with input thresholds referenced to a V_{REF} voltage). If the die voltage disturbance exceeds the instantaneous noise margin for the interface, then a non-changing input or output can be interpreted as changing.

Parasitic Factors Derating Method (PFDM)

This section describes a method to evaluate whether a design is within the SSO limits when taking into account the specific electrical characteristics of the user's unique system.

The SSO limits in Table 6-40 and Table 6-42 assume nominal values for the parasitic factors of the system. These factors fall into three groups of electrical characteristics:

- PCB PDS parasitics (nominal 1 nH per via)
- Maximum allowable power system disturbance voltage (nominal 600 mV)
- Capacitive loading (nominal 10 pF per load)

When the electrical characteristics of a design differ from the nominal values, the system SSO limit changes. The degree of difference determines the new effective limit for the design. A figure called “SSO Allowance” is used as a single derating factor, taking into account the combined effect of all three groups of system electrical characteristics.

The SSO allowance is a number ranging from 0 to 100% and is a product of three scaling factors:

The *First Scaling Factor* accounts for the PCB PDS parasitic inductance. It is determined by dividing the nominal PCB PDS inductance by the user's PCB PDS inductance, L_{PDS_USR}. The PCB PDS inductance is determined based on a set of board geometries: board thickness, via diameter, breakout trace width and length, and any other additional structures including sockets.

The *Second Scaling Factor* accounts for the maximum allowable power system disturbance. It is determined by dividing the user's maximum allowable power system disturbance, ($V_{DISTURBANCE\_USER}$) by the nominal maximum power system disturbance. $V_{DISTURBANCE\_USER}$ is usually determined by taking the lesser of input undershoot voltage and input logic low threshold.

The *Third Scaling Factor* accounts for the capacitive loading of outputs driven by the FPGA. It is based on the transient current impact of every additional picofarad of load capacitance above the assumed nominal. For every additional 1 pF of load capacitance over the nominal, approximately 9 mV of additional power system disturbance will occur. The additional power system disturbance is compared to the nominal power system disturbance, and a scale factor is derived from the relationship. C_{LOAD_USER} is the user's average load capacitance.

Example calculations show how each scale factor is computed, as well as the SSO allowance. The system parameters used in this example are:

L_{PDS_USER}= 1.1 nH

$V_{DISTURBANCE\_USER}$= 550 mV

C_{LOAD_USER}= 22 pF

First Scaling Factor (SF1)= L_{PDS_NOM}/L_{PDS_USER}
= 1.0 nH/1.1 nH
= 0.909

Second Scaling Factor (SF2)= $V_{DISTURBANCE\_USER}/V_{DISTURBANCE\_NOM}$

= 550 mV/600 mV

= 0.917

Third Scaling Factor (SF3)

= $V_{DISTURBANCE\_NOM}/((C_{LOAD\_USER} - C_{LOAD\_NOM}) \times 9\ mV/pF) + V_{DISTURBANCE\_NOM}$

= 600 mV/((22 pF – 15 pF) × 9 mV/pF) + 600 mV

= 600 mV/663 mV

= 0.905

SSO Allowance= SF1 × SF2 × SF3 × 100%

= 0.909 × 0.917 × 0.905 × 100%

= 75.4%

Weighted Average Calculation of SSO

This section describes the SSO calculation where the SSO contributions of all I/O in a bank are combined into a single figure.

SSO of an individual bank is calculated by summing the SSO contributions of the individual I/O standards in the bank. The SSO contribution is the percentage of full utilization of any one I/O standard in any one bank. For drivers of each I/O standard, the calculation follows:

Bank SSO limit (I/O group n)
= (I/O Standard SSO limit × Equivalent V_{CCO}/GND pairs in bank)

SSO Contribution (I/ O group n) = (quantity of drivers)/(Bank SSO limit)

For a bank with drivers of multiple I/O standards, the SSO calculation is:

$$Bank\ SSO = \sum_{(1\ to\ n)} SSO\ Contribution(n)$$

A sample SSO calculation follows. The system parameters used are:

Device:XC4VLX60 FF1148

Bank:1

I/O Standards, Quantities:
SSTL2_II, 22

LVCMOS25_16 Fast, 6

LVCMOS25_6 Fast, 19

First, SSO limits for each I/O standard are obtained from Table 6-42:

I/O Group	I/O Standard	SSO Limit (Drivers per V_{CCO}/GND Pair)
1	SSTL2_II	10
2	LVCMOS25_16 Fast	8
3	LVCMOS25_6 Fast	17

From Table 6-41, the number of equivalent V_{CCO}/GND pairs in Bank 1 for the FF1148 package is *eight*.

The Bank SSO limit is calculated for each I/O standard:

Bank SSO Limit =(# drivers per V_{CCO}/GND pair × 8 V_{CCO}/GND pairs

Bank SSO Limit (1)=10 drivers per V_{CCO}/GND pair × 8 V_{CCO}/GND pairs= 80 drivers

Bank SSO Limit (2)= 8 drivers per V_{CCO}/GND pair × 8 V_{CCO}/GND pairs= 64 drivers

Bank SSO Limit (3)=18 drivers per V_{CCO}/GND pair × 8 V_{CCO}/GND pairs= 136 drivers

The SSO contribution of each I/O standard is calculated as:

SSO Contribution = (quantity of drivers)/(Bank SSO limit)

SSO Contribution (1)=22/80= 27.5%

SSO Contribution (2)=6/64= 9.3%

SSO Contribution (3)=19/136= 14.0%

Finally, the bank SSO is calculated:

Bank 1 SSO = SSO contribution (1) + SSO contribution (2) + SSO Contribution (3)

= 27.5% + 9.3% + 14.0% = 50.9%

Calculation of Full Device SSO

Three separate criteria must be satisfied for a full device design to be within the SSO limit. The first criterion ensures the number of simultaneously switching outputs does not exceed the per-bank limit. The second criterion ensures even distribution of output drivers across the package. A final criterion ensures overall power system disturbance in the chip is not excessive. The SSO allowance is used in both of the latter two constraints, taking into account design-specific parameters. The criteria are as follows:

- SSO for any single bank cannot exceed 100%.
- Average SSO of two adjacent banks cannot exceed 105% of SSO allowance.
- Package SSO cannot exceed SSO allowance.

SSO is computed first on a per I/O bank basis. Next, the average SSO of each adjacent bank pair is computed. Finally the average SSO is computed for all banks to determine the effective utilization for the entire package.

Full Device SSO Example

A sample calculation of full-device SSO is shown for a Virtex-4 FPGA XC4VLX60 FF1148 package. The subscript *NOM* denotes a nominal value while the subscript *DES* denotes a value for the design under analysis.

Step 0: Calculate the SSO Allowance

SSO Allowance =
$(L_{NOM}/L_{DES}) \times (V_{NOISE\_DES}/V_{NOISE\_NOM}) \times$
$(V_{NOISE\_NOM}/(((C_{LOAD\_DES} - C_{LOAD\_NOM}) \times V_{COEFF}) + V_{NOISE\_NOM})$

SSO Allowance =
(1.0 nH/1.1 nH) × (550 mV/600 mV) ×
(600 mV/(((22 × pF – 15 pF) × 9 mV/pF) + 600 mV))

SSO Allowance = 75.4%

Step 1: Calculate the SSO for Each Individual Bank for Bank 1

Ensure the SSO for each bank does not exceed 100%.

Bank1 SSO:50.9% < 100%

Bank2 SSO:50.9% < 100%

Bank3 SSO:0%< 100%

Bank4 SSO:60% < 100%

Bank5 SSO: 35% < 100%

Bank6 SSO:40% < 100%

Bank7 SSO:15% < 100%

Bank8 SSO:30% < 100%

Bank9 SSO:12% < 100%

Bank10 SSO:22% < 100%

Bank11 SSO:80% < 100%

Bank12 SSO:0% < 100%

Bank13 SSO:5% < 100%

Bank14 SSO:60% < 100%

OK!

If the SSO of any bank exceeds 100%, apply ground bounce reduction techniques to the bank until the SSO of all individual banks is less than 100%.

Step 2: Calculate the Adjacent Bank SSO Average for All Adjacent Bank Pairs

Ensure the adjacent bank averages do not exceed 105% of the SSO allowance.

First, calculate the adjacent bank SSO average for Banks 7 and 11, and check against 105% of SSO allowance.

Adjacent Bank SSO Average 7 - 11 =
(SSO of bank 7 + SSO of bank 11)/2 = (15% + 80%)/2 = 47.5%

SSO allowance × 105% > adjacent bank SSO average 7 - 11

= 79.2% > 47.5%

OK!

Then calculate adjacent bank SSO average for all adjacent bank pairs. If the average SSO of two adjacent banks exceeds 105% of the SSO allowance, apply ground bounce reduction

techniques to one or both of these two banks until the average SSO of all adjacent bank pairs is less than or equal to 105% of the SSO allowance.

Step 3: Calculate the Package SSO

Ensure the package SSO does not exceed the SSO allowance.

All Bank SSO average
= (Sum of SSO from all banks)/(number of banks available in the package)

= (51 + 51 + 0 + 60 +35 + 40 +15 + 30 + 12 + 22 + 80 + 0 + 5 + 60)/14 = 32.2%

SSO allowance > All Bank SSO average

75.4% > 32.9

OK!

If the package SSO exceeds the SSO allowance, apply ground bounce reduction techniques to one or more of all I/O banks until the all-bank SSO average is less than or equal to the SSO allowance.

Full Device SSO Calculator

A Microsoft Excel-based spreadsheet, the Virtex-4 FPGA SSO calculator, automates all the PFDM and WASSO calculations. The Virtex-4 FPGA SSO calculator uses PCB geometry, (board thickness, via diameter, and breakout trace width and length) to determine power system inductance. It determines the smallest undershoot and logic-Low threshold voltage among all input devices, calculates the average output capacitance, and determines the SSO allowance by taking into account all of the board-level design parameters mentioned in this document. In addition, the Virtex-4 FPGA SSO calculator checks the adjacent bank and package SSO ensuring the full device design does not exceed the SSO allowance. Since bank-number assignment for Virtex-4 devices is different from package to package due to its columnar architecture (versus the peripheral I/O architecture of previous devices), there is a separate tab at the bottom of the SSO calculator display for each Virtex-4 FPGA package. This customizing allows for the arrangement of physically adjacent banks (as they appear clockwise on each unique package, even though they are not labeled in a contiguous manner), and the hard-coding of the number of V_{CCO}/GND pairs per bank.

The Virtex-4 FPGA SSO calculator can be downloaded from the Xilinx website at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=30163

Other SSO Assumptions

LVDCI and HSLVDCI Drivers

All limits for controlled impedance DCI I/O standards assume a 50Ω output impedance. For higher reference resistor (RR) values, less drive strength is needed, and the SSO limit increases linearly. To calculate the SSO limit for a controlled impedance driver with different reference resistors, the following formula is used:

$$User\ SSO = \frac{User\ RR}{50\Omega}\Omega \quad SSO\ Limit\ for\ \Omega$$

Example

The designer uses LVDCI_18 driver with 65Ω reference resistors. The LVDCI_18 SSO limit for 50Ω impedance is first taken from Table 6-42. The SSO limit for LVDCI_18 at 50Ω is 11 SSO per V_{CCO}/GND pin pair. Therefore, the SSO limit for LVDCI_18 at 65Ω is:

SSO Limit LVDCI_18 at 65Ω = ((65Ω)/50Ω) × 11 = 14.3

Bank 0

Bank 0 in all devices contains only configuration and dedicated signals. Since there is no user I/O in Bank 0, no SSO analysis is necessary for this bank.

Chapter 7

SelectIO Logic Resources

Introduction

This chapter describes the logic directly behind the I/O drivers and receivers covered in Chapter 6, "SelectIO Resources".

Virtex-4 FPGAs contain all of the basic I/O logic resources from Virtex®-II/Virtex-II Pro FPGAs. These resources include the following:

- Combinatorial input/output
- 3-state output control
- Registered input/output
- Registered 3-state output control
- Double-Data-Rate (DDR) input/output
- DDR output 3-state control

In addition, the following architectural improvements have been implemented:

- IDELAY provides users control of an adjustable, fine-resolution input delay element.
- SAME_EDGE output DDR mode
- SAME_EDGE and SAME_EDGE_PIPELINED input DDR mode

ILOGIC Resources

ILOGIC blocks include four storage elements and a programmable absolute delay element, shown in Figure 7-1.

To build an edge-triggered D-type flip-flop, the topmost register (IFF1) is used. Only this register can optionally be configured as a level sensitive latch. The other three registers (IFF2, IFF3, and IFF4) are used to build various input DDR registers. See "Input DDR Overview (IDDR)," page 323 for further discussion on input DDR.

All ILOGIC block registers have a common clock enable signal (CE1) that is active High by default. If left unconnected, the clock enable pin for any storage element defaults to the active state.

All ILOGIC block registers have a common synchronous or asynchronous set and reset (SR and REV signals). The set/reset input pin, SR forces the storage element into the state specified by the SRVAL attributes. When using SR, a second input, REV forces the storage element into the opposite state. The reset condition predominates over the set condition. Table 7-1 and Table 7-2 describe the operation of SR in conjunction with REV.

Table 7-1: **Truth Table when SRVAL = 0 (Default Condition)**

SR	REV	Function
0	0	NOP
0	1	Set
1	0	Reset
1	1	Reset

Table 7-2: **Truth Table when SRVAL = 1**

SR	REV	Function
0	0	NOP
0	1	Reset
1	0	Set
1	1	Reset

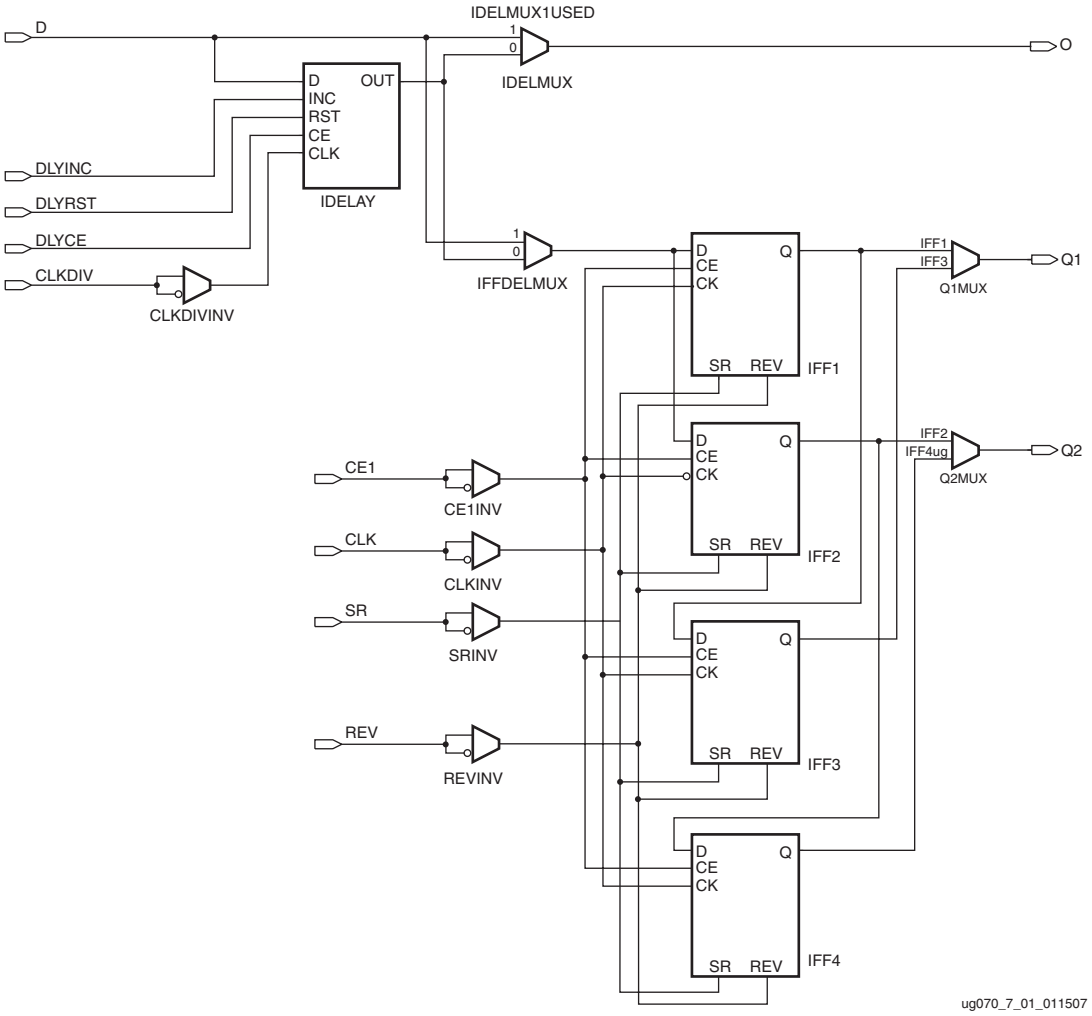


Figure 7-1: **ILOGIC Block Diagram**

The SRVAL attributes can be set individually for each storage element in the ILOGIC block, but the choice of synchronous or asynchronous set/reset (SRTYPE) can not be set individually for each storage element in the ILOGIC block.

The SR and REV pins are shared between adjacent ILOGIC/ISERDES and OLOGIC/OSERDES.

Most of the control signals have an optional inverter. Any inverter placed on a control signal is automatically absorbed into the ILOGIC block (i.e., no CLBs are used).

The following sections discuss the various resources within the ILOGIC blocks. All connections between the ILOGIC resources are managed in Xilinx® software.

Combinatorial Input Path

The combinatorial input path is used to create a direct connection from the input driver to the FPGA logic. This path is used by software automatically when:

1. There is a direct (unregistered) connection from input data to logic resources in the FPGA logic.
2. The "pack I/O register/latches into IOBs" is set to OFF.

Input DDR Overview (IDDR)

Virtex-4 devices have dedicated registers in the ILOGIC to implement input double-data-rate (DDR) registers. This feature is used by instantiating the IDDR primitive.

There is only one clock input to the IDDR primitive. Falling edge data is clocked by a locally inverted version of the input clock. All clocks feeding into the I/O tile are fully multiplexed, i.e., there is no clock sharing between ILOGIC or OLOGIC blocks. The IDDR primitive supports the following modes of operation:

- OPPOSITE_EDGE mode
- SAME_EDGE mode
- SAME_EDGE_PIPELINED mode

The SAME_EDGE and SAME_EDGE_PIPELINED modes are new for the Virtex-4 architecture. These new modes allow designers to transfer falling edge data to the rising edge domain within the ILOGIC block, saving CLB and clock resources, and increasing performance. These modes are implemented using the DDR_CLK_EDGE attribute. The following sections describe each of the modes in detail.

OPPOSITE_EDGE Mode

A traditional input DDR solution, or OPPOSITE_EDGE mode, is accomplished via a single input signal driving two registers (IFF1 and IFF2) in the ILOGIC. Both registers are rising edge triggered. The second register (IFF2) receives an inverted version of the clock. The result is that rising edge data is presented to the fabric via the first register output (Q1) and falling edge data via the second register output (Q2). This structure is similar to the Virtex-II and Virtex-II Pro FPGA implementation. Figure 7-2 shows a simplified input

DDR register and the signals/ports associated with OPPOSITE_EDGE mode. Figure 7-3 shows the timing diagram of the input DDR using the OPPOSITE_EDGE mode.

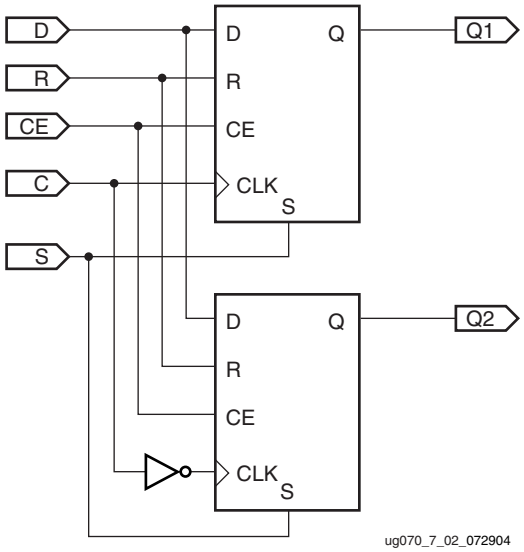


Figure 7-2: **Input DDR in OPPOSITE_EDGE Mode**

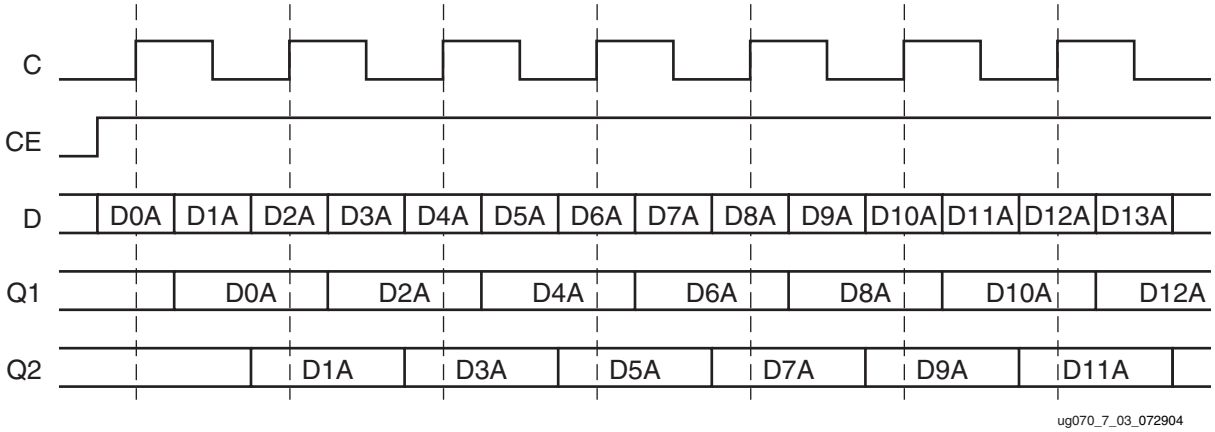


Figure 7-3: **Input DDR Timing in OPPOSITE_EDGE Mode**

SAME_EDGE Mode

In the SAME_EDGE mode a third register (IFF4), clocked by the rising edge clock, is placed on the output of the falling edge register. Figure 7-4 shows input DDR registers and the signals associated with the SAME_EDGE mode.

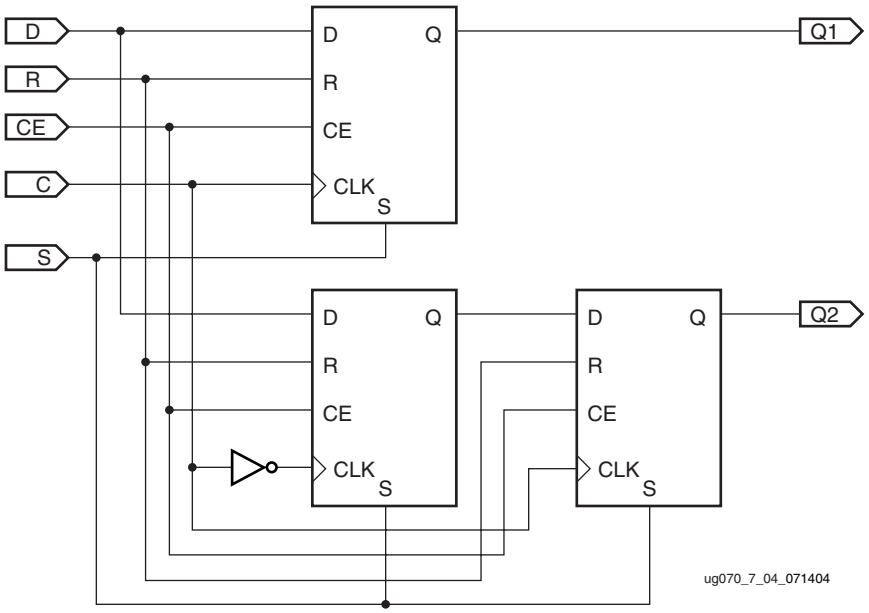


Figure 7-4: **Input DDR in SAME_EDGE Mode**

By adding the third register, data is presented into the FPGA fabric on the same clock edge. However, the additional register causes the data pair to be separated by one clock cycle. Figure 7-5 shows the timing diagram of the input DDR using the SAME_EDGE mode. In the timing diagram, the output pairs are no longer (0) and (1). Instead, the first pair presented is pair (0) and (don't care), followed by pair (1) and (2) on the next clock cycle.

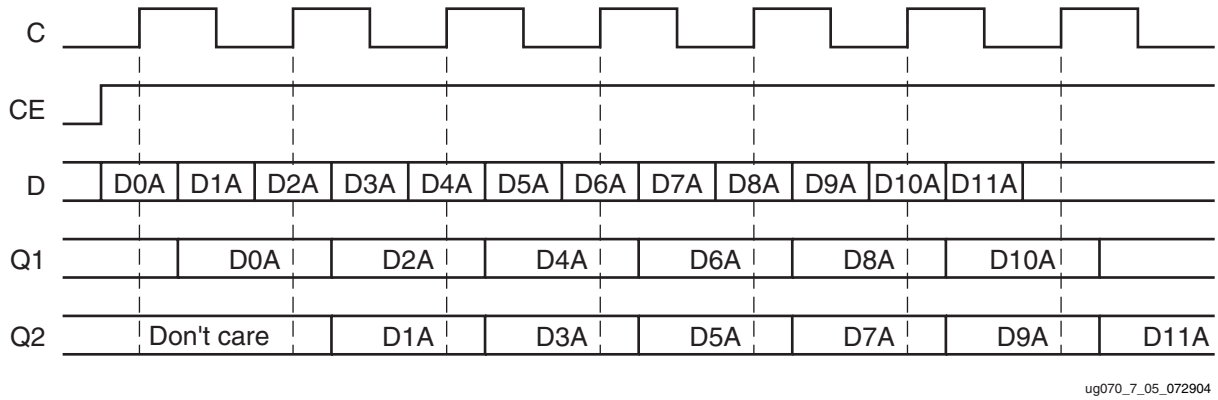


Figure 7-5: **Input DDR Timing in SAME_EDGE Mode**

SAME_EDGE_PIPELINED Mode

In the SAME_EDGE_PIPELINED mode a fourth register (IFF3), clocked by the rising-edge clock, is placed on the output of the two registers. Figure 7-6 shows the input DDR registers and the signals involved using the SAME_EDGE_PIPELINED mode.

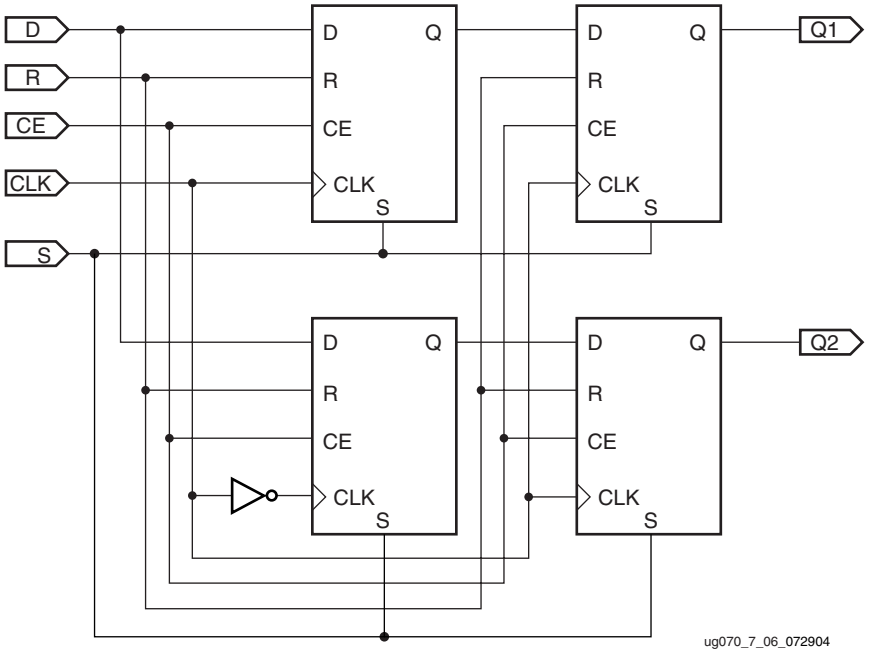


Figure 7-6: **Input DDR in SAME_EDGE_PIPELINED Mode**

By adding the additional register, data is presented into the FPGA fabric on the same clock edge. Unlike the SAME_EDGE mode, the additional registers do not cause the data pair to be separated. However, an additional clock latency is required to remove the separated effect of the SAME_EDGE mode. Figure 7-7 shows the timing diagram of the input DDR using the SAME_EDGE_PIPELINED mode. The output pairs, Q1 and Q2 are presented to the FPGA fabric at the same time.

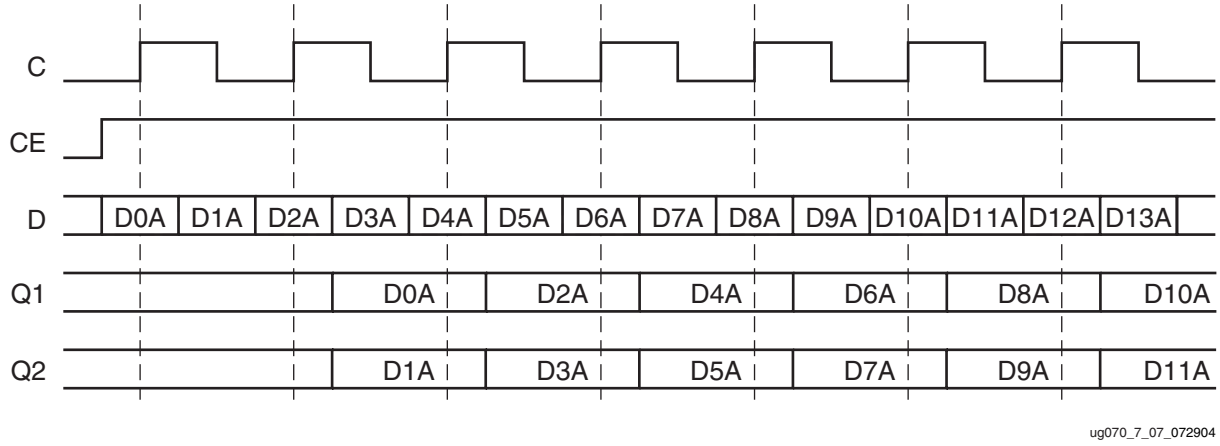


Figure 7-7: **Input DDR Timing in SAME_EDGE_PIPELINED Mode**

Input DDR Primitive (IDDR)

Figure 7-8 shows the block diagram of the IDDR primitive. Table 7-3 lists the IDDR port signals. Table 7-4 describes the various attributes available and default values for the IDDR primitive.

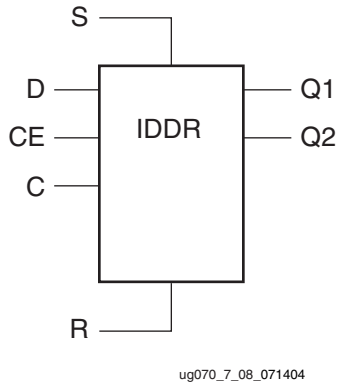


Figure 7-8: **IDDR Primitive Block Diagram**

Table 7-3: **IDDR Port Signals**

Port Name	Function	Description
Q1 and Q2	Data outputs	IDDR register outputs. Q1 is rising edge data, Q2 is falling edge data.
C	Clock input port	The C pin represents the clock input pin.
CE	Clock enable port	The enable pin affects the loading of data into the DDR flip-flop. When Low, clock transitions are ignored and new data is not loaded into the DDR flip-flop. CE must be High to load new data into the DDR flip-flop.
D	Data input (DDR)	IDDR register input from IOB.
R	Reset	Synchronous/Asynchronous reset pin. Reset is asserted High.
S	Set	Synchronous/Asynchronous set pin. Set is asserted High.

Table 7-4: **IDDR Attributes**

Attribute Name	Description	Possible Values
DDR_CLK_EDGE	Sets the IDDR mode of operation with respect to clock edge	OPPOSITE_EDGE (default), SAME_EDGE, SAME_EDGE_PIPELINED
INIT_Q1	Sets the initial value for Q1 port	0 (default), 1
INIT_Q2	Sets the initial value for Q2 port	0 (default), 1
SRTYPE	Set/Reset type with respect to clock (C)	ASYNC, SYNC (default)

IDDR VHDL and Verilog Templates

The following examples illustrate the instantiation of the IDDR primitive in VHDL and Verilog.

IDDR VHDL Template

```
--Example IDDR component declaration

component IDDR
  generic(
      DDR_CLK_EDGE : string := "OPPOSITE_EDGE";
      INIT_Q1      : bit    := '0';
      INIT_Q2      : bit    := '0';
      SRTYPE       : string := "SYNC";
    );

  port(
      Q1          : out std_ulogic;
      Q2          : out std_ulogic;

      C           : in  std_ulogic;
      CE          : in  std_ulogic;
      D           : in  std_ulogic;
      R           : in  std_ulogic;
      S           : in  std_ulogic
    );
  end component;

--Example IDDR instantiation

U_IDDR : IDDR
Port map(
      Q1 => user_q1,
      Q2 => user_q2,
      C => user_c,
      CE => user_ce,
      D => user_d,
      R => user_r,
      S => user_s
    );
```

IDDR Verilog Template

```
//Example IDDR module declaration

module IDDR (Q1, Q2, C, CE, D, R, S);

    output Q1;
    output Q2;

    input C;
    input CE;
    input D;
    tri0 GSR = glbl.GSR;
    input R;
    input S;
```

```
    parameter DDR_CLK_EDGE = "OPPOSITE_EDGE";
    parameter INIT_Q1 = 1'b0;
    parameter INIT_Q2 = 1'b0;
    parameter SRTYPE = "SYNC";

endmodule;

//Example IDDR instantiation

IDDR U_IDDR (
.Q1(user_q1),
.Q2(user_q2),
.C(user_c),
.CE(user_ce),
.D(user_d),
.R(user_r),
.S(user_s)
);
```

ILOGIC Timing Models

This section describes the timing associated with the various resources within the ILOGIC block.

ILOGIC Timing Characteristics

Figure 7-9 illustrates ILOGIC register timing. When IDELAY is used, T_{IDOCK} is replaced by T_{IDOCKD}.

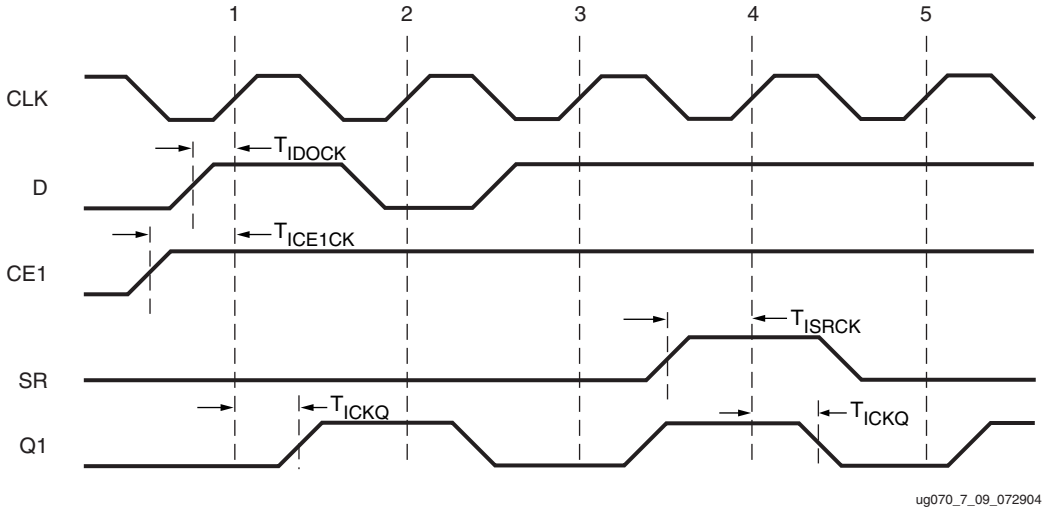


Figure 7-9: **ILOGIC Input Register Timing Characteristics**

Clock Event 1

- At time T_{ICE1CK} before Clock Event 1, the input clock enable signal becomes valid-High at the CE1 input of the input register, enabling the input register for incoming data.
- At time T_{IDOCK} before Clock Event 1, the input signal becomes valid-High at the D input of the input register and is reflected on the Q1 output of the input register at time T_{ICKQ} after Clock Event 1.

Clock Event 4

- At time T_{ISRCK} before Clock Event 4, the SR signal (configured as synchronous reset in this case) becomes valid-High resetting the input register and reflected at the Q1 output of the IOB at time T_{ICKQ} after Clock Event 4.

ILOGIC Timing Characteristics, DDR

Figure 7-10 illustrates the ILOGIC in IDDR mode timing characteristics. When IDELAY is used, T_{IDOCK} is replaced by T_{IDOCKD}. The example shown uses IDDR in OPPOSITE_EDGE mode. For other modes, add the appropriate latencies as shown in Figure 7-7, page 326.

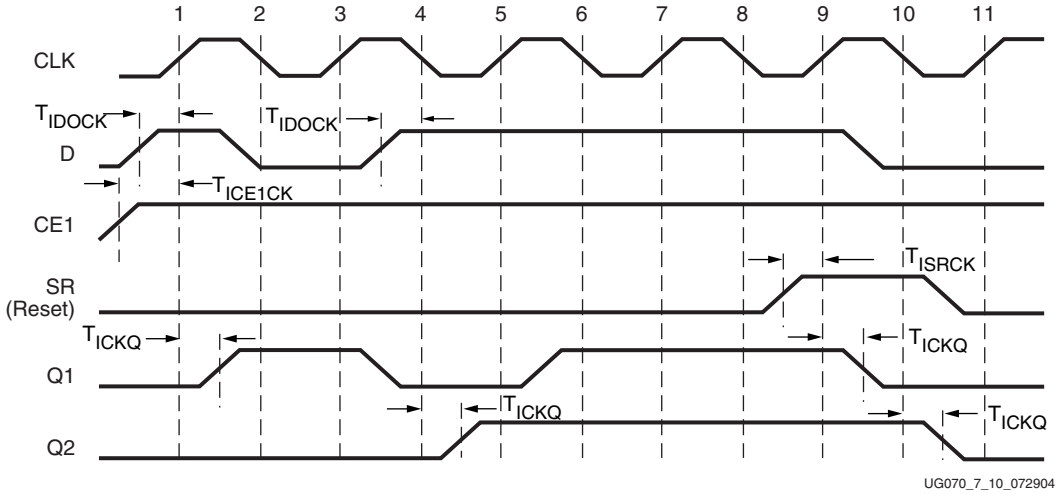


Figure 7-10: **ILOGIC in IDDR Mode Timing Characteristics (OPPOSITE_EDGE Mode)**

Clock Event 1

- At time T_{ICE1CK} before Clock Event 1, the input clock enable signal becomes valid-High at the CE1 input of both of the DDR input registers, enabling them for incoming data. Since the CE1 and D signals are common to both DDR registers, care must be taken to toggle these signals between the rising edges and falling edges of CLK as well as meeting the register setup-time relative to both clocks.
- At time T_{IDOCK} before Clock Event 1 (rising edge of CLK), the input signal becomes valid-High at the D input of both registers and is reflected on the Q1 output of input-register 1 at time T_{ICKQ} after Clock Event 1.

Clock Event 2

- At time T_{IDOCK} before Clock Event 2 (falling edge of CLK), the input signal becomes valid-Low at the D input of both registers and is reflected on the Q2 output of input-register 2 at time T_{ICKQ} after Clock Event 2 (no change in this case).

Clock Event 9

- At time T_{ISRCK} before Clock Event 9, the SR signal (configured as synchronous reset in this case) becomes valid-High resetting IFF1 (Q1) at time T_{ICKQ} after Clock Event 9, and IFF2 (Q2) at time T_{ICKQ} after Clock Event 10.

Table 7-5 describes the function and control signals of the ILOGIC switching characteristics in the *Virtex-4 Data Sheet*.

Table 7-5: **ILOGIC Switching Characteristics**

Symbol	Description
Setup/Hold	
T_{ICE1CK}/T_{ICKCE1}	CE1 pin setup/hold with respect to CLK
T_{ICECK}/T_{ICKCE}	DLYCE pin setup/hold with respect to CLKDIV
T_{IRSTCK}/T_{ICKRST}	DLYRST pin setup/hold with respect to CLKDIV
T_{IINCCK}/T_{ICKINC}	DLYINC pin setup/hold with respect to CLKDIV
T_{ISRCK}/T_{ICKSR}	SR/REV pin setup/hold with respect to CLK
T_{IDOCK}/T_{IOCKD}	D pin setup/hold with respect to CLK without Delay
T_{IDOCKD}/T_{IOCKDD}	D pin setup/hold with respect to CLK (IOBDELAY_TYPE = DEFAULT)
	D pin setup/hold with respect to CLK (IOBDELAY_TYPE = FIXED, IOBDELAY_VALUE = 0)
Combinatorial	
T_{IDI}	D pin to O pin propagation delay, no Delay
T_{IDID}	D pin to O pin propagation delay (IOBDELAY_TYPE = DEFAULT)
	D pin to O pin propagation delay (IOBDELAY_TYPE = FIXED, IOBDELAY_VALUE = 0)
Sequential Delays	
T_{IDLO}	D pin to Q1 pin using flip-flop as a latch without Delay
T_{IDLOD}	D pin to Q1 pin using flip-flop as a latch (IOBDELAY_TYPE = DEFAULT)
	D pin to Q1 pin using flip-flop as a latch (IOBDELAY_TYPE = FIXED, IOBDELAY_VALUE = 0)
T_{ICKQ}	CLK to Q outputs
T_{ICE1Q}	CE1 pin to Q1 using flip-flop as a latch, propagation delay
T_{RQ}	SR/REV pin to OQ/TQ out

Input Delay Element (IDELAY)

Every ILOGIC block contains a programmable absolute delay element called IDELAY. (Refer to Figure 7-1, "ILOGIC Block Diagram.") IDELAY is a 64-tap wrap-around delay element with a fixed, guaranteed tap resolution (see *Virtex-4 Data Sheet*). It can be applied to the combinatorial input path, registered input path, or both. IDELAY allows incoming signals to be delayed on an individual basis. The delay element is calibrated to provide an absolute delay value ($T_{IDELAYRESOLUTION}$) independent of process, voltage, and temperature variation. Three modes of operation are available:

- Zero-hold time delay mode (IOBDELAY_TYPE = DEFAULT)

  This mode of operation allows backward compatibility for designs using the zero-hold time delay feature in Virtex-II and Virtex-II Pro devices. When used in this mode, the IDELAYCTRL primitive does not need to be instantiated (see IDELAYCTRL Usage and Design Guidelines for more details).

- Fixed delay mode (IOBDELAY_TYPE = FIXED)

  In the fixed delay mode, the delay value is preset at configuration to the tap number determined by the attribute IOBDELAY_VALUE. Once configured, this value cannot be changed. When used in this mode, the IDELAYCTRL primitive must be instantiated (see IDELAYCTRL Usage and Design Guidelines for more details).

- Variable delay mode (IOBDELAY_TYPE = VARIABLE)

  In the variable delay mode, the delay value can be changed after configuration by manipulating the control signals CE and INC. When used in this mode, the IDELAYCTRL primitive must be instantiated (see IDELAYCTRL Usage and Design Guidelines for more details).

IDELAY Primitive

Figure 7-11 shows the IDELAY primitive.

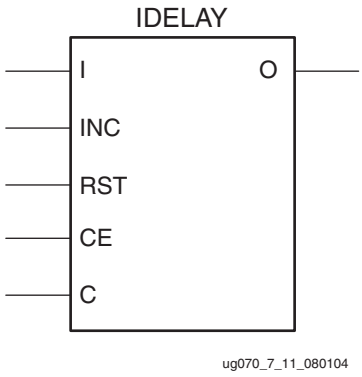


Figure 7-11: **IDELAY Primitive**

Table 7-6 lists the available ports in the IDELAY primitive.

Table 7-6: **IDELAY Primitive**

Port Name	Direction	Function
I	Input	Serial input data from IOB
C	Input	Clock input when in Variable mode
INC	Input	Increment/decrement number of tap delays when in Variable mode
CE	Input	Enable increment/decrement function when in Variable mode
RST	Input	Reset delay element to pre-programmed value. If no value programmed, reset to 0.
O	Output	Combinatorial output

IDELAY Ports

Data Input and Output - I and O

The data input (I) is driven by its associated IOB (i.e., input from the pin). The IDELAY data output (O) can drive directly to the fabric, to the registers in the ILOGIC block, or to both.

Clock Input - C

All control inputs to IDELAY (RST, CE and INC) are synchronous to the clock input (C). A clock must be connected to this port when IDELAY is configured in variable mode.

Module Reset - RST

The IDELAY reset signal, RST, resets the delay element to a value set by the IOBDELAY_VALUE attribute. If the IOBDELAY_VALUE attribute is not specified, a value of 0 is assumed. The RST signal is an active-High reset and is synchronous to the input clock signal (C).

Increment/Decrement Signals - CE, INC

The increment/decrement enable signal (CE) controls when an increment/decrement function is to be performed. As long as CE remains High, IDELAY will increment or decrement by $T_{IDELAYRESOLUTION}$ every clock (C) cycle. The state of INC determines whether IDELAY will increment or decrement; INC = 1 increments, INC = 0 decrements, synchronously to the clock (C). If CE is Low the delay through IDELAY will not change regardless of the state of INC.

IDELAY is a wrap-around programmable delay element. When the end of the delay element is reached (tap 63) a subsequent increment function will return to tap 0. The same applies to the decrement function: decrementing below zero moves to tap 63. The increment/decrement operation is summarized in Table 7-7.

Table 7-7: **Increment/Decrement Operations**

Operation	RST	CE	INC
Reset to IOBDELAY_VALUE	1	x	x
Increment tap count	0	1	1
Decrement tap count	0	1	0
No change	0	0	x

Notes:

1. RST takes precedence over CE and INC.

IDELAY Attributes

Table 7-8 summarizes the IDELAY attributes.

Table 7-8: **IDELAY Attribute Summary**

IDELAY Attribute	Description	Value	Default Value
IOBDELAY_TYPE	Sets the type of tap delay.	String: DEFAULT, FIXED, or VARIABLE	DEFAULT
IOBDELAY_VALUE	Specifies the initial tap setting.	Integer: 0 to 63	0

IOBDELAY_TYPE Attribute

The IOBDELAY_TYPE attribute sets the type of delay used. The attribute values are DEFAULT, FIXED, and VARIABLE. When set to DEFAULT, the zero-hold time delay element is selected. This delay element is used to guarantee non-positive hold times when global clocks are used without DCMs to capture data (pin-to-pin parameters).

When set to FIXED, the tap-delay value is fixed at the number of taps determined by the IOBDELAY_VALUE attribute setting. This value is preset and cannot be changed after configuration.

When set to VARIABLE, the variable tap delay element is selected. The tap delay can be incremented by setting CE = 1 and INC = 1, or decremented by CE = 1 and INC = 0. The increment/decrement operation is synchronous to C, the input clock signal.

IOBDELAY_VALUE Attribute

The IOBDELAY_VALUE attribute specifies the initial number of tap delays. The possible values are any integer from 0 to 63. The default value is zero. The value of the tap delay reverts to IOBDELAY_VALUE when the tap delay is reset.

IDELAY Timing

Table 7-9 shows the IDELAY switching characteristics.

Table 7-9: **Input Delay Switching Characteristics**

Symbol	Description
$T_{IDELAYRESOLUTION}$	IDELAY tap resolution
T_{ICECK}/T_{ICKCE}	CE pin setup/hold with respect to C
T_{IINCCK}/T_{ICKINC}	INC pin setup/hold with respect to C
T_{IRSTCK}/T_{ICKRST}	RST pin setup/hold with respect to C

Figure 7-12 shows an IDELAY timing diagram. It is assumed that IOBDELAY_VALUE = 0.

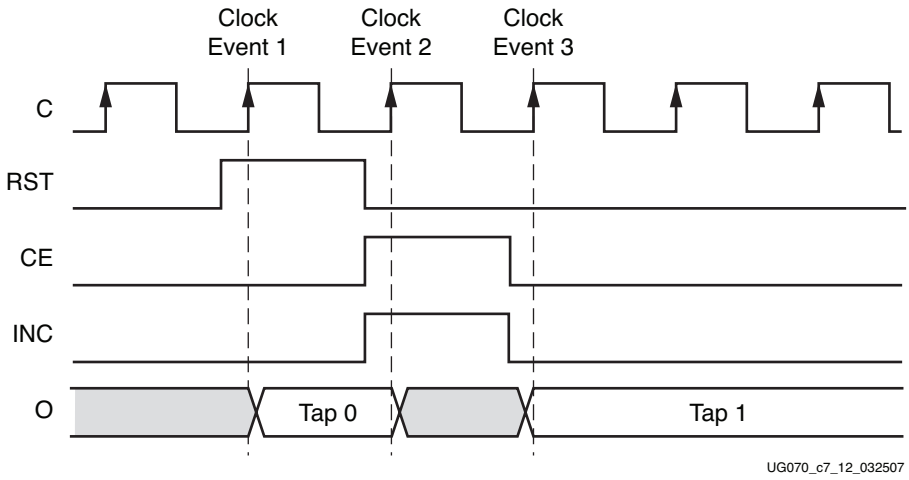


Figure 7-12: **IDELAY Timing Diagram**

Clock Event 1

On the rising edge of C, a reset is detected, causing the output O to select tap 0 as the output from the 64-tap chain (assuming IOBDELAY_VALUE = 0).

Clock Event 2

A pulse on CE and INC is captured on the rising edge of C. This indicates an Increment operation. For the remainder of the cycle associated with clock event 2, the output is changing from tap 0 to tap 1. During this time, the output may be unstable and contain erroneous data.

Clock Event 3

By this time, the output has stabilized at tap 1, thus completing the Increment operation. The output remains at tap 1 indefinitely until there is further activity on the RST, CE, or INC pins.

Note on Instability after an Increment/Decrement Operation

In Figure 7-12 there is a period of instability when the output is changing from one tap to another. If the IDELAY output is sampled during that period of instability, the captured data may be incorrect. In the case where the IDELAY output is sampled by the same clock that is connected to the C input, there is no danger of capturing a bit during the unstable period because the settling time is less than one period of clock C.

However, IDELAY is often used in conjunction with an ISERDES or an IDDR element. These elements sample the output of IDELAY at a much higher rate than the clock connected to the C input of IDELAY. The result is that data is sampled during the unstable period (resulting in at most one bit error). This potentially corrupt sample must propagate through the ISERDES or IDDR before becoming visible to the user. When the potential bit error emerges, the user should treat it as corrupted. The user must examine the latency of the ISERDES or IDDR being used (it differs depending on modes) to determine when the data is valid again. If a small increase in latency is not a concern, the user can wait 4 CLKDIV cycles when using an ISERDES or 4 CLK cycles when using an IDDR. This is a

safe, conservative waiting period for all modes of those modules. A closer examination of latency for a specific path in a specific mode can trim these waiting periods down further.

If IDELAY is used in the path of a clock channel, and the delay is dynamically adjusted during operation, every circuit that runs on that clock should be held in reset while the delay is changed. This is because the clock may experience a short glitch when the tap setting is incremented or decremented, and this could result in erroneous behavior in state machines and other circuits.

If an IDELAY is used in the path of a data signal that is passing constant user traffic in which even a single bit error is unacceptable, and the user desires to change the tap setting in real-time, a redundant path through a second IDELAY and ISERDES (or IDDR) must be added to the design to allow the user to switch to the redundant path while changing the tap setting of the primary path.

IDELAY VHDL and Verilog Instantiation Template

VHDL and Verilog instantiation templates are available in the Libraries Guide for all primitives and submodules.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signals names.

VHDL for Zero-Hold Time Delay Mode

The following VHDL code shows how to instantiate the IDELAY module in a zero-hold time delay mode.

```
-- Module: IDELAY
-- Description: VHDL instantiation template
-- Zero Hold Time Mode
--
-- Device: Virtex-4 Family
-------------------------------------------------------------------------
-- Components Declarations
-- Component Declaration for IDELAY should be placed
-- after architecture statement but before "begin" keyword

component IDELAY

generic (IOBDELAY_TYPE : string := "DEFAULT"; --(DEFAULT, FIXED,
            VARIABLE)
            IOBDELAY_VALUE : integer := 0 --(0 to 63)
          );
port (
      O : out STD_LOGIC;
      I : in STD_LOGIC;
      C : in STD_LOGIC;
      CE : in STD_LOGIC;
      INC : in STD_LOGIC;
      RST : in STD_LOGIC
     );
end component;

-- Component Attribute specification for IDELAY
-- should be placed after architecture declaration but
-- before the "begin" keyword
--
```

```
-- Architecture Section
--
attribute IOBDELAY_TYPE : string;
attribute IOBDELAY_VALUE: integer;

-- Component Instantiation for IDELAY should be placed
-- in architecture after the "begin" keyword
--
-- Instantiation Section
--
U1 : IDELAY

generic map (
             IOBDELAY_TYPE => "DEFAULT", -- Set to DEFAULT for
                                         -- Zero Hold Time Mode
             IOBDELAY_VALUE => 0  -- (0 to 63)
            );

port map (
          O => data_output,
          I => data_input,
          C => '0',
          CE => '0',
          INC => '0',
          RST => '0'
         );
```

Verilog for Zero-Hold Time Delay Mode

```
// Module: IDELAY
// Description: Verilog instantiation template
// Zero Hold Time Mode
//
// Device: Virtex-4 Family
//-----------------------------------------------------------------------
// Instantiation Section
//
IDELAY U1
   (
      O(data_output)
      I(data_input),
      C(1'b0),
      CE(1'b0),
      INC(1'b0),
      RST(1'b0)
   );

//Set IOBDELAY_TYPE attribute to DEFAULT for Zero Hold Time Mode
//synthesis attribute IOBDELAY_TYPE of U1 is "DEFAULT";
//synthesis attribute IOBDELAY_VALUE of U1 is 0;
```

Fixed Delay Mode

The following code shows how to instantiate the IDELAY module in fixed delay mode with a tap setting of 31. IDELAYCTRL must also be instantiated when operating in this mode (see "IDELAYCTRL Overview," page 341).

VHDL for Fixed Delay Mode

```
-- The IDELAYCTRL primitive must be instantiated in conjunction with the
-- IDELAY primitive when used in Fixed Delay Mode.

-- Module: IDELAY
-- Description: VHDL instantiation template
-- Fixed Delay Mode
--
-- Device: Virtex-4 Family
------------------------------------------------------------------------
-- Components Declarations
-- Component Declaration for IDELAY should be placed
-- after architecture statement but before "begin" keyword

component IDELAY

generic (
            IOBDELAY_TYPE : string := "DEFAULT"; --(DEFAULT, FIXED,
VARIABLE)
            IOBDELAY_VALUE : integer := 0 --(0 to 63)
           );

port (
      O : out STD_LOGIC;
      I : in STD_LOGIC;
      C : in STD_LOGIC;
      CE : in STD_LOGIC;
      INC : in STD_LOGIC;
      RST : in STD_LOGIC
     );
end component;

-- Component Attribute specification for IDELAY
-- should be placed after architecture declaration but
-- before the "begin" keyword
--
-- Architecture Section
--
attribute IOBDELAY_TYPE : string;
attribute IOBDELAY_VALUE: integer;

-- Component Instantiation for IDELAY should be placed
-- in architecture after the "begin" keyword
--
-- Instantiation Section
--
U1 : IDELAY

generic map (
             IOBDELAY_TYPE => "FIXED", -- Set to FIXED for
                                       -- Fixed delay mode
             IOBDELAY_VALUE => 31  -- Set the delay value equal
                               -- to the center of the delay element
            );

port map (
          O => data_output,
          I => data_input,
```

```
            C => '0',
            CE => '0',
            INC => '0',
            RST => '0'
           );
```

Verilog Code for Fixed Delay Mode

```
// The IDELAYCTRL primitive must be instantiated in conjunction with
// IDELAY
// primitive when used in Fixed Delay Mode.
// Module: IDELAY
// Description: Verilog instantiation template
// Fixed Delay Mode
//
// Device: Virtex-4 Family
//-----------------------------------------------------------------
// Instantiation Section
//
IDELAY U1
   (
      O(data_output)
      I(data_input),
      C(1'b0),
      CE(1'b0),
      INC(1'b0),
      RST(1'b0)
   );

//Set IOBDELAY_TYPE attribute to FIXED for Fixed Delay Mode
//synthesis attribute IOBDELAY_TYPE of U1 is "FIXED";
//Set IOBDELAY_VALUE attribute to 31 for center of delay element
//synthesis attribute IOBDELAY_VALUE of U1 is 31;
```

Variable Delay Mode

The following code shows how to instantiate the IDELAY module in variable delay mode. IDELAYCTRL must also be instantiated when operating in this mode (see "IDELAYCTRL Overview," page 341).

VHDL Code for Variable Delay Mode

```
-- The IDELAYCTRL primitive must be instantiated in conjunction with the
IDELAY
-- primitive when used in Variable Delay Mode.

-- Module: IDELAY
-- Description: VHDL instantiation template
-- Variable Delay Mode
--
-- Device: Virtex-4 Family
-----------------------------------------------------------------
-- Components Declarations
-- Component Declaration for IDELAY should be placed
-- after architecture statement but before "begin" keyword

component IDELAY

generic (
```

```
            IOBDELAY_TYPE : string := "DEFAULT"; --(DEFAULT, FIXED,
VARIABLE)
            IOBDELAY_VALUE : integer := 0 --(0 to 63)
           );

port (
      O : out STD_LOGIC;
      I : in STD_LOGIC;
      C : in STD_LOGIC;
      CE : in STD_LOGIC;
      INC : in STD_LOGIC;
      RST : in STD_LOGIC
     );
end component;

-- Component Attribute specification for IDELAY
-- should be placed after architecture declaration but
-- before the "begin" keyword
--
-- Architecture Section
--
attribute IOBDELAY_TYPE : string;
attribute IOBDELAY_VALUE: integer;

-- Component Instantiation for IDELAY should be placed
-- in architecture after the "begin" keyword
--
-- Instantiation Section
--
U1 : IDELAY

generic map (
             IOBDELAY_TYPE => "VARIABLE", -- Set to VARIABLE for
                                          -- Variable Delay Mode
             IOBDELAY_VALUE => 0
            );

port map (
          O => data_output,
          I => data_input,
          C => clkdiv,
          CE => dlyce,
          INC => dlyinc,
          RST => dlyrst
         );
```

Verilog Code for Variable Delay Mode

```
// The IDELAYCTRL primitive must be instantiated in conjunction with the
// IDELAY primitive when used in Variable Delay Mode.

// Module: IDELAY
// Description: Verilog instantiation template
// Variable Delay Mode
//
// Device: Virtex-4 Family
//-----------------------------------------------------------------------
// Instantiation Section
//
```

```
IDELAY U1
   (
      O(data_output)
      I(data_input),
      C(clkdiv),
      CE(dlyce),
      INC(dlyinc),
      RST(dlyrst)
   );

//Set IOBDELAY_TYPE attribute to VARIABLE for Variable Delay Mode
//synthesis attribute IOBDELAY_TYPE of U1 is "VARIABLE";
//synthesis attribute IOBDELAY_VALUE of U1 is 0;
```

IDELAYCTRL Overview

If the IDELAY or ISERDES primitive is instantiated with the IOBDELAY_TYPE attribute set to FIXED or VARIABLE, the IDELAYCTRL module must be instantiated. The IDELAYCTRL module continuously calibrates the individual delay elements (IDELAY) in its region (see Figure 7-15, page 343), to reduce the effects of process, voltage, and temperature variations. The IDELAYCTRL module calibrates IDELAY using the user supplied REFCLK.

IDELAYCTRL Primitive

Figure 7-13 shows the IDELAYCTRL primitive.

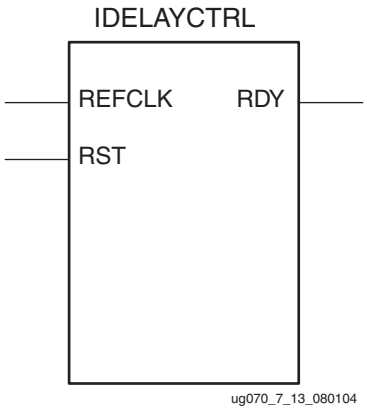


Figure 7-13: **IDELAYCTRL Primitive**

IDELAYCTRL Ports

RST - Reset

The reset input pin (RST) is an active-High asynchronous reset. IDELAYCTRL must be reset after configuration (and the REFCLK signal has stabilized) to ensure proper IDELAY operation. A reset pulse width $T_{IDELAYCTRL\_RPW}$ is required. IDELAYCTRL must be reset after configuration.

REFCLK - Reference Clock

The reference clock (REFCLK) provides a time reference to IDELAYCTRL to calibrate all IDELAY modules in the same region. This clock must be driven by a global clock buffer (BUFGCTRL). REFCLK must be $F_{IDELAYCTRL\_REF}$ ± the specified frequency variation in MHz (IDELAYCTRL_REF_PRECISION) to guarantee a specified IDELAY resolution ($T_{IDELAYRESOLUTION}$). REFCLK can be supplied directly from a user-supplied source or from the DCM, and must be routed on a global clock buffer. All valid M & D configurations are supported. Use the DCM Wizard to determine the correct settings in order to create the 200 MHz reference clock.

RDY - Ready

The ready (RDY) signal indicates when the IDELAY modules in the specific region are calibrated. The RDY signal is deasserted if REFCLK is held High or Low for one clock period or more. If RDY is deasserted Low, the IDELAYCTRL module must be reset. The implementation tools allow RDY to be unconnected/ignored. Figure 7-14 illustrates the timing relationship between RDY and RST.

IDELAYCTRL Timing

Table 7-10 shows the IDELAYCTRL switching characteristics.

Table 7-10: **IDELAYCTRL Switching Characteristics**

Symbol	Description
$F_{IDELAYCTRL\_REF}$	REFCLK frequency
IDELAYCTRL_REF_PRECISION	REFCLK precision
$T_{IDELAYCTRL\_RPW}$	Reset pulse width
$T_{IDELAYCTRLCO\_RDY}$	Reset/Startup to Ready for IDELAYCTRL

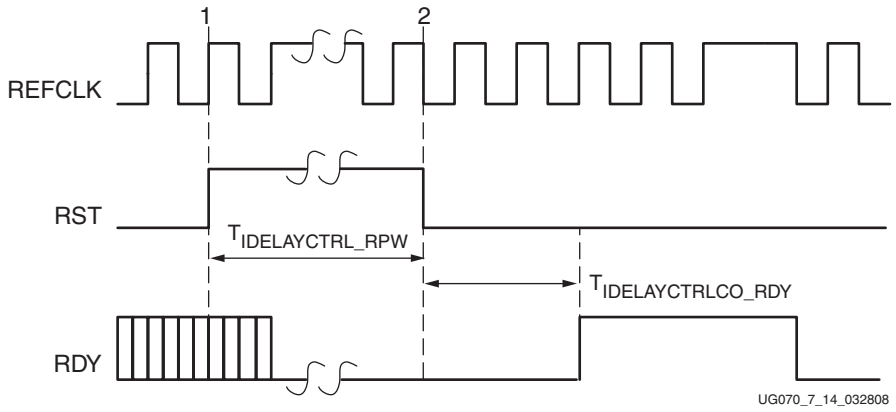


Figure 7-14: **Timing Relationship Between RST and RDY**

RST Event 1

- At RST Event 1, the RST pin is asserted.

RST Event 2

- At RST Event 2, the RST pin is deasserted.
- At $T_{IDELAYCTRLCO\_RDY}$, after RST Event 2, RDY is asserted High.

IDELAYCTRL Locations

IDELAYCTRL modules exist in every I/O column in every clock region. An IDELAYCTRL module calibrates all the IDELAY modules within its clock region. See "Global and Regional Clocks" in Chapter 1 for the definition of a clock region.

Figure 7-15 illustrates the relative locations of the IDELAYCTRL modules for an XC4VLX15 device.

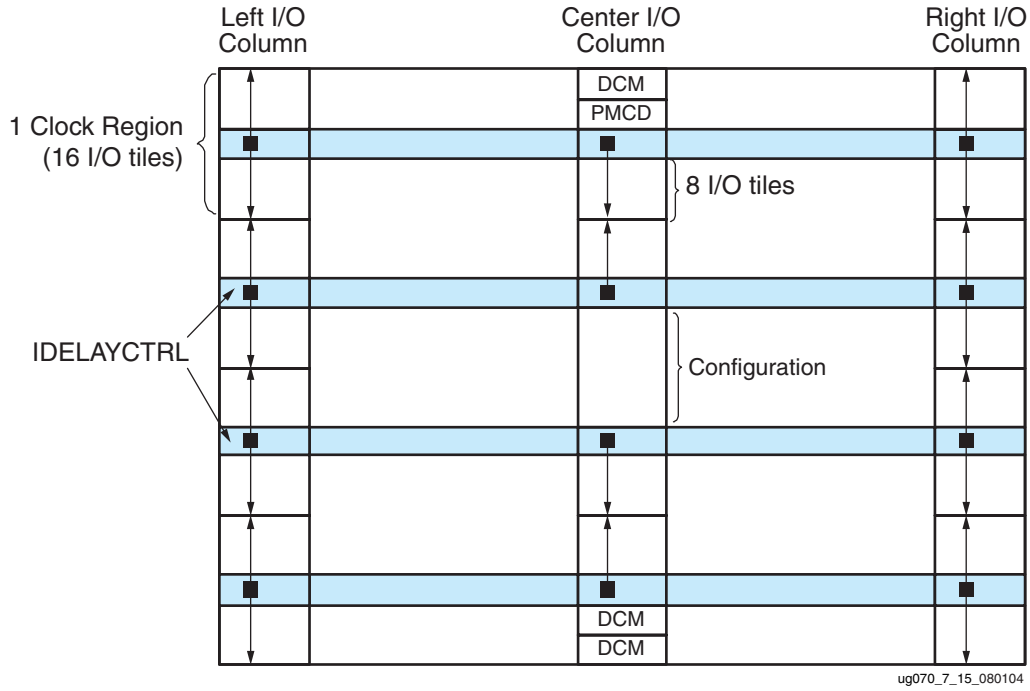


Figure 7-15: **Relative Locations of IDELAYCTRL Modules for an XC4VLX15 Device**

IDELAYCTRL Usage and Design Guidelines

This section describes using the Virtex-4 FPGA IDELAYCTRL modules, design guidelines, and recommended usage.

Instantiating IDELAYCTRL without LOC Constraints

When instantiating IDELAYCTRL without LOC constraints, the user must instantiate only one instance of IDELAYCTRL in the HDL design code. The implementation tools auto-replicate IDELAYCTRL instances throughout the entire device, even in clock regions not using the delay element. This results in higher power consumption due to higher resource utilization, the use of one global clock resource in every clock region, and a greater use of routing resources. The signals connected to the RST and REFCLK input ports of the instantiated IDELAYCTRL instance are connected to the corresponding input ports of the replicated IDELAYCTRL instances.

In ISE® software 9.1 Service Pack 1 and all later versions of the ISE tool, IDELAYCTRL instances that are replicated to clock regions where they are not needed are trimmed out of the design automatically by the ISE tool.

There are two special cases:

1. When the RDY port is ignored, the RDY signals of all the replacement IDELAYCTRL instances are left unconnected.

   The VHDL and Verilog use models for instantiating an IDELAYCTRL primitive without LOC constraints leaving the RDY output port unconnected are provided.

   VHDL Use Model

```
-- Only one instance of IDELAYCTRL primitive is instantiated.
-- The RDY port is left open

dlyctrl:IDELAYCTRL
          port map(
                    RDY => open,
                    REFCLK => refclk,
                    RST => rst
                   );
```

   Verilog Use Model

```
// Only one instance of IDELAYCTRL primitive is instantiated.
// The RDY port is left open

IDELAYCTRL dlyctrl (
                     .RDY(),
                     .REFCLK(refclk),
                     .RST(rst)
                    );
```

   The resulting circuitry after instantiating the IDELAYCTRL components is illustrated in Figure 7-16.

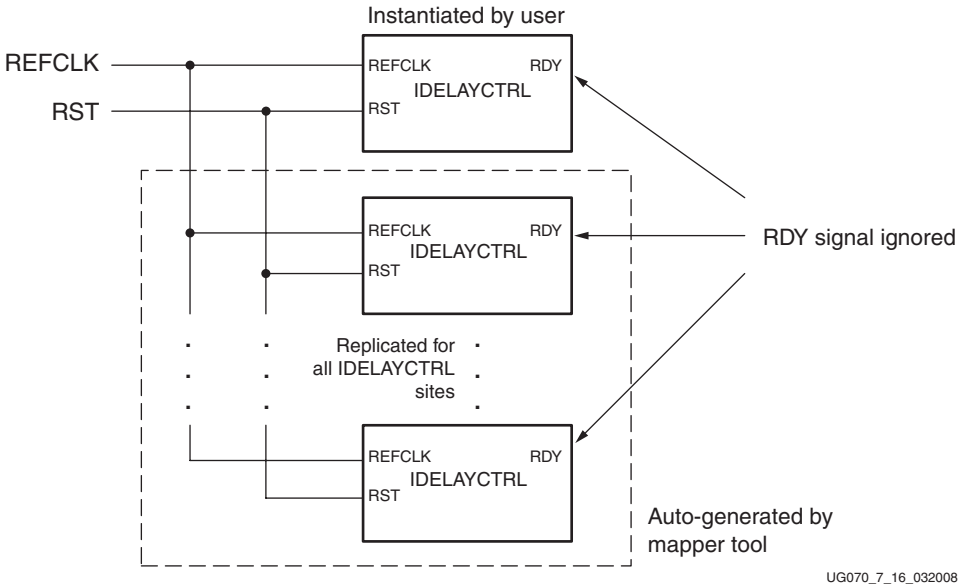


Figure 7-16: **Instantiate IDELAYCTRL without LOC Constraints - RDY Unconnected**

2. When RDY port is connected, an AND gate of width equal to the number of clock regions is instantiated and the RDY output ports from the instantiated and replicated IDELAYCTRL instances are connected to the inputs of the AND gate. The tools assign the signal name connected to the RDY port of the instantiated IDELAYCTRL instance to the output of the AND gate.

The VHDL and Verilog use models for instantiating an IDELAYCTRL primitive without LOC constraints with the RDY port connected are provided.

VHDL Use Model

```
-- Only one instance of IDELAYCTRL primitive is instantiated.
-- The RDY port is connected
dlyctrl:IDELAYCTRL
          port map(
                   RDY => rdy,
                   REFCLK => refclk,
                   RST => rst
                  );
```

Verilog Use Model

```
// Only one instance of IDELAYCTRL primitive is instantiated.
// The RDY port is connected
IDELAYCTRL dlyctrl (
                    .RDY(rdy),
                    .REFCLK(refclk),
                    .RST(rst)
                   );
```

The resulting circuitry after instantiating the IDELAYCTRL components is illustrated in Figure 7-17.

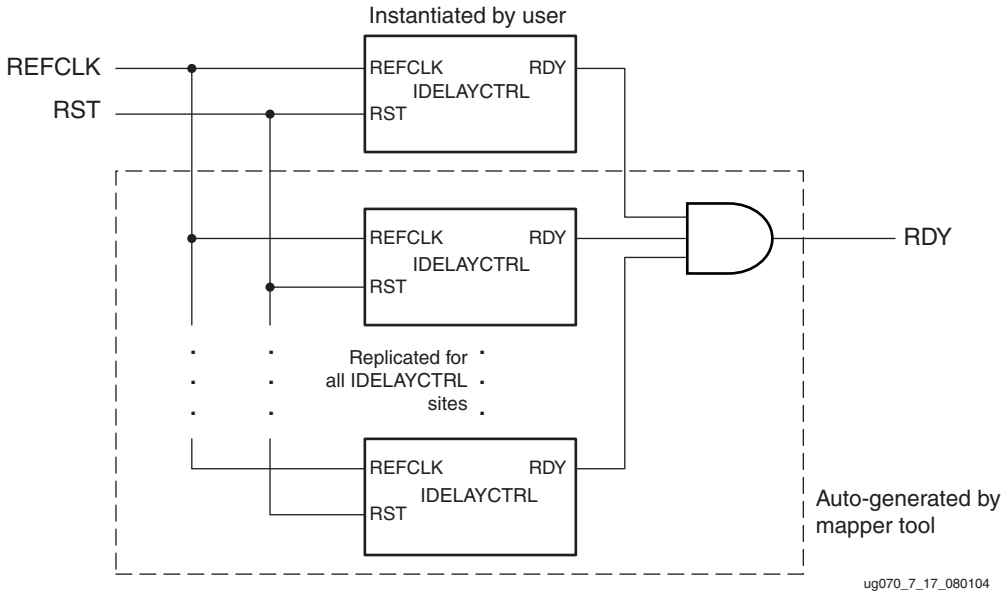


Figure 7-17: **Instantiate IDELAYCTRL Without LOC Constraints - RDY Connected**

Instantiating IDELAYCTRL with Location (LOC) Constraints

The most efficient way to use the IDELAYCTRL module is to define and lock down the placement of every IDELAYCTRL instance used in a design. This is done by instantiating the IDELAYCTRL instances with location (LOC) constraints. The user must define and lock placement of all ISERDES and IDELAY components using the delay element. (IOBDELAY_TYPE attribute set to FIXED or VARIABLE). Once completed, IDELAYCTRL sites can be chosen and LOC constraints assigned. Xilinx strongly recommends using IDELAYCTRL with a LOC constraint.

Location Constraints

Each IDELAYCTRL module has XY location coordinates (X:row, Y:column). To constrain placement, IDELAYCTRL instances can have LOC properties attached to them. The naming convention for IDELAYCTRL placement coordinates is different from the convention used in naming CLB locations. This allows LOC properties to transfer easily from array to array.

There are two methods of attaching LOC properties to IDELAYCTRL instances.

1. Insert LOC constraints in a UCF file
2. Embed LOC constraints directly into HDL design files

Inserting LOC Constraints in a UCF File

The following syntax is used for inserting LOC constraints in a UCF file.

```
INST "instance_name" LOC=IDELAYCTRL_X#Y#;
```

Embedding LOC Constraints Directly into HDL Design Files

The following syntax is used to embed LOC constraints into a Verilog design file.

```
// synthesis attribute loc of instance_name is "IDELAYCTRL_X#Y0#";
```

In VHDL code, the LOC constraint is described with VHDL attributes. Before it can be used, the constraint must be declared with the following syntax:

```
attribute loc : string;
```

Once declared, the LOC constraint can be specified as:

```
attribute loc of instance_name:label is "IDELAYCTRL_X#Y0#";
```

This section describes the VHDL and Verilog use models for instantiating IDELAYCTRL primitives with LOC constraints.

VHDL Use Model

```
-- Multiple instances of IDELAYCTRL primitives are instantiated.
-- Each instance has its own RST and RDY signal to allow for partial
-- reconfiguration.
-- The REFCLK signal is common to all instances
dlyctrl_1:IDELAYCTRL
          port map(
                   RDY => rdy _1,
                   REFCLK => refclk,
                   RST => rst_1
                  );
dlyctrl_2:IDELAYCTRL
          port map(
                   RDY => rdy _2,
                   REFCLK => refclk,
                   RST => rst_2
                  );
.
.
.
dlyctrl_n:IDELAYCTRL
          port map(
                   RDY => rdy _n,
                   REFCLK => refclk,
```

```
                    RST => rst_n
                   );
-- The user either declares the LOC constraints in the
-- VHDL design file, or in the UCF file.

-- Declaring LOC constraints in the VHDL file.
attribute loc : string;
attribute loc of dlyctrl_1:label is "IDELAYCTRL_X0Y0";
attribute loc of dlyctrl_2:label is "IDELAYCTRL_X0Y1";
.
.
.
attribute loc of dlyctrl_n:label is "IDELAYCTRL_XnYn";

-- Declaring LOC constraints in the UCF file.
INST "dlyctrl_1" LOC=IDELAYCTRL_X0Y0;
INST "dlyctrl_2" LOC=IDELAYCTRL_X0Y1;
.
.
.
INST "dlyctrl_n" LOC=IDELAYCTRL_XnYn;
```

Verilog Use Model

```
// Multiple instances of IDELAYCTRL primitives are instantiated.
// Each instance has its own RST and RDY signal to allow for partial
// reconfiguration.
// The REFCLK signal is common to all instances
IDELAYCTRL dlyctrl_1 (
                      .RDY(rdy _1),
                      .REFCLK(refclk),
                      .RST(rst_1)
                     );
IDELAYCTRL dlyctrl_2 (
                      .RDY(rdy_2),
                      .REFCLK(refclk),
                      .RST(rst_2)
                     );
.
.
.
IDELAYCTRL dlyctrl_n (
                      .RDY(rdy_n),
                      .REFCLK(refclk),
                      .RST(rst_n)
                     );
// The user either declares the LOC constraints in the
// Verilog design file or in the following UCF file.

// Declaring LOC constraints in the Verilog file.
// synthesis attribute loc of dlyctrl_1 is "IDELAYCTRL_X0Y0";
// synthesis attribute loc of dlyctrl_2 is "IDELAYCTRL_X0Y1";
.
.
.
// synthesis attribute loc of dlyctrl_N is "IDELAYCTRL_XnYn";

// Declaring LOC constraints in the UCF file:
INST "dlyctrl_1" LOC=IDELAYCTRL_X0Y0;
```

```
INST "dlyctrl_2" LOC=IDELAYCTRL_X0Y1;
.
.
.
INST "dlyctrl_n" LOC=IDELAYCTRL_XnYn;
```

The circuitry that results from instantiating the IDELAYCTRL components is shown in Figure 7-18.

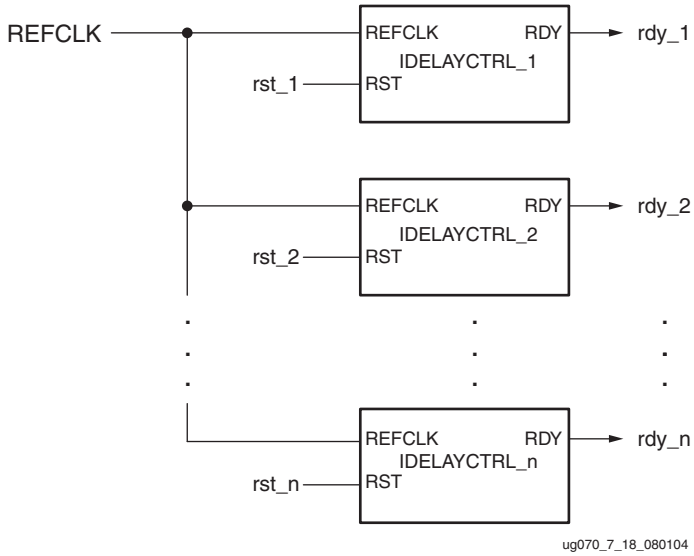


Figure 7-18: **Instantiate IDELAYCTRL with LOC Constraint**

Instantiating IDELAYCTRL with and without LOC Constraints

There are cases where the user instantiates an IDELAYCTRL module with a LOC constraint but also instantiates an IDELAYCTRL module without a LOC constraint. In the case where an IP Core is instantiated with a non-location constrained IDELAYCTRL module and also wants to instantiate an IDELAYCTRL module without a LOC constraint for another part of the design, the implementation tools will perform the following:

- Instantiate the LOC IDELAYCTRL instances as described in the section Instantiating IDELAYCTRL with Location (LOC) Constraints.
- Replicate the non-location constrained IDELAYCTRL instance to populate with an IDELAYCTRL instance in every clock region without a location constrained IDELAYCTRL instance in place.
- The signals connected to the RST and REFCLK input ports of the non-location constrained IDELAYCTRL instance are connected to the corresponding input ports of the replicated IDELAYCTRL instances.
- If the RDY port of the non-location constrained IDELAYCTRL instance is ignored, then all the RDY signals of the replicated IDELAYCTRL instances are also ignored.
- If the RDY port of the non-location constrained IDELAYCTRL instance is connected, then the RDY port of the non-location constrained instance plus the RDY ports of the replicated instances are connected to an auto-generated AND gate. The implementation tools assign the signal name connected to the RDY port of the non-location constrained instance to the output of the AND gate.
- All the ports of the location constrained instances (RST, REFCLK, and RDY) are independent from each other and from the replicated instances.

The VHDL and Verilog use models for instantiating a mixed usage model are provided. In the example, a user is instantiating a non-location constrained IDELAYCTRL instance with the RDY signal connected. This discussion is also valid when the RDY signal is ignored.

VHDL Use Model

```
-- Multiple instantiations of IDELAYCTRL primitives with LOC
-- constraints.
-- Each instance has its own RST and RDY signal to allow for partial
-- reconfiguration.
-- The REFCLK signal is common to all instances (LOC and replicated
-- instances)
dlyctrl_1:IDELAYCTRL
          port map(
                   RDY => rdy _1,
                   REFCLK => refclk,
                   RST => rst_1
                  );
dlyctrl_2:IDELAYCTRL
          port map(
                   RDY => rdy _2,
                   REFCLK => refclk,
                   RST => rst_2
                  );
.
.
.
dlyctrl_n:IDELAYCTRL
          port map(
                   RDY => rdy _n,
                   REFCLK => refclk,
                   RST => rst_n
                  );
-- The user should either declare the LOC constraints in the
-- VHDL design file or in the UCF file.
-- Declaring LOC constraints in the VHDL file.
attribute loc : string;
attribute loc of dlyctrl_1:label is "IDELAYCTRL_X0Y0";
attribute loc of dlyctrl_2:label is "IDELAYCTRL_X0Y1";
.
.
.
attribute loc of dlyctrl_n:label is "IDELAYCTRL_XnYn";
-- Declaring LOC constraints in the UCF file:
INST "dlyctrl_1" LOC=IDELAYCTRL_X0Y0;
INST "dlyctrl_2" LOC=IDELAYCTRL_X0Y1;
.
.
.
INST "dlyctrl_n" LOC=IDELAYCTRL_XnYn;

-- One instantiation of an IDELAYCTRL primitive without LOC constraint
-- RST and RDY port signals are independent from LOC-ed instances
dlyctrl_noloc:IDELAYCTRL
          port map(
                   RDY => rdy_noloc,
                   REFCLK => refclk,
                   RST => rst_noloc
                  );
```

Verilog Use Model

```
// Multiple instantiations of IDELAYCTRL primitives with LOC contraints
// Each instance has its own RST and RDY signal to allow for partial
// reconfiguration.
// The REFCLK signal is common to all instances (LOC and replicated
// instances)
IDELAYCTRL dlyctrl_1 (
                      .RDY(rdy _1),
                      .REFCLK(refclk),
                      .RST(rst_1)
                     );
IDELAYCTRL dlyctrl_2 (
                      .RDY(rdy_2),
                      .REFCLK(refclk),
                      .RST(rst_2)
                     );
.
.
.
IDELAYCTRL dlyctrl_n (
                      .RDY(rdy_n),
                      .REFCLK(refclk),
                      .RST(rst_n)
                     );
// The user should either declare the LOC constraints in the
// Verilog design file or in the UCF file.
// Declaring LOC constraints in the Verilog file.
// synthesis attribute loc of dlyctrl_1 is "IDELAYCTRL_X0Y0";
// synthesis attribute loc of dlyctrl_2 is "IDELAYCTRL_X0Y1";
.
.
.
// synthesis attribute loc of dlyctrl_N is "IDELAYCTRL_XnYn";
// Declaring LOC constraints in the UCF file:
INST "dlyctrl_1" LOC=IDELAYCTRL_X0Y0;
INST "dlyctrl_2" LOC=IDELAYCTRL_X0Y1;
.
.
.
INST "dlyctrl_n" LOC=IDELAYCTRL_XnYn;

// One instantiation of an IDELAYCTRL primitive without LOC constraint
// RST and RDY port signals are independent from LOC-ed instances
IDELAYCTRL dlyctrl_noloc (
                          .RDY(rdy_noloc),
                          .REFCLK(refclk),
                          .RST(rst_noloc)
                         );
```

The circuitry that results from instantiating the IDELAYCTRL components as shown is illustrated in Figure 7-19.

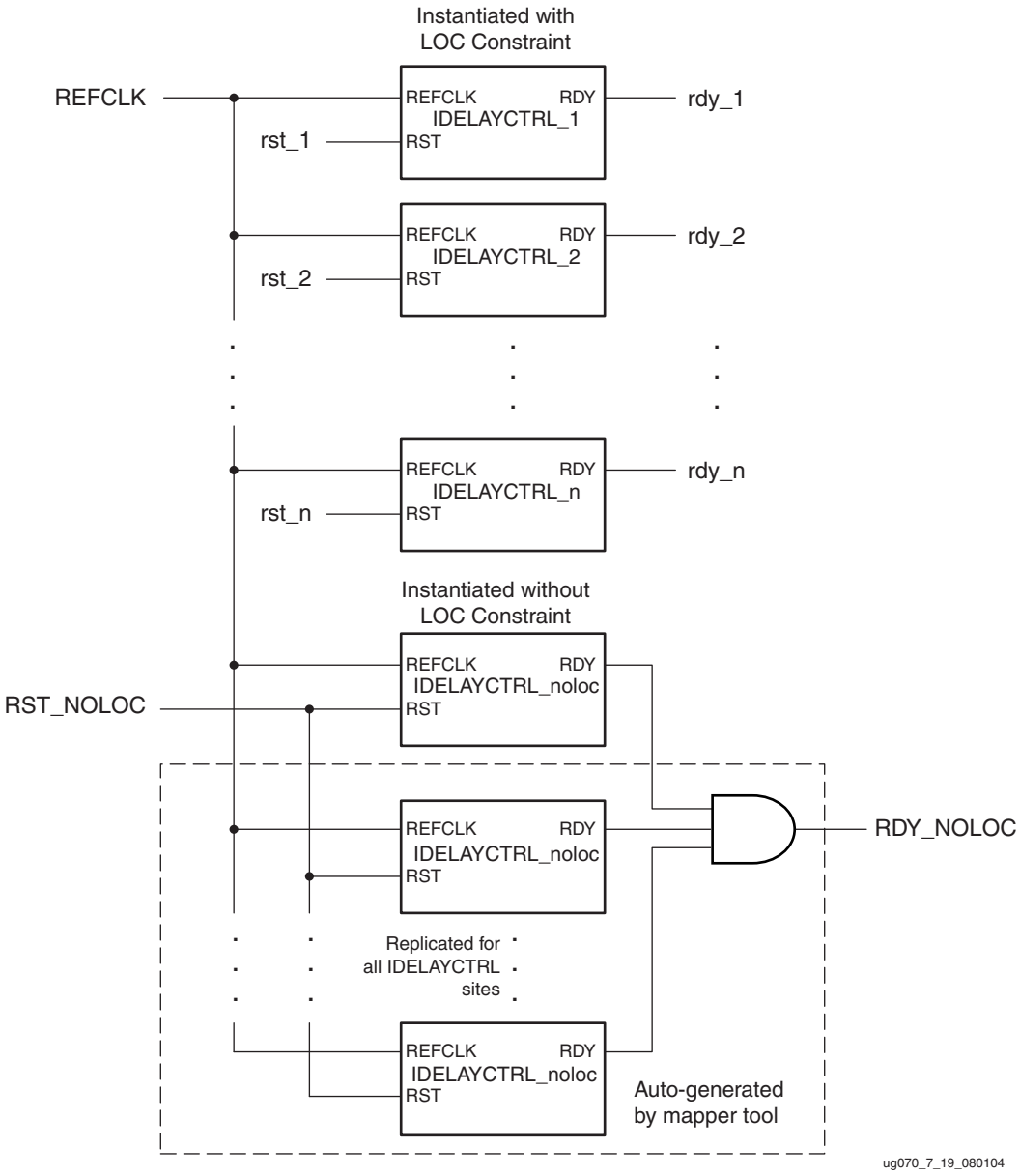


Figure 7-19: **Mixed Instantiation of IDELAYCTRL Elements**

Instantiating Multiple IDELAYCTRLs without LOC Constraints

Instantiating multiple IDELAYCTRL instances without LOC properties is prohibited. If this occurs, an error is issued by the implementation tools.

OLOGIC Resources

OLOGIC blocks include six storage elements (shown in Figure 7-20.) The top three registers (TFF1, TFF2, and TFF3) are used for 3-state control. The bottom three registers (OFF1, OFF2, and OFF3) are used for data output. Both sets of registers are functionally the same.

To build an edge-triggered D-type flip-flop, use the topmost register (OFF1/TFF1). This register is also the only register that can be configured as a level sensitive latch. The other two registers (OFF2/TFF2 and OFF3/TFF3) are used to build various output DDR registers. See "Output DDR Overview (ODDR)," page 354 for further discussion on output DDR.

The three data registers share a common clock enable (OCE). Similarly, the three 3-state control registers share a different clock enable (TCE). The clock enable signals are default active High. If left unconnected, the clock enable pin for the storage element defaults to the active state.

All registers in OLOGIC have a common clock and synchronous or asynchronous set and reset (SR and REV signals). Table 7-1 and Table 7-2 describe the operation of SR in conjunction with REV.

For each storage element in the OLOGIC block, the SRVAL attributes are independent. Synchronous or asynchronous set/reset (SRTYPE) can not be set individually for each storage element in an OLOGIC block.

Most of the control signals have optional inverter. Any inverter placed on a control input is automatically absorbed.

Figure 7-20 illustrates the various logic resources in the OLOGIC block.

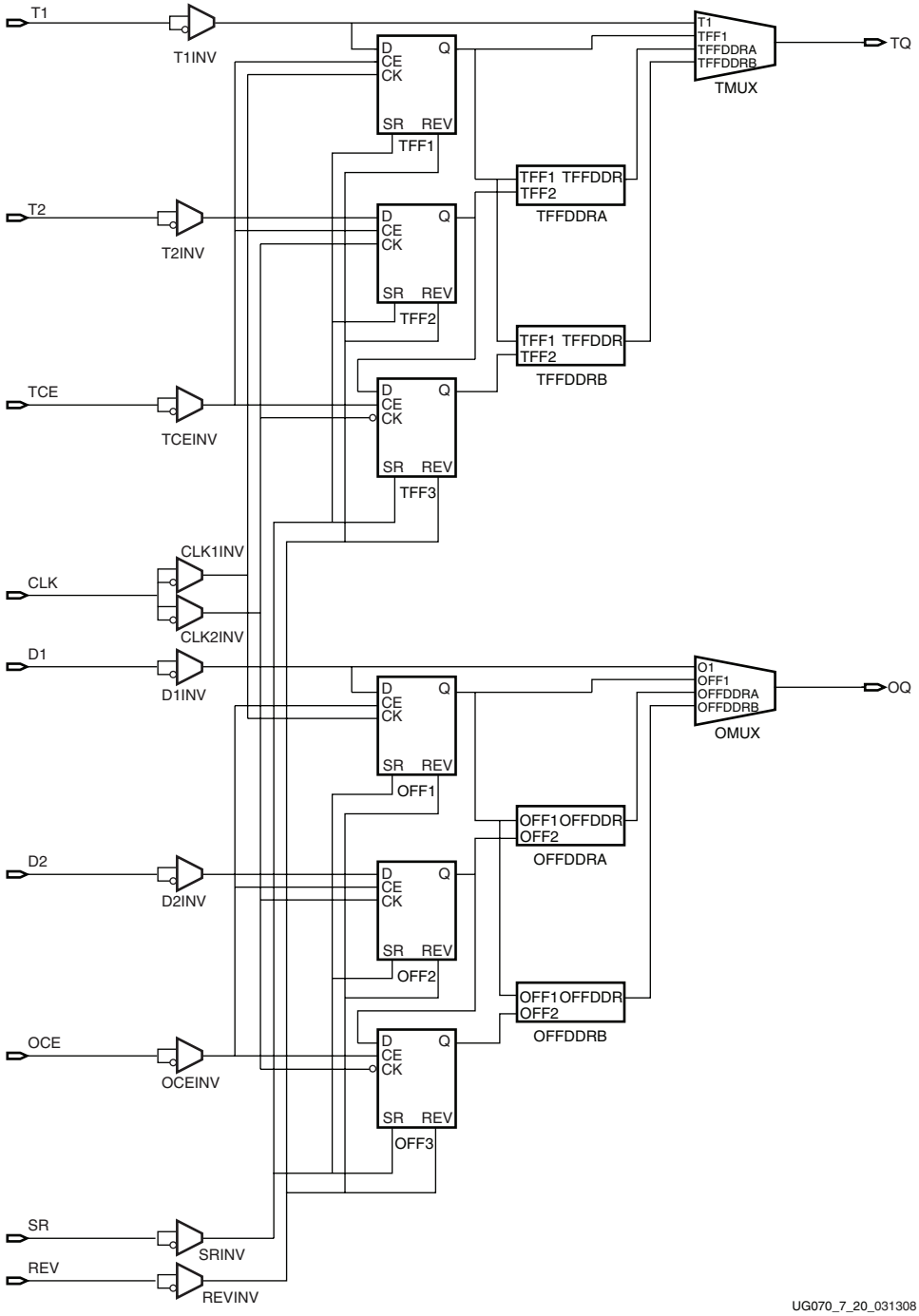


Figure 7-20: **OLOGIC Block Diagram**

This section of the documentation discusses the various features available using the OLOGIC resources. All connections between the OLOGIC resources are managed in Xilinx software.

Combinatorial Output Data and 3-State Control Path

The combinatorial output paths create a direct connection from the FPGA fabric to the output driver or output driver control. These paths is used when:

1. There is direct (unregistered) connection from logic resources in the FPGA fabric to the output data or 3-state control.
2. The "pack I/O register/latches into IOBs" is set to OFF.

Output DDR Overview (ODDR)

Virtex-4 devices have dedicated registers in the OLOGIC to implement output DDR registers. This feature is accessed when instantiating the ODDR primitive. DDR multiplexing is automatic when using OLOGIC. No manual control of the mux-select is needed. This control is generated from the clock.

There is only one clock input to the ODDR primitive. Falling edge data is clocked by a locally inverted version of the input clock. All clocks feeding into the I/O tile are fully multiplexed, i.e., there is no clock sharing between ILOGIC or OLOGIC blocks. The ODDR primitive supports the following modes of operation:

- OPPOSITE_EDGE mode
- SAME_EDGE mode

The SAME_EDGE mode is new for the Virtex-4 architecture. This new mode allows designers to present both data inputs to the ODDR primitive on the rising-edge of the ODDR clock, saving CLB and clock resources, and increasing performance. This mode is implemented using the DDR_CLK_EDGE attribute. It is supported for 3-state control as well. The following sections describe each of the modes in detail.

OPPOSITE_EDGE Mode

In OPPOSITE_EDGE mode, two output registers are used to clock data from the FPGA fabric at twice the throughput of a single rising-edge clocking scheme.

Both registers are rising-edge triggered. A second register receives an inverted version of the clock. Both register outputs are then multiplexed and presented to the data input or 3-state control input of the IOB. This structure is similar to the Virtex-II and Virtex-II Pro FPGA implementation. The simplified output DDR registers and the signals associated with the OPPOSITE_EDGE mode are shown in Figure 7-21.

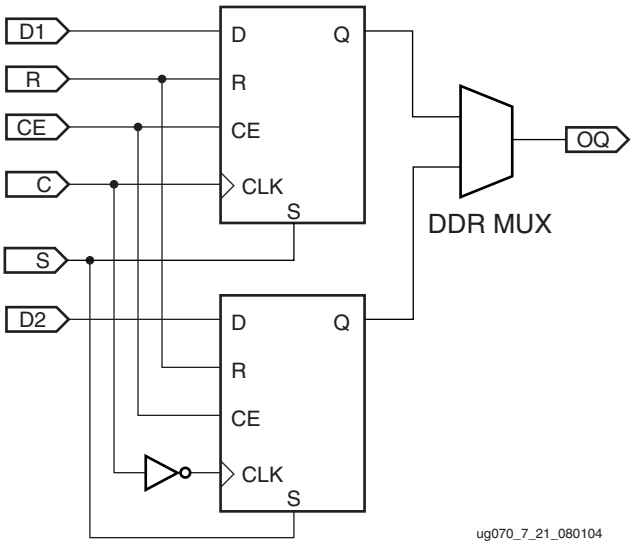


Figure 7-21: **Output DDR in OPPOSITE_EDGE Mode**

The timing diagram of the output DDR using the OPPOSITE_EDGE mode is shown in Figure 7-22.

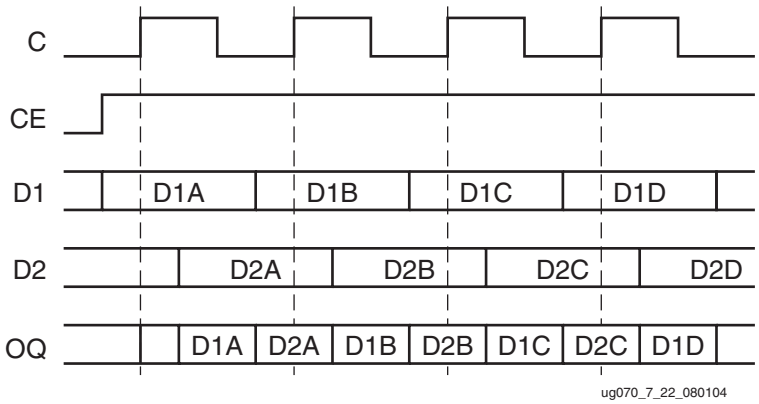


Figure 7-22: **Output DDR Timing in OPPOSITE_EDGE Mode**

SAME_EDGE Mode

In SAME_EDGE mode, a third register (OFF3 or TFF3), clocked by a rising edge clock, is placed on the input of the falling edge register. The output DDR registers and the signals associated with the SAME_EDGE mode are shown in Figure 7-23.

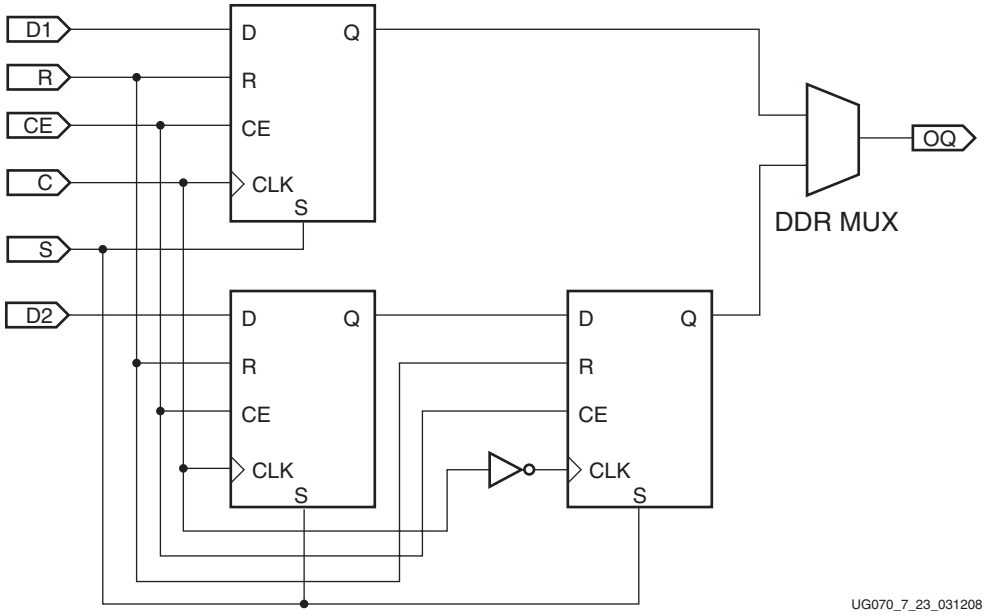


Figure 7-23: **Output DDR in SAME_EDGE Mode**

Using this scheme, data can now be presented to the IOB on the same clock edge. Presenting the data to the IOB on the same clock edge avoids setup time violations and allows the user to perform higher DDR frequency with minimal register to register delay, as opposed to using the CLB registers. The additional register is used to maintain an alternating bits output of DATA_1 and DATA_2 on the DDR multiplexer. Figure 7-24 shows the timing diagram of the output DDR using the SAME_EDGE mode.

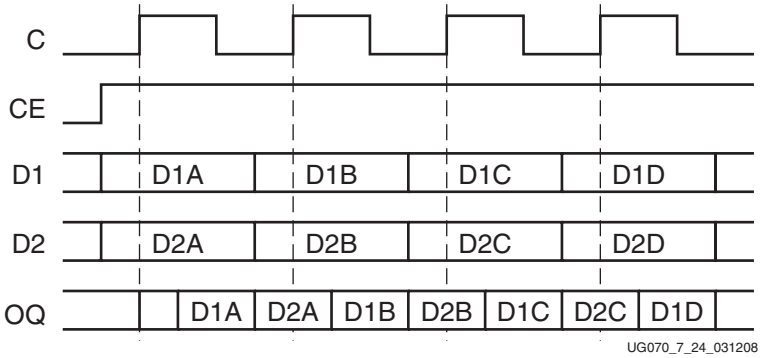


Figure 7-24: **Output DDR Timing in SAME_EDGE Mode**

Clock Forwarding

Output DDR can forward a copy of the clock to the output. This is useful for propagating a clock and DDR data with identical delays, and for multiple clock generation, where every clock load has a unique clock driver. This is accomplished by tying the D1 input of the ODDR primitive Low, and the D2 input High. Xilinx recommends using this scheme to forward clocks from the FPGA fabric to the output pins.

Output DDR Primitive (ODDR)

Figure 7-25 shows the ODDR primitive block diagram. Table 7-11 lists the ODDR port signals. Table 7-12 describes the various attributes available and default values for the ODDR primitive.

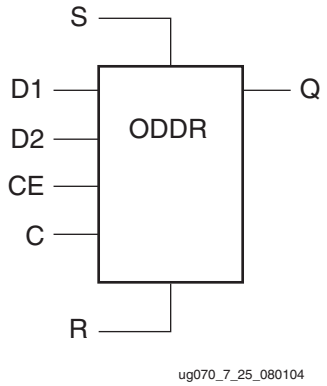


Figure 7-25: **ODDR Primitive Block Diagram**

Table 7-11: **ODDR Port Signals**

Port Name	Function	Description
Q	Data output (DDR)	ODDR register output.
C	Clock input port	The C pin represents the clock input pin.
CE	Clock enable port	CE represents the clock enable pin. When asserted Low, this port disables the output clock driving port Q.
D1 and D2	Data inputs	ODDR register inputs.
R	Reset	Synchronous/Asynchronous reset pin. Reset is asserted High.
S	Set	Synchronous/Asynchronous set pin. Set is asserted High.

Table 7-12: **ODDR Attributes**

Attribute Name	Description	Possible Values
DDR_CLK_EDGE	Sets the ODDR mode of operation with respect to clock edge	OPPOSITE_EDGE (default), SAME_EDGE
INIT	Sets the initial value for Q port	0 (default), 1
SRTYPE	Set/Reset type with respect to clock (C)	ASYNC, SYNC (default)

ODDR VHDL and Verilog Templates

The following examples illustrate the instantiation of the OSERDES module in VHDL and Verilog.

ODDR VHDL Template

```
--Example ODDR component declaration

component ODDR
  generic(
      DDR_CLK_EDGE : string := "OPPOSITE_EDGE";
      INIT         : bit    := '0';
      SRTYPE       : string := "SYNC";
    );

  port(
      Q           : out std_ulogic;

      C           : in  std_ulogic;
      CE          : in  std_ulogic;
      D1          : in  std_ulogic;
      D2          : in  std_ulogic;
      R           : in  std_ulogic;
      S           : in  std_ulogic
    );
  end component;

--Example ODDR instantiation

U_ODDR : ODDR
Port map(
      Q => user_q,
      C => user_c,
      CE => user_ce,
      D1 => user_d1,
      D2 => user_d2,
      R => user_r,
      S => user_s
    );
```

ODDR Verilog Template

```
//Example ODDR module declaration

module ODDR (Q, C, CE, D1, D2, R, S);

    output Q;

    input C;
    input CE;
    input D1;
    input D2;
    tri0 GSR = glbl.GSR;
    input R;
    input S;

    parameter DDR_CLK_EDGE = "OPPOSITE_EDGE";
```

```
    parameter INIT = 1'b0;
    parameter SRTYPE = "SYNC";

endmodule;

//Example ODDR instantiation
ODDR U_ODDR(
.Q(user_q),
.C(user_c),
.CE(user_ce),
.D1(user_d1),
.D2(user_d2),
.R(user_r),
.S(user_s)
);
```

OLOGIC Timing Models

This section discusses all timing models associated with the OLOGIC block. Table 7-13 describes the function and control signals of the OLOGIC switching characteristics in the *Virtex-4 Data Sheet*.

Table 7-13: **OLOGIC Switching Characteristics**

Symbol	Description
Setup/Hold	
T_{ODCK}/T_{OCKD}	D1/D2 pins setup/hold with respect to CLK
T_{OOCECK}/T_{OCKOCE}	OCE pin setup/hold with respect to CLK
T_{OSRCK}/T_{OCKSR}	SR/REV pin setup/hold with respect to CLK
T_{OTCK}/T_{OCKT}	T1/T2 pins setup/hold with respect to CLK
T_{OTCECK}/T_{OCKTCE}	TCE pin setup/hold with respect to CLK
Clock to Out	
T_{OCKQ}	CLK to OQ/TQ out
T_{RQ}	SR/REV pin to OQ/TQ out

Timing Characteristics

Figure 7-26 illustrates the OLOGIC output register timing.

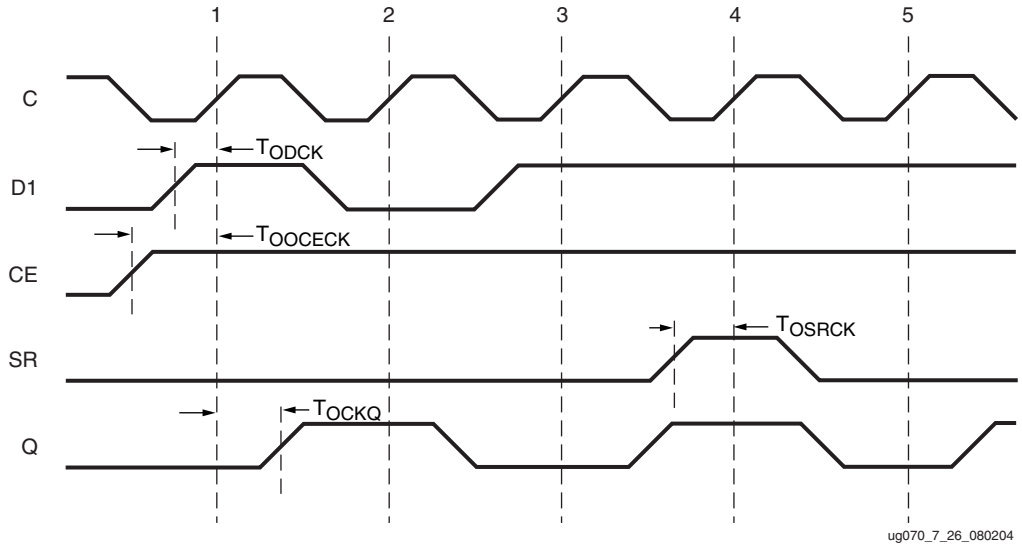


Figure 7-26: **OLOGIC Output Register Timing Characteristics**

Clock Event 1

- At time T_{OOCECK} before Clock Event 1, the output clock enable signal becomes valid-High at the CE input of the output register, enabling the output register for incoming data.
- At time T_{ODCK} before Clock Event 1, the output signal becomes valid-High at the D1 input of the output register and is reflected at the Q output at time T_{OCKQ} after Clock Event 1.

Clock Event 4

At time T_{OSRCK} before Clock Event 4, the SR signal (configured as synchronous reset in this case) becomes valid-High, resetting the output register and reflected at the Q output at time T_{RQ} after Clock Event 4.

Figure 7-27 illustrates the OLOGIC ODDR register timing.

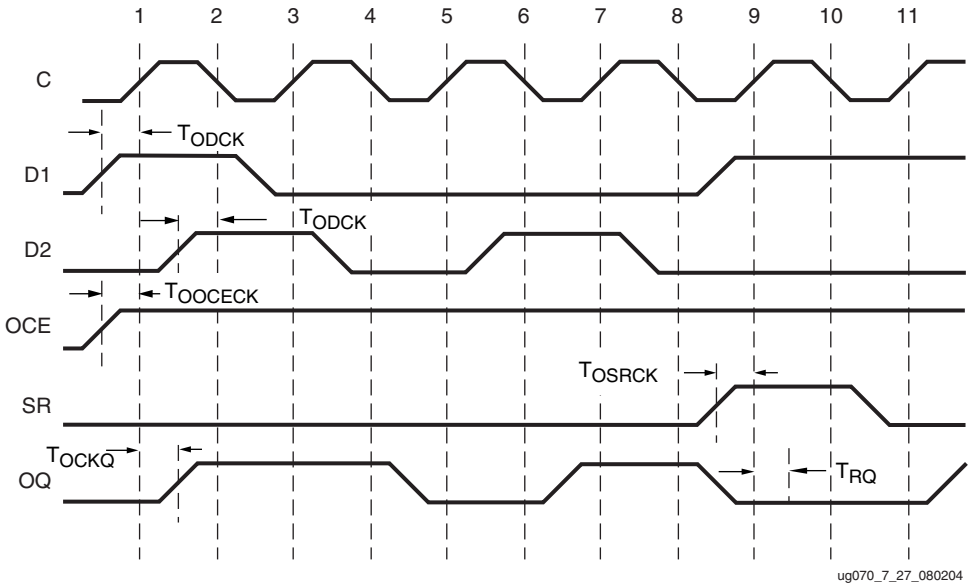


Figure 7-27: **OLOGIC ODDR Register Timing Characteristics**

Clock Event 1

- At time T_{OOCECK} before Clock Event 1, the ODDR clock enable signal becomes valid-High at the OCE input of the ODDR registers, enabling them for incoming data. Since the OCE signal is common to all ODDR registers, care must be taken to toggle this signal between the rising edges and falling edges of C as well as meeting the register setup-time relative to both clock edges.
- At time T_{ODCK} before Clock Event 1 (rising edge of C), the data signal D1 becomes valid-High at the D1 input of ODDR register 1 and is reflected on the OQ output at time T_{OCKQ} after Clock Event 1.

Clock Event 2

- At time T_{ODCK} before Clock Event 2 (falling edge of C), the data signal D2 becomes valid-High at the D2 input of ODDR register 2 and is reflected on the OQ output at time T_{OCKQ} after Clock Event 2 (no change at the OQ output in this case).

Clock Event 9

At time T_{OSRCK} before Clock Event 9 (rising edge of C), the SR signal (configured as synchronous reset in this case) becomes valid-High resetting ODDR Register 1, reflected at the OQ output at time T_{RQ} after Clock Event 9 (no change at the OQ output in this case) and resetting ODDR Register 2, reflected at the OQ output at time T_{RQ} after Clock Event 10 (no change at the OQ output in this case).

Figure 7-28 illustrates the OLOGIC 3-state register timing.

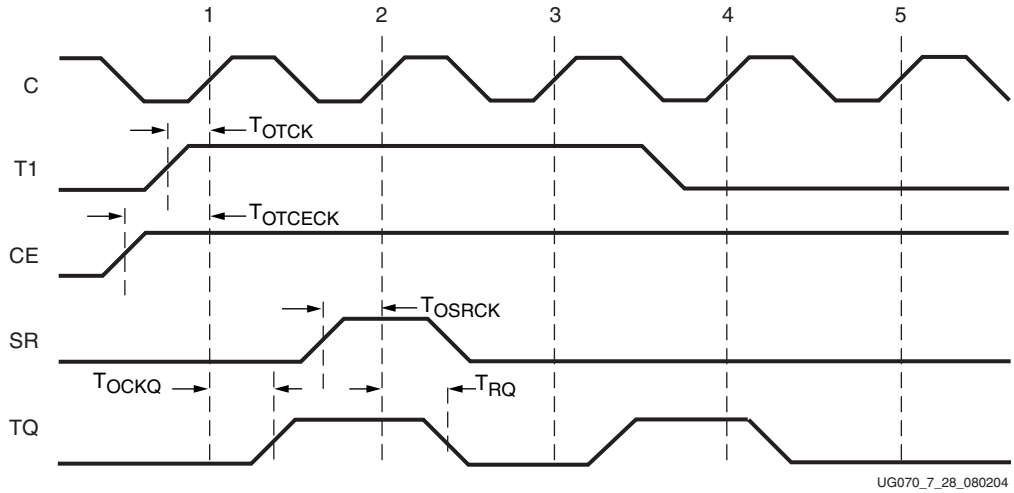


Figure 7-28: **OLOGIC 3-State Register Timing Characteristics**

Clock Event 1

- At time T_{OTCECK} before Clock Event 1, the 3-state clock enable signal becomes valid-High at the TCE input of the 3-state register, enabling the 3-state register for incoming data.
- At time T_{OTCK} before Clock Event 1 the 3-state signal becomes valid-High at the T input of the 3-state register, returning the pad to high-impedance at time T_{OCKQ} after Clock Event 1.

Clock Event 2

- At time T_{OSRCK} before Clock Event 2, the SR signal (configured as synchronous reset in this case) becomes valid-High, resetting the 3-state register at time T_{RQ} after Clock Event 2.

Figure 7-29 illustrates IOB DDR 3-state register timing. This example is shown using DDR in opposite edge mode. For other modes add the appropriate latencies as shown in Figure 7-7, page 326.

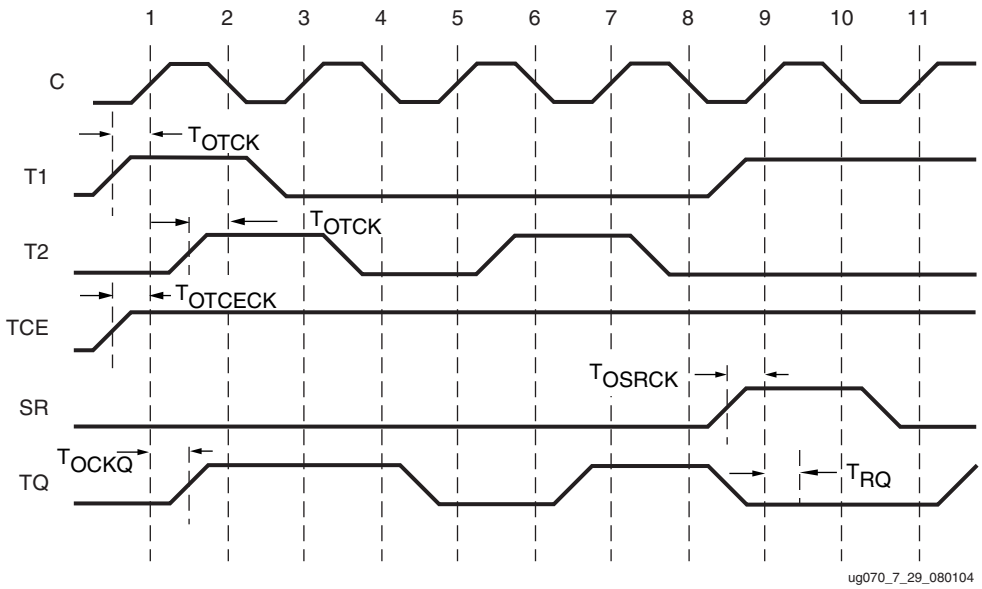


Figure 7-29: **OLOGIC ODDR 3-State Register Timing Characteristics**

Clock Event 1

- At time T_{OTCECK} before Clock Event 1, the 3-state clock enable signal becomes valid-High at the TCE input of the ODDR 3-state registers, enabling them for incoming data. Since the TCE signal is common to all ODDR registers, care must be taken to toggle this signal between the rising edges and falling edges of C as well as meeting the register setup-time relative to both clock edges.
- At time T_{OTCK} before Clock Event 1 (rising edge of C), the 3-state signal T1 becomes valid-High at the T1 input of 3-state register 1 and is reflected on the TQ output at time T_{OCKQ} after Clock Event 1.

Clock Event 2

- At time T_{OTCK} before Clock Event 2 (falling edge of C), the 3-state signal T2 becomes valid-High at the T2 input of 3-state register 2 and is reflected on the TQ output at time T_{OCKQ} after Clock Event 2 (no change at the TQ output in this case).

Clock Event 9

- At time T_{OSRCK} before Clock Event 9 (rising edge of C), the SR signal (configured as synchronous reset in this case) becomes valid-High resetting 3-state Register 1, reflected at the TQ output at time T_{RQ} after Clock Event 9 (no change at the TQ output in this case) and resetting 3-state Register 2, reflected at the TQ output at time T_{RQ} after Clock Event 10 (no change at the TQ output in this case)

XILINX®

Chapter 8

Advanced SelectIO Logic Resources

Introduction

The Virtex-4 FPGA I/O functionality is described in Chapter 6 through Chapter 8 of this user guide.

- Chapter 6 covers the electrical characteristics of input receivers and output drivers, and their compliance with many industry standards.
- Chapter 7 describes the register structures dedicated for sending and receiving SDR or DDR data.

This chapter covers additional Virtex-4 FPGA resources:

- Input serial-to-parallel converters (ISERDES) and output parallel-to-series converters (OSERDES) support very fast I/O data rates, and allow the internal logic to run up to ten times slower than the I/O.
- The Bitslip submodule can re-align data to word boundaries, detected with the help of a training pattern.

Input Serial-to-Parallel Logic Resources (ISERDES)

The Virtex-4 FPGA ISERDES is a dedicated serial-to-parallel converter with specific clocking and logic features designed to facilitate the implementation of high-speed source-synchronous applications. The ISERDES avoids the additional timing complexities encountered when designing deserializers in the FPGA logic.

ISERDES features include:

- Dedicated Deserializer/Serial-to-Parallel Converter

  The ISERDES deserializer enables high-speed data transfer without requiring the FPGA fabric to match the input data frequency. This converter supports both single data rate (SDR) and double data rate (DDR) modes. In SDR mode, the serial-to-parallel converter creates a 2-, 3-, 4-, 5-, 6-, 7-, or 8-bit wide parallel word. In DDR mode, the serial-to-parallel converter creates a 4-, 6-, 8-, or 10-bit-wide parallel word.

- Digitally Controlled Delay Element – IDELAY

  Every ISERDES block contains a programmable absolute delay element called IDELAY. IDELAY is a 64-tap, wraparound, delay element with a fixed, guaranteed tap resolution (see *Virtex-4 Data Sheet*). It can be applied to the combinatorial input path, registered input path, or both. There are three modes of operation:

  a. DEFAULT – Zero-hold time delay mode (similar to the Virtex®-II and Virtex-II Pro FPGA delay elements)

  b. FIXED – Delay value is set to the value in the IOBDELAY _VALUE

c. VARIABLE – Delay value can be changed at run-time by manipulating a set of control signals

The section "Input Delay Element (IDELAY)" in Chapter 7 discusses IDELAY in detail.

- Bitslip Submodule

  The Bitslip submodule allows designers to reorder the sequence of the parallel data stream going into the FPGA logic. This can be used for training source-synchronous interfaces that include a training pattern.

- Dedicated Support for Strobe-based Memory Interfaces

  ISERDES contains dedicated circuitry (including the OCLK input pin) to handle the strobe-to-FPGA clock domain crossover entirely within the ISERDES block. This allows for higher performance and a simplified implementation.

- Dedicated support for Networking interfaces.

Figure 8-1 shows the block diagram of the ISERDES, highlighting all the major components and features of the block.

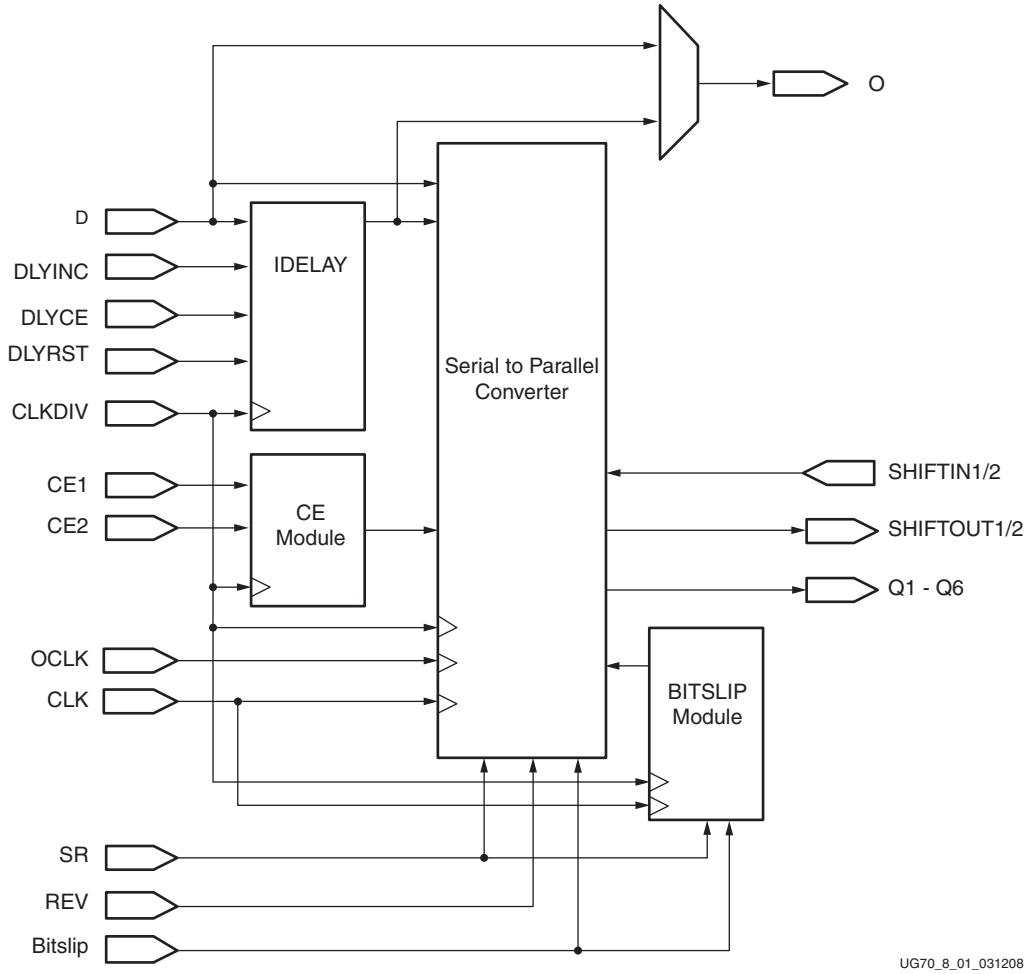


Figure 8-1: **ISERDES Block Diagram**

ISERDES Primitive

Figure 8-2 shows the ISERDES primitive.

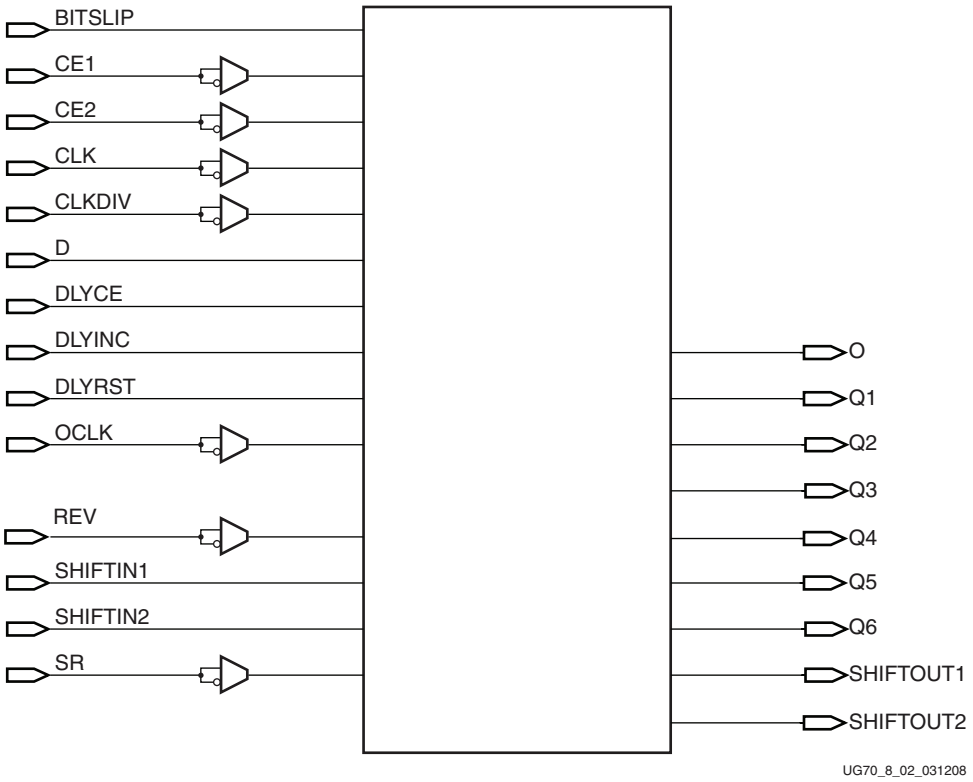


Figure 8-2: **ISERDES Primitive**

Table 8-1 lists the available ports in the ISERDES primitive.

Table 8-1: **ISERDES Port List and Definitions**

Port Name	Type	Width	Description
O	Output	1	Combinatorial output.
Q1 – Q6	Output	1 (each)	Registered outputs.
SHIFTOUT1	Output	1	Carry out for data width expansion. Connect to SHIFTIN1 of slave IOB. See "ISERDES Width Expansion."
SHIFTOUT2	Output	1	Carry out for data width expansion. Connect to SHIFTIN2 of slave IOB. See "ISERDES Width Expansion."
BITSLIP	Input	1	Invokes the Bitslip operation.
CE1, CE2	Input	1 (each)	Clock enable inputs.
CLK	Input	1	High-speed clock input. Clocks serial input data stream.
CLKDIV	Input	1	Divided clock input. Clocks delay element, deserialized data, Bitslip submodule, and CE unit.
D	Input	1	Serial input data from IOB.

Table 8-1: **ISERDES Port List and Definitions *(Continued)***

Port Name	Type	Width	Description
DLYCE	Input	1	Enable IDELAY increment/decrement function. The DLYCE port is the same as the CE port in the IDELAY primitive. See "IDELAY Ports".
DLYINC	Input	1	Increment/decrement number of tap delays in IDELAY. The DLYINC port is the same as the INC port in the IDELAY primitive. See "IDELAY Ports".
DLYRST	Input	1	Reset IDELAY to pre-programmed value. If no value programmed, reset to 0. The DLYRST port is the same as the RST port in the IDELAY primitive. See "IDELAY Ports".
OCLK	Input	1	High-speed clock input for memory applications.
REV	Input	1	Reverse SR pin. Not available in the ISERDES block; connect to GND.
SHIFTIN1	Input	1	Carry input for data width expansion. Connect to SHIFTOUT1 of master IOB. See "ISERDES Width Expansion".
SHIFTIN2	Input	1	Carry input for data width expansion. Connect to SHIFTOUT2 of master IOB. See "ISERDES Width Expansion".
SR	Input	1	Active High reset. See "Reset Input – SR" in section "ISERDES Ports."

ISERDES Ports

Combinatorial Output – O

The combinatorial output port (O) is an unregistered output of the ISERDES module. This output can come directly from the data input (D), or from the data input (D) via the IDELAY block.

Registered Outputs – Q1 to Q6

The output ports Q1 to Q6 are the registered outputs of the ISERDES module. The outputs are synchronous to CLKDIV. The first bit clocked into the ISERDES are clocked out on Q*n*, where *n* is the width of the deserialization. One ISERDES block can support up to six bits (i.e., a 1:6 deserialization). Bit widths greater than 6 (up to 10) can be supported (see "ISERDES Width Expansion").

The bit ordering at the input of an OSERDES is the opposite of the bit ordering at the output of an ISERDES, as shown in Figure 8-3, page 369. For example, the least significant bit "A" of the word "FEDCBA" is placed at the D1 input of an OSERDES, but the same bit "A" emerges from the ISERDES at the Q6 output. In other words, D1 is the least significant input to the OSERDES, while Q6 is the least significant output of the ISERDES. When width expansion is used, D1 of the master OSERDES is the least significant input, while Q6 of the slave ISERDES is the least significant output.

Bitslip Operation – BITSLIP

The BITSLIP pin performs a Bitslip operation synchronous to CLKDIV when asserted (active High). Subsequently, the data seen on the Q1 to Q6 output ports shift, as in a barrel-shifter operation, one position every time Bitslip is invoked. The nature of the shift differs for SDR and DDR modes. See "BITSLIP Submodule" for more details.

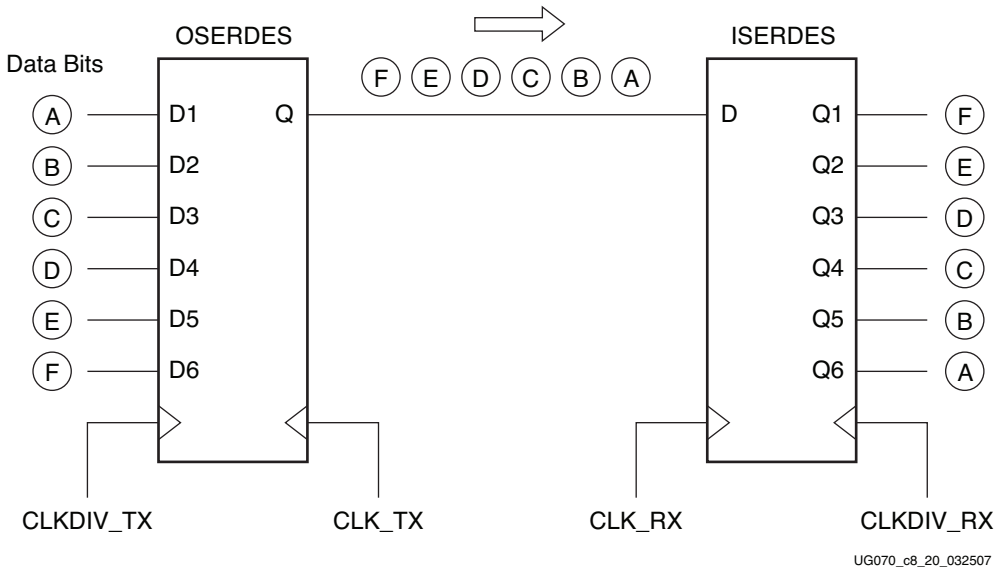


Figure 8-3: **Bit Ordering at Q1–Q6 Outputs of ISERDES**

Clock Enable Inputs – CE1 and CE2

Each ISERDES block contains an input clock enable module. Figure 8-4 shows the Input Clock Enable module.

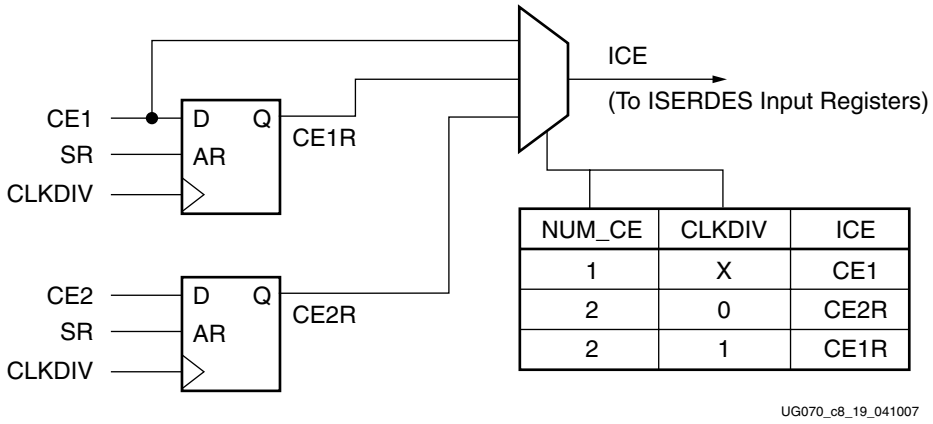


Figure 8-4: **Input Clock Enable Module**

When NUM_CE = 1, the CE2 input is not used, and the CE1 input is an active High clock enable connected directly to the input registers in the ISERDES.

When NUM_CE = 2, the CE1 and CE2 inputs are both used, with CE1 enabling the ISERDES for half of a CLKDIV cycle, and CE2 enabling the ISERDES for the other half.

The internal clock enable signal ICE shown in Figure 8-4 is derived from the CE1 and CE2 inputs. ICE drives the clock enable inputs of registers FF0, FF1, FF2, and FF3 shown in Figure 8-6, page 373. The remaining registers in Figure 8-6 do not have clock enable inputs.

High-Speed Clock Input – CLK

The high-speed clock input (CLK) is used to clock in the input serial data stream.

Divided Clock Input – CLKDIV

The divided clock input (CLKDIV) is typically a divided version of CLK (depending on the width of the implemented deserialization). It drives the output of the serial-to-parallel converter, the delay element, the Bitslip submodule, and the CE module.

Serial Input Data from IOB – D

The serial input data port (D) is the serial (high-speed) data input port of the ISERDES. This port works in conjunction with all the Virtex-4 FPGA I/O resources to accommodate the desired I/O standards.

High-Speed Clock for Strobe-Based Memory Interfaces – OCLK

The OCLK clock input is used to transfer strobe-based memory data onto a free-running clock domain. OCLK is a free-running FPGA clock at the same frequency as the strobe on the CLK input. The domain transfer from CLK to OCLK is shown in the block diagram of Figure 8-6. The timing of the domain transfer must be set by the user by adjusting the delay of the strobe signal to the CLK input (e.g., using IDELAY). Examples of setting the timing of this domain transfer are given in several memory-related application notes, including XAPP721 (available on www.xilinx.com). When INTERFACE_TYPE is NETWORKING, this port is unused and should be grounded.

Reset Input – SR

The reset input causes the outputs of all data flip-flops in the CLK and CLKDIV domains to be driven LOW asynchronously. For circuits in the ISERDES running on the CLK domain where timing is critical, there is an internal, dedicated circuit to re-time the SR input to produce a reset signal synchronous to the CLK domain. Similarly, there is also a dedicated circuit to re-time the SR input to produce a reset signal synchronous to the CLKDIV domain. Because there are circuits in the ISERDES that re-time the SR input, the user is only required to provide a reset pulse to the SR input that meets timing on the CLKDIV frequency domain. Therefore, SR should be driven High for a minimum of one CLKDIV cycle.

When building an interface consisting of multiple ISERDES, it may be important that all ISERDES in the interface are synchronized to one another. The internal re-timing of the SR input guarantees that all ISERDES that receive the same reset pulse come out of reset in sync with one another. The reset timing of multiple ISERDES is shown in Figure 8-5, page 371.

Clock Event 1

A reset pulse is generated on the rising edge of CLKDIV. Because the pulse must take two different routes to get to ISERDES0 and ISERDES1, there are different propagation delays for both paths. The difference in propagation delay is emphasized in Figure 8-5. The path to ISERDES0 is very long and the path to ISERDES 1 is very short, such that each ISERDES receives the reset pulse in a different CLK cycle. The internal resets for both CLK and CLKDIV go into reset asynchronously when the SR input is asserted.

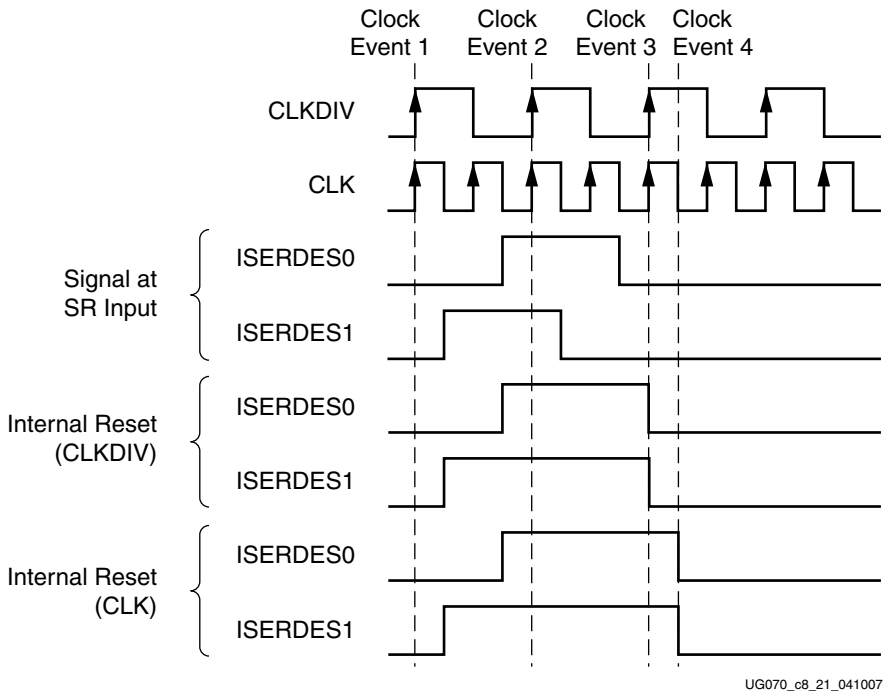


Figure 8-5: **Two ISERDES Coming Out of Reset Synchronously with One Another**

Clock Event 2

The reset pulse is deasserted on the rising edge of CLKDIV. The difference in propagation delay between the two ISERDES causes the SR input to come out of reset in two different CLK cycles. If there were no internal re-timing, ISERDES1 would come out of reset one CLK cycle before ISERDES0, which would leave both ISERDES out of sync.

Clock Event 3

The release of the reset signal at the SR input is re-timed internally to CLKDIV. This brings ISERDES 0 and 1 back into sync.

Clock Event 4

The release of the reset signal at the SR input is re-timed internally to CLK.

ISERDES Attributes

Table 8-2 summarizes all the applicable ISERDES attributes. A detailed description of each attribute follows the table. For more information on applying these attributes in UCF, VHDL, or Verilog code, refer to the Xilinx® ISE® Software Manual.

Table 8-2: **ISERDES Attributes**

Attribute Name	Description	Value	Default Value
BITSLIP_ENABLE	Allows the user to use the Bitslip submodule or bypass it.	Boolean: "TRUE" or "FALSE"	FALSE
DATA_RATE	Enables incoming data stream to be processed as SDR or DDR data.	String: "SDR" or "DDR"	DDR
DATA_WIDTH	Defines the width of the serial-to-parallel converter. The legal value depends on the DATA_RATE attribute (SDR or DDR).	Integer: 2, 3, 4, 5, 6, 7, 8, or 10. If DATA_RATE = DDR, value is limited to 4, 6, 8, or 10. If DATA_RATE = SDR, value is limited to 2, 3, 4, 5, 6, 7, or 8.	4
INTERFACE_TYPE	Chooses the ISERDES use model.	String: "MEMORY" or "NETWORKING"	MEMORY
IOBDELAY	Applies delay to combinatorial or registered paths, both, or neither.	String: "NONE", "IBUF", "IFD", or "BOTH"	NONE
IOBDELAY_TYPE	Sets the type of delay. See "Input Delay Element (IDELAY)".	String: "DEFAULT", "FIXED", or "VARIABLE"	DEFAULT
IOBDELAY_VALUE	Specifies the initial delay. See "Input Delay Element (IDELAY)".	Integer: 0 to 63	0
NUM_CE	Defines the number of clock enables.	Integer: 1 or 2	2
SERDES_MODE	Defines whether the ISERDES module is a master or slave when using width expansion.	String: "MASTER" or "SLAVE"	MASTER

BITSLIP_ENABLE Attribute

The BITSLIP_ENABLE attribute enables the Bitslip submodule. The possible values are TRUE and FALSE (default). BITSLIP_ENABLE must be set to TRUE when INTERFACE_TYPE is NETWORKING and FALSE when INTERFACE_TYPE is MEMORY. When set to TRUE, the Bitslip submodule responds to the BITSLIP signal. When set to FALSE, the Bitslip submodule is bypassed. See "BITSLIP Submodule".

DATA_RATE Attribute

The DATA_RATE attribute defines whether the incoming data stream is processed as single data rate (SDR) or double data rate (DDR). The allowed values for this attribute are SDR and DDR. The default value is DDR.

DATA_WIDTH Attribute

The DATA_WIDTH attribute defines the parallel data output width of the serial-to-parallel converter. The possible values for this attribute depend on the INTERFACE_TYPE and DATA_RATE attributes. See Table 8-3 for allowable data widths.

Table 8-3: **Allowable Data Widths**

INTERFACE_TYPE	DATA_RATE	Allowable Data Widths
NETWORKING	SDR	2, 3, 4, 5, 6, 7, 8
	DDR	4, 6, 8, 10
MEMORY	SDR	None
	DDR	4

When the DATA_WIDTH is set to widths larger than six, a pair of ISERDES must be configured into a master-slave configuration. See "ISERDES Width Expansion." Width expansion is not allowed in memory mode.

INTERFACE_TYPE Attribute

The INTERFACE_TYPE attribute determines whether the ISERDES is configured in memory or networking mode. The allowed values for this attribute are MEMORY or NETWORKING. The default mode is MEMORY. It is recommended to use the Memory Interface Generator (MIG) when using ISERDES in Memory mode.

When INTERFACE_TYPE is set to NETWORKING, the Bitslip submodule is available and the OCLK port is unused. Even if the Bitslip module is not used in networking mode, BITSLIP_ENABLE must be set to TRUE, and the Bitslip port can be tied Low to disable Bitslip operation. When set to MEMORY, the Bitslip submodule is not available (BITSLIP_ENABLE must be set to FALSE), and the OCLK port can be used.

Figure 8-6 illustrates the ISERDES internal connections when in Memory mode.

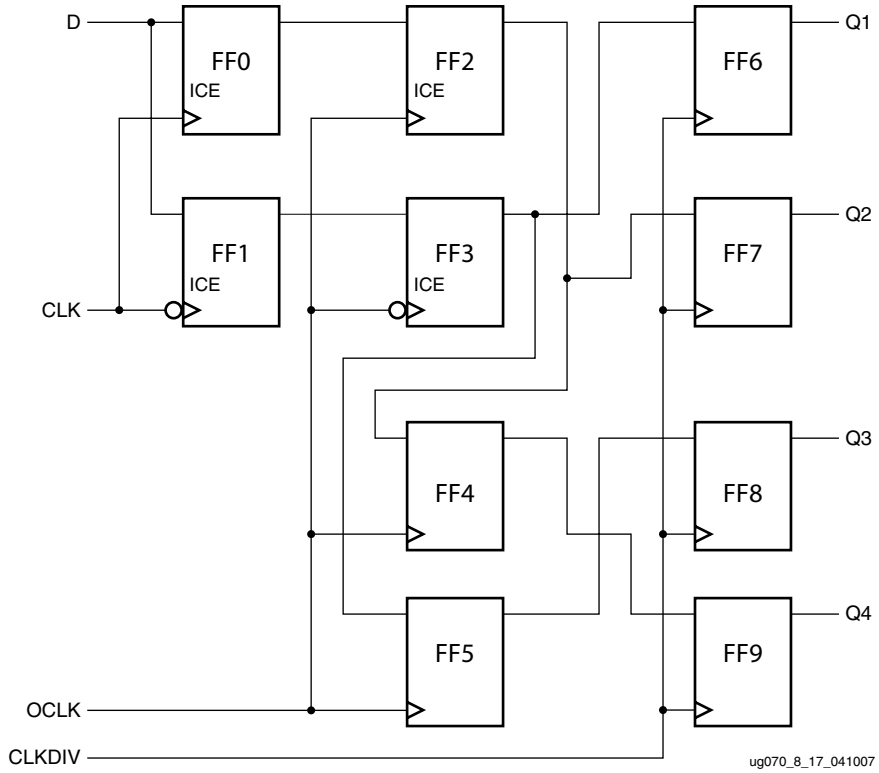


Figure 8-6: **Internal Connections of ISERDES When in Memory Mode**

IOBDELAY Attribute

The IOBDELAY attribute chooses the paths (combinatorial or registered) where the delay through the delay element is applied. The possible values for this attribute are NONE (default), IBUF, IFD, and BOTH. Table 8-4 summarizes the various output paths used for each attribute value.

Table 8-4: **IOBDELAY Attribute Value**

IOBDELAY Value	Delay Element Applied on Combinatorial Output Path (O)?	Delay Element Applied on Registered Output Path (Q1–Q6)?
NONE	No	No
IBUF	Yes	No
IFD	No	Yes
BOTH	Yes	Yes

NUM_CE Attribute

The NUM_CE attribute defines the number of clock enables (CE1 and CE2) used. The possible values are 1 and 2 (default = 1).

SERDES_MODE Attribute

The SERDES_MODE attribute defines whether the ISERDES module is a master or slave when using width expansion. The possible values are MASTER and SLAVE. The default value is MASTER. See "ISERDES Width Expansion."

ISERDES Clocking Methods

Networking Interface Type

The phase relationship of CLK and CLKDIV is important in the serial-to-parallel conversion process. Ideally, CLK and CLKDIV are phase-aligned. There is of course a tolerance around the ideal phase alignment. There are several clocking arrangements within the FPGA that are guaranteed by design to meet the phase relationship requirements of CLK and CLKDIV (shown below). These are the only valid clocking arrangements for the ISERDES.

- CLK driven by BUFIO, CLKDIV driven by BUFR
- CLK driven by DCM, CLKDIV driven by the CLKDV output of the same DCM
- CLK driven by PMCD, CLKDIV driven by CLKA1Dx of same PMCD

Memory Interface Type

- CLK driven by BUFIO or BUFG
- OCLK driven by DCM and CLKDIV driven by CLKDV output of same DCM
- OCLK driven by PMCD and CLKDIV driven by CLKA1Dx of same PMCD

The clocking arrangement using BUFIO and BUFR is shown in Figure 8-7. In the figure, it appears that BUFIO and BUFR are not phase-aligned at the inputs of the ISERDES. However, the hardware is slightly different from the software model. In hardware, BUFIO and BUFR are actually connected in parallel, such that the CLK and CLKDIV inputs of the ISERDES receive phase-aligned clocks. Connecting BUFIO and BUFR in HDL as shown in

Figure 8-7 results in the correct hardware connection (phase-aligned inputs to CLK and CLKDIV). No phase relationship between CLK and OCLK is expected. Calibration must be performed for reliable data transfer from CLK to OCLK domain. See section "High-Speed Clock for Strobe-Based Memory Interfaces – OCLK" for more information about transferring data between CLK and OCLK.

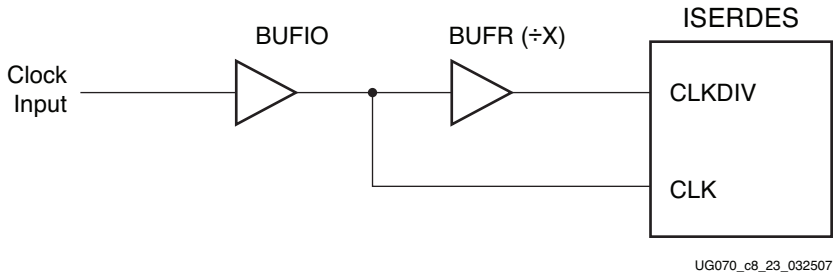


Figure 8-7: **Clocking Arrangement Using BUFIO and BUFR**

ISERDES Width Expansion

Two ISERDES modules are used to build a serial-to-parallel converter larger than 1:6. In every I/O tile (see "I/O Tile Overview" in Chapter 6) there are two ISERDES modules; one master and one slave. By connecting the SHIFTOUT ports of the master ISERDES to the SHIFTIN ports of the slave ISERDES the serial-to-parallel converter can be expanded to up to 1:10 (DDR) and 1:8 (SDR).

Figure 8-8 illustrates a block diagram of a 1:10 DDR serial-to-parallel converter using the master and slave ISERDES modules. Ports Q3–Q6 are used for the last four bits of the parallel interface on the slave ISERDES (LSB to MSB).

If the input is differential, the master ISERDES must be on the positive side of the differential input pair. If the input is not differential, the input buffer associated with the slave ISERDES is not available for use.

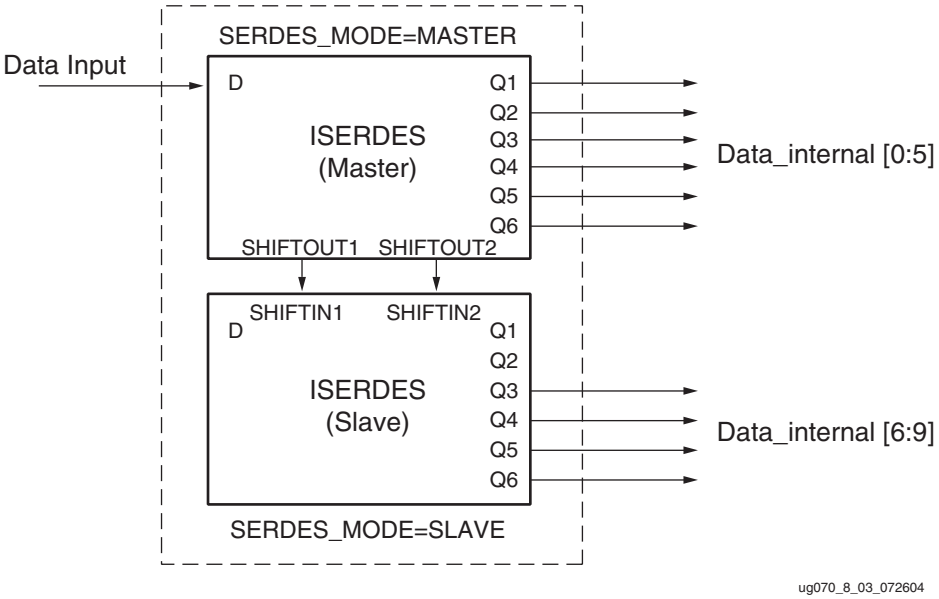


Figure 8-8: **Block Diagram of ISERDES Width Expansion**

Guidelines for Expanding the Serial-to-Parallel Converter Bit Width

1. Both ISERDES modules must be adjacent master and slave pairs.
2. Both ISERDES modules must be in NETWORKING mode (width expansion is not available in MEMORY mode).
3. Set the SERDES_MODE attribute for the master ISERDES to MASTER and the slave ISERDES to SLAVE (see "SERDES_MODE Attribute").
4. The user must connect the SHIFTIN ports of the SLAVE to the SHIFTOUT ports of the MASTER.
5. The SLAVE only uses the ports Q3 to Q6 as outputs.
6. DATA_WIDTH for Master and Slave must match.

Verilog Instantiation Template to use Width Expansion Feature

The following Verilog code uses the width expansion feature in DDR mode with a deserialization factor of 1:10.

```
//
// Module: serial_parallel_converter
//
// Description: Verilog instantiation template for
// a serial-to-parallel converter function using the
// ISERDES.
//
// Device: Virtex-4 Family
//////////////////////////////////////////////////////////////////////
`timescale 1ps/1ps

module serial_parallel_converter (
      Din,
        clk_in,
        rst,
       );

input  Din;
input  clk_in;
input  rst;

wire   iserdes_clkout;
wire   iobclk;
wire   clkdiv;
wire   shiftdata1;
wire   shiftdata2;
wire [9:0] data_internal;


// Instantiate ISERDES for forwarded clock
ISERDES fwd_clk (
                  .O(iserdes_clkout),
                  .Q1(),
                  .Q2(),
                  .Q3(),
                  .Q4(),
                  .Q5(),
                  .Q6(),
                  .SHIFTOUT1(),
```

```
                        .SHIFTOUT2(),
                        .BITSLIP(1'b0),
                        .CE1(1'b1),
                        .CE2(1'b1),
                        .CLK(iobclk),
                        .CLKDIV(clkdiv),
                        .D(clk_in),
                        .DLYCE(1'b0),
                        .DLYINC(1'b0),
                        .DLYRST(1'b0),
                        .OCLK(1'b0),
                        .REV(1'b0),
                        .SHIFTIN1(1'b0),
                        .SHIFTIN2(1'b0),
                        .SR(rst),
                        );
// synthesis BITSLIP_ENABLE of fwd_clk is  "TRUE";
// synthesis DATA_RATE of fwd_clk is  "DDR";
// synthesis DATA_WIDTH of fwd_clk is  4;
// synthesis INTERFACE_TYPE of fwd_clk is  "NETWORKING";
// synthesis IOBDELAY of fwd_clk is  "NONE";
// synthesis IOBDELAY_TYPE of fwd_clk is  "DEFAULT";
// synthesis IOBDELAY_VALUE of fwd_clk is 0;
// synthesis NUM_CE of fwd_clk is 1;
// synthesis SERDES_MODE of fwd_clk is  "MASTER";

// Instantiate Master ISERDES for data channel
// 1:10 Deserialization Factor
ISERDES data_chan_master (
                           .O(),
                           .Q1(data_internal[0]),
                           .Q2(data_internal[1]),
                           .Q3(data_internal[2),
                           .Q4(data_internal[3]),
                           .Q5(data_internal[4]),
                           .Q6(data_internal[5]),
                           .SHIFTOUT1(shiftdata1),
                           .SHIFTOUT2(shiftdata2),
                           .BITSLIP(1'b0),
                           .CE1(1'b1),
                           .CE2(1'b1),
                           .CLK(iobclk),
                           .CLKDIV(clkdiv),
                           .D(Din),
                           .DLYCE(1'b0),
                           .DLYINC(1'b0),
                           .DLYRST(1'b0),
                           .OCLK(1'b0),
                           .REV(1'b0),
                           .SHIFTIN1(1'b0),
                           .SHIFTIN2(1'b0),
                           .SR(rst),
                           );
// synthesis BITSLIP_ENABLE of data_chan_master is  "TRUE";
// synthesis DATA_RATE of data_chan_master is  "DDR";
// synthesis DATA_WIDTH of data_chan_master is  10;
// synthesis INTERFACE_TYPE of data_chan_master is  "NETWORKING";
// synthesis IOBDELAY of data_chan_master is  "NONE";
// synthesis IOBDELAY_TYPE of data_chan_master is  "DEFAULT";
```

```
// synthesis IOBDELAY_VALUE of data_chan_master is  0;
// synthesis NUM_CE of data_chan_master is  1;
// synthesis SERDES_MODE of data_chan_master is  "MASTER";
//
// Instantiate Slave ISERDES for data channel
// 1:10 Deserialization Factor
ISERDES data_chan_slave (
                         .O(),
                         .Q1(),
                         .Q2(),
                         .Q3(data_internal[6]),
                         .Q4(data_internal[7]),
                         .Q5(data_internal[8]),
                         .Q6(data_internal[9]),
                         .SHIFTOUT1(),
                         .SHIFTOUT2(),
                         .BITSLIP(1'b0),
                         .CE1(1'b1),
                         .CE2(1'b1),
                         .CLK(iobclk),
                         .CLKDIV(clkdiv),
                         .D(1'b0),
                         .DLYCE(1'b0),
                         .DLYINC(1'b0),
                         .DLYRST(1'b0),
                         .OCLK(1'b0),
                         .REV(1'b0),
                         .SHIFTIN1(shiftdata1),
                         .SHIFTIN2(shiftdata2),
                         .SR(rst),
                        );
// synthesis BITSLIP_ENABLE of data_chan_slave is  "TRUE";
// synthesis DATA_RATE of data_chan_slave is  "DDR";
// synthesis DATA_WIDTH of data_chan_slave is  10;
// synthesis INTERFACE_TYPE of data_chan_slave is  "NETWORKING";
// synthesis IOBDELAY of data_chan_slave is  "NONE";
// synthesis IOBDELAY_TYPE of data_chan_slave is  "DEFAULT";
// synthesis IOBDELAY_VALUE of data_chan_slave is  0;
// synthesis NUM_CE of data_chan_slave is  1;
// synthesis SERDES_MODE of data_chan_slave is  "SLAVE";
//
BUFIO bufio1 (
               .O(iobclk),
               .I(iserdes_clkout)
             );


// To get a 1:10 deserialization factor in DDR mode,
// set the clock divide factor to "5"
BUFR bufr1 (
            .O(clkdiv),
            .CE(1'b1),
            .CLR(1'b0),
            .I(iobclk)
            );
// synthesis BUFR_DIVIDE of bufr1 is  "5";

endmodule
```

ISERDES Latencies

When the ISERDES interface type is MEMORY, the latency through the OCLK stage is 1 CLKDIV cycle. However the total latency through the ISERDES depends on the phase relationship between the CLK and the OCLK clock inputs. When it is NETWORKING, the latency is 2 CLKDIV cycles. See Figure 8-12, page 385 and Figure 8-13, page 385 for a visualization of latency in networking mode. The extra CLKDIV cycle of latency in NETWORKING mode (compared to MEMORY mode) is due to the Bitslip submodule.

ISERDES Timing Model and Parameters

Table 8-5 describes the function and control signals of the ISERDES switching characteristics in the *Virtex-4 Data Sheet*.

Table 8-5: **ISERDES Switching Characteristics**

Symbol	Description
Setup/Hold for Control Lines	
$T_{ISCCK\_SR\_SYNC}$ / $T_{ISCKC\_SR\_SYNC}$	SR Pin setup/hold with respect to CLKDIV
$T_{ISCCK\_BITSLIP}$/ $T_{ISCKC\_BITSLIP}$	BITSLIP pin setup/hold with respect to CLKDIV
T_{ISCCK_CE} /T_{ISCKC_CE}	CE pin setup/hold with respect to CLK (for CE1)
T_{ISCCK_CE} /T_{ISCKC_CE}	CE pin setup/hold with respect to CLKDIV (for CE2)
T_{ISCCK_DLYCE} / T_{ISCKC_DLYCE}	DLYCE pin setup/hold with respect to CLKDIV
T_{ISCCK_DLYINC}/ T_{ISCKC_DLYINC}	DLYINC pin setup/hold with respect to CLKDIV
T_{ISCCK_DLYRST}/ T_{ISCKC_DLYRST}	DLYRST pin setup/hold with respect to CLKDIV
Setup/Hold for Data Lines	
T_{ISDCK_D} / T_{ISCKD_D}	D pin setup/hold with respect to CLK (IOBDELAY = IBUF or NONE)
	D pin setup/hold with respect to CLK (IOBDELAY = IFD or BOTH, IOBDELAY_TYPE = DEFAULT)
	D pin setup/hold with respect to CLK (IOBDELAY = IFD or BOTH, IOBDELAY_TYPE = FIXED, IOBDELAY_VALUE = 0)
T_{ISDCK_DDR} / T_{ISCKD_DDR}	D pin setup/hold with respect to CLK at DDR mode (IOBDELAY = IBUF or NONE)
	D pin setup/hold with respect to CLK at DDR mode (IOBDELAY = IFD or BOTH, IOBDELAY_TYPE = DEFAULT)
	D pin setup/hold with respect to CLK at DDR mode (IOBDELAY = IFD or BOTH, IOBDELAY_TYPE = FIXED, IOBDELAY_VALUE = 0)
Sequential Delay	
T_{ISCKO_Q}	CLKDIV to Out at Q pins

Timing Characteristics

In the timing diagrams of Figure 8-9, the timing parameter names change for different modes (SDR/DDR). However, the names do not change when a different bus input width, including when two ISERDES components are cascaded together to form 10 bits. In DDR mode, the data input (D) switches at every CLK edge (rising and falling).

Figure 8-9 illustrates an ISERDES timing diagram for the input data to the ISERDES in SDR mode.

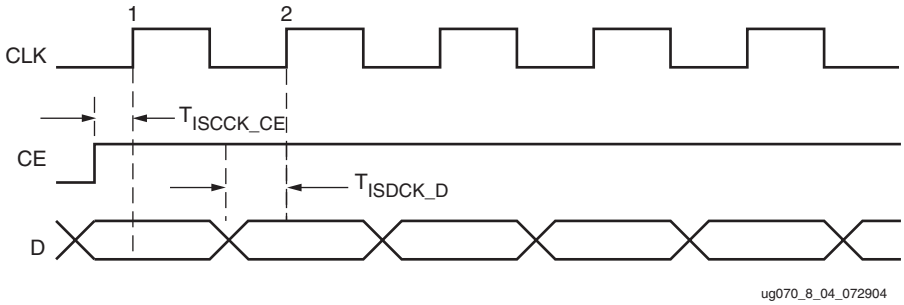


Figure 8-9: **ISERDES Input Data Timing Diagram**

Clock Event 1

- At time T_{ISCCK_CE}, before Clock Event 1, the clock enable signal becomes valid-High and the ISERDES can sample data.

Clock Event 2

- At time T_{ISDCK_D}, before Clock Event 2, the input data pin (D) becomes valid and is sampled at the next positive clock edge.

ISERDES VHDL and Verilog Instantiation Template

VHDL and Verilog instantiation templates are available in the Libraries Guide for all primitives and submodules.

In VHDL, each template has a component declaration section and an architecture section.

Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

ISERDES VHDL Instantiation

```
-- Module: ISERDES
-- Description: VHDL instantiation template
--
-- Device: Virtex-4 Family
-------------------------------------------------------------------------
-- Component Declaration for ISERDES should be placed
-- after architecture statement but before "begin" keyword

component ISERDES

generic (
              BITSLIP_ENABLE : string := "FALSE"; --(TRUE, FALSE)
              DATA_RATE : string := "DDR"; --(SDR, DDR)
              DATA_WIDTH : integer := 4; --(2,3,4,5,6,7,8,10)
```

```
               INTERFACE_TYPE : string := "MEMORY"; --(MEMORY,
NETWORKING)
               IOBDELAY : string := "NONE"; --(NONE,IBUF,IFD,BOTH)
               IOBDELAY_TYPE : string := "DEFAULT"; --(DEFAULT,FIXED,
VARIABLE)
               IOBDELAY_VALUE : integer := 0; --(0 to 63)
               NUM_CE : integer := 2; --(1,2)
               SERDES_MODE : string := "MASTER"; --(MASTER, SLAVE)
            );

port (
      O : out STD_LOGIC;
      Q1 : out STD_LOGIC;
      Q2 : out STD_LOGIC;
      Q3 : out STD_LOGIC;
      Q4 : out STD_LOGIC;
      Q5 : out STD_LOGIC;
      Q6 : out STD_LOGIC;
      SHIFTOUT1 : out STD_LOGIC;
      SHIFTOUT2 : out STD_LOGIC;
      BITSLIP : in STD_LOGIC;
      CE1 : in STD_LOGIC;
      CE2 : in STD_LOGIC;
      CLK : in STD_LOGIC;
      CLKDIV : in STD_LOGIC;
      D : in STD_LOGIC;
      DLYCE : in STD_LOGIC;
      DLYINC : in STD_LOGIC;
      DLYRST : in STD_LOGIC;
      OCLK : in STD_LOGIC;
      REV : in STD_LOGIC;
      SHIFTIN1 : in STD_LOGIC;
      SHIFTIN2 : in STD_LOGIC;
      SR : in STD_LOGIC;
     );
end component;

-- Component Attribute specification for ISERDES
-- should be placed after architecture declaration but
-- before the "begin" keyword

attribute BITSLIP_ENABLE : string;
attribute DATA_RATE : string;
attribute DATA_WIDTH : integer;
attribute INTERFACE_TYPE : string;
attribute IOBDELAY : string;
attribute IOBDELAY_TYPE : string;
attribute IOBDELAY_VALUE : integer;
attribute NUM_CE : integer;
attribute SERDES_MODE : string;

-- Component Instantiation for ISERDES should be placed
-- in architecture after the "begin" keyword
--
-- Instantiation Section
--
U1 : ISERDES

generic map (
```

```
                BITSLIP_ENABLE => "FALSE", --(TRUE, FALSE)
                DATA_RATE => "DDR", --(SDR, DDR)
                DATA_WIDTH => 4, --(2,3,4,5,6,7,8,10)
                INTERFACE_TYPE => "MEMORY", --(MEMORY, NETWORKING)
                IOBDELAY => "NONE", --(NONE,IBUF,IFD,BOTH)
                IOBDELAY_TYPE => "DEFAULT", --(DEFAULT, FIXED, VARIABLE)
                IOBDELAY_VALUE => 0, --(0 to 63)
                NUM_CE => 2, --(1,2)
                SERDES_MODE => "MASTER", --(MASTER, SLAVE)
                );

        port map (
                  O => data_output,
                  Q1 => Q(0),
                  Q2 => Q(1),
                  Q3 => Q(2),
                  Q4 => Q(3),
                  Q5 => open,
                  Q6 => open,
                  SHIFTOUT1 => open,
                  SHIFTOUT2 => open,
                  BITSLIP => bitslip,
                  CE1 => ce,
                  CE2 => open,
                  CLK => clk
                  CLKDIV => clkdiv
                  D => data_input
                  DLYCE => dlyce,
                  DLYINC => dlyinc,
                  DLYRST => dlyrst,
                  OCLK => open,
                  REV => open,
                  SHIFTIN1 => open,
                  SHIFTIN2 => open,
                  SR => rst,
                 );
```

ISERDES Verilog Instantiation

```
// Module: ISERDES
// Description: Verilog instantiation template
//
// Device: Virtex-4 Family
//-------------------------------------------------------------------------
// Instantiation Section
//
ISERDES U1
   (
     .O (data_output),
     .Q1 (Q[0]),
     .Q2 (Q[1]),
     .Q3 (Q[2]),
     .Q4 (Q[3]),
     .Q5 (open),
     .Q6 (open),
     .SHIFTOUT1 (open),
     .SHIFTOUT2 (open),
     .BITSLIP (bitslip),
     .CE1 (ce),
```

```
    .CE2 (open),
    .CLK (clk),
    .CLKDIV (clkdiv),
    .D (data_input),
    .DLYCE (dlyce),
    .DLYINC (dlyinc),
    .DLYRST (dlyrst),
    .OCLK (open),
    .REV (open),
    .SHIFTIN1 (open),
    .SHIFTIN2 (open),
    .SR (rst),
);
```

BITSLIP Submodule

All ISERDES blocks in Virtex-4 devices contain a Bitslip submodule. Bitslip shifts the parallel data in the ISERDES block, allowing every combination of a repeating serial pattern received by the deserializer to be presented to the FPGA logic. This repeating serial pattern is typically called a training pattern (training patterns are supported by many networking and telecommunication standards).

Bitslip Operation

By asserting the Bitslip pin of the ISERDES block, the incoming serial data stream is reordered at the parallel side. This operation is repeated until the training pattern is seen. The tables in Figure 8-10 illustrate the effects of a Bitslip operation in SDR and DDR mode. For illustrative purposes the data width is eight. The Bitslip operation is synchronous to CLKDIV. In SDR mode, every Bitslip operation causes the output pattern to shift left by one. In DDR mode, every Bitslip operation causes the output pattern to alternate between a shift right by one and shift left by three. In this example, on the eighth Bitslip operation, the output pattern reverts to the initial pattern. This assumes that serial data is an eight bit repeating pattern.

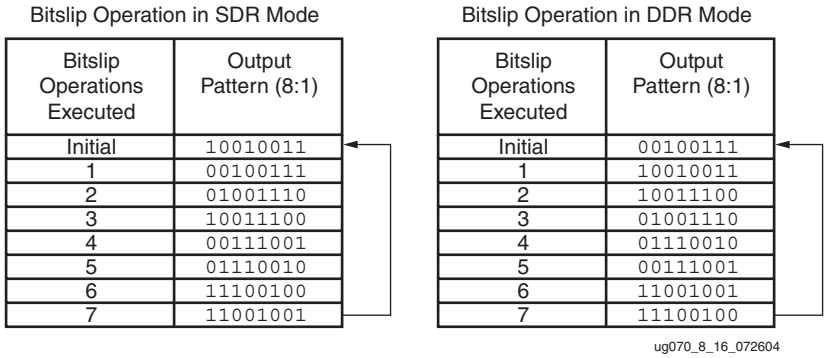


Figure 8-10: **Bitslip Operation Examples**

Figure 8-11 illustrates the ISERDES configured in 1:8 SDR mode with Bitslip_Enable set to TRUE. Two ISERDES modules are in a master-slave configuration for a data width of eight.

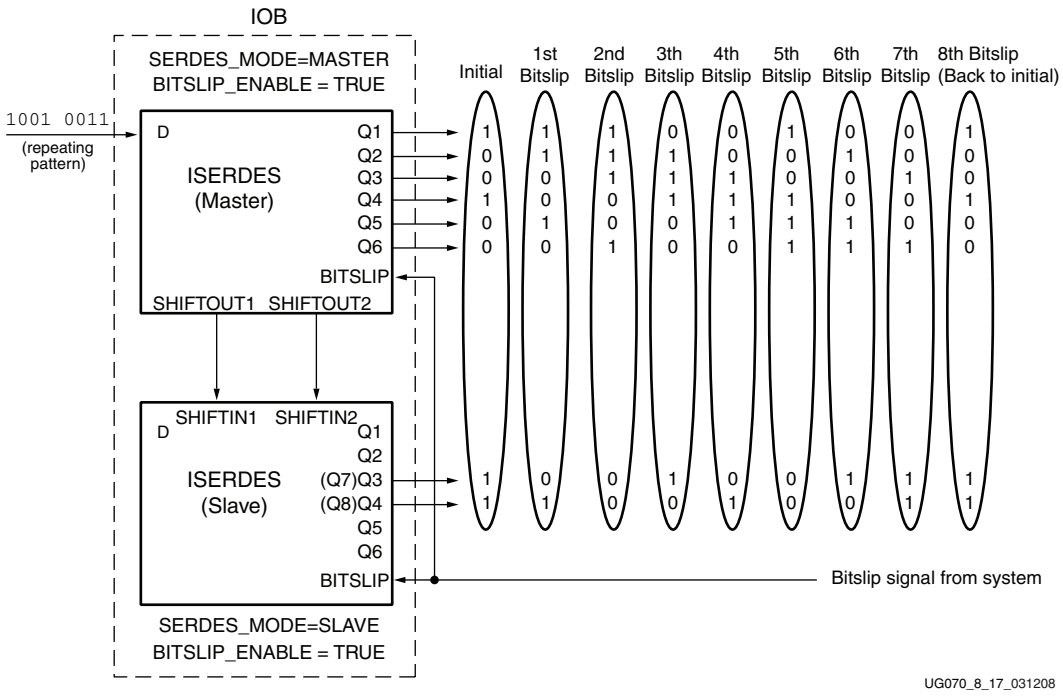


Figure 8-11: **Circuit Diagram for Bitslip Configuration in 1:8 SDR Mode**

Guidelines for Using the Bitslip Submodule

Set the BITSLIP_ENABLE attribute to TRUE. When BITSLIP_ENABLE is set to FALSE, the Bitslip pin has no effect. In a master-slave configuration, the BITSLIP_ENABLE attribute in both modules must be set to TRUE.

To invoke a Bitslip operation, the Bitslip port must be asserted High for one CLKDIV cycle. In SDR mode, Bitslip cannot be asserted for two consecutive CLKDIV cycles; Bitslip must be deasserted for at least one CLKDIV cycle between two Bitslip assertions. In both SDR and DDR mode, the total latency from when the ISERDES captures the asserted Bitslip input to when the "bit-slipped" ISERDES outputs Q1–Q6 are sampled into the FPGA logic by CLKDIV is two CLKDIV cycles.

Bitslip Timing Model and Parameters

This section discusses the timing models associated with the Bitslip controller in a 1:4 DDR configuration. Data (D) is a repeating, 4-bit training pattern ABCD. ABCD could appear at the parallel outputs Q1–Q4 of the ISERDES in four possible ways: *ABCD*, *BCDA*, *CDAB*, and *DABC*. Only one of these four alignments of the parallel word makes sense to the user's downstream logic that reads the data from the Q1–Q4 outputs of the ISERDES. In this case, it is assumed that *ABCD* is the word alignment that makes sense. Asserting Bitslip allows the user to see all possible configurations of ABCD and then choose the expected alignment (*ABCD*). Figure 8-12 shows the timing of two Bitslip operations and the corresponding re-alignments of the ISERDES parallel outputs Q1–Q4.

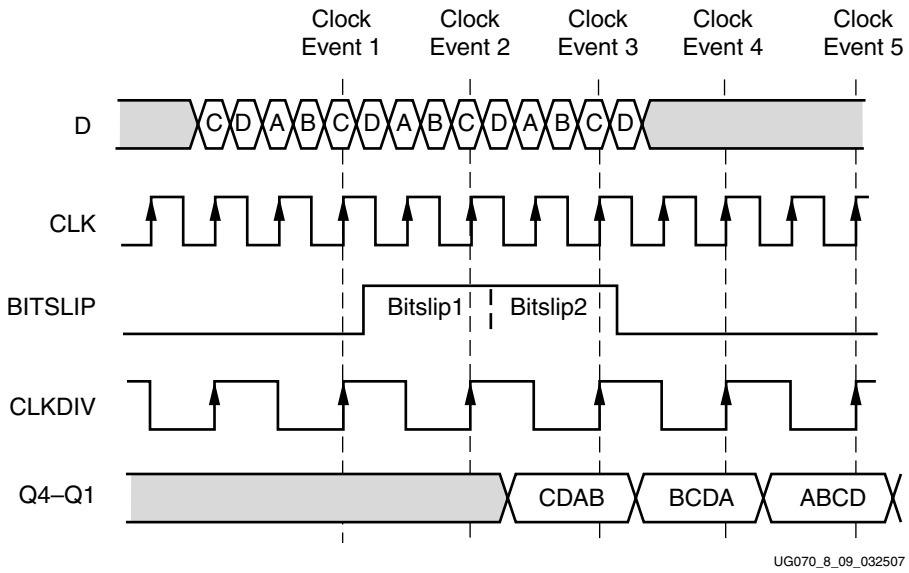


Figure 8-12: **Bitslip Timing Diagram**

Clock Event 1

The entire first word *CDAB* has been sampled into the input side registers of the ISERDES. The Bitslip pin is not asserted, so the word propagates through the ISERDES without any realignment.

Clock Event 2

The second word *CDAB* has been sampled into the input side registers of the ISERDES. The Bitslip pin is asserted, which causes the Bitslip controller to shift all bits internally by 1 bit to the right.

Clock Event 3

The third word *CDAB* has been sampled into the input side registers of the ISERDES. The Bitslip pin is asserted for a second time, which causes the Bitslip controller to shift all bits internally by three bits to the left.

On this same edge of CLKDIV, the first word sampled is presented to Q1–Q4 without any realignment. The actual bits from the input stream that appear at the Q1–Q4 outputs during this cycle are shown in *A* of Figure 8-13.

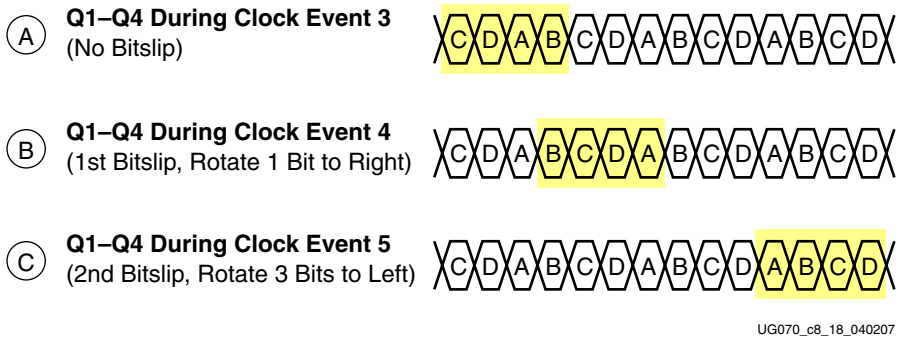


Figure 8-13: **Bits from Data Input Stream (D) of Figure 8-12**

Clock Event 4

The first two bits of the fourth word CD have been sampled into the input side registers of the ISERDES.

On this same edge of CLKDIV, the second word sampled is presented to Q1–Q4 with one bit shifted to the right. The actual bits from the input stream that appear at the Q1–Q4 outputs during this cycle are shown in *B* of Figure 8-13.

The realigned bits on Q1–Q4 are sampled into the FPGA fabric on the CLKDIV domain. The total latency from when the ISERDES captures the asserted Bitslip input to when the realigned ISERDES outputs Q1–Q4 are sampled by CLKDIV is 2 CLKDIV cycles.

Clock Event 5

The third word sampled is presented to Q1–Q4 with three bits shifted to the left. The actual bits from the input stream that appear at the Q1–Q4 outputs during this cycle are shown in *C* of Figure 8-13.

Output Parallel-to-Serial Logic Resources (OSERDES)

The Virtex-4 FPGA OSERDES is a dedicated parallel-to-serial converter with specific clocking and logic resources designed to facilitate the implementation of high-speed source-synchronous interfaces. Every OSERDES module includes a dedicated serializer for data and 3-state control. Both Data and 3-state serializers can be configured in SDR and DDR mode. Data serialization can be up to 6:1 (10:1 if using "OSERDES Width Expansion"). 3-state serialization can be up to 4:1.

Figure 8-14 shows a block diagram of the OSERDES, highlighting all the major components and features of the block.

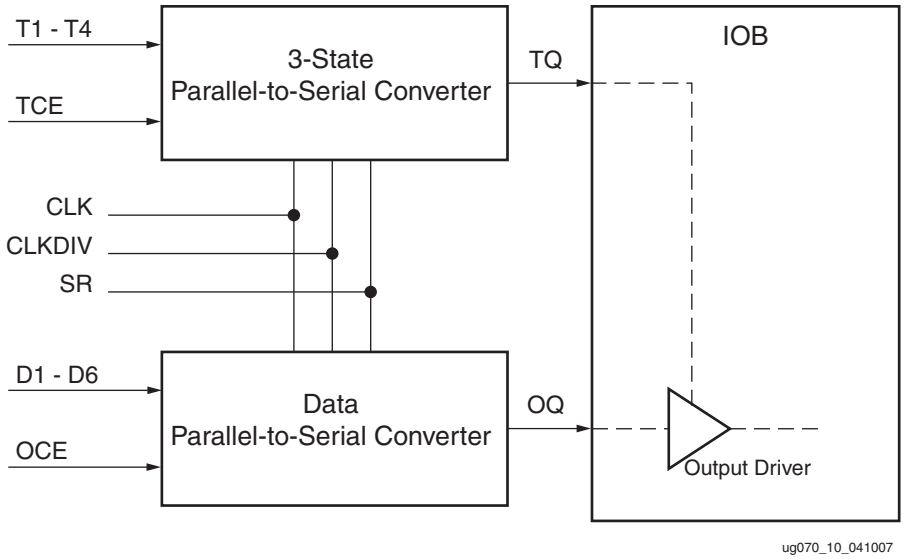


Figure 8-14: **OSERDES Block Diagram**

Data Parallel-to-Serial Converter

The data parallel-to-serial converter in one OSERDES blocks receives two to six bits of parallel data from the fabric (10:1 if using "OSERDES Width Expansion"), serializes the data, and presents it to the IOB via the OQ outputs. Parallel data is serialized from lowest

order data input pin to highest (i.e., data on the D1 input pin is the first bit transmitted at the OQ pins). The data parallel-to-serial converter is available in two modes; single-data rate (SDR) and double-data rate (DDR).

The OSERDES uses two clocks, CLK and CLKDIV, for data rate conversion. CLK is the high-speed serial clock, CLKDIV is the divided parallel clock. It is assumed that CLK and CLKDIV are phase aligned. It is required that a reset be applied to the OSERDES prior to use. The OSERDES contains an internal counter that controls dataflow, and failure to synchronize the reset with the CLKDIV results in unexpected output. Table 8-6 describes the relationship between CLK and CLKDIV in all modes.

Table 8-6: **CLK/CLKDIV Relationship of the Data Parallel-to-Serial Converter**

Input Data Width Output in SDR Mode	Input Data Width Output in DDR Mode	CLK	CLKDIV
2	4	2X	X
3	6	3X	X
4	8	4X	X
5	10	5X	X
6	–	6X	X
7	–	7X	X
8	–	8X	X

3-State Parallel-to-Serial Conversion

In addition to parallel-to-serial conversion of data, an OSERDES module also contains a parallel-to-serial converter for 3-state control of the IOB. Unlike data conversion, the 3-state converter can only serialize up to four bits of parallel 3-state signals. The 3-state converter cannot be cascaded.

OSERDES Primitive

The OSERDES primitive is shown in Figure 8-15.

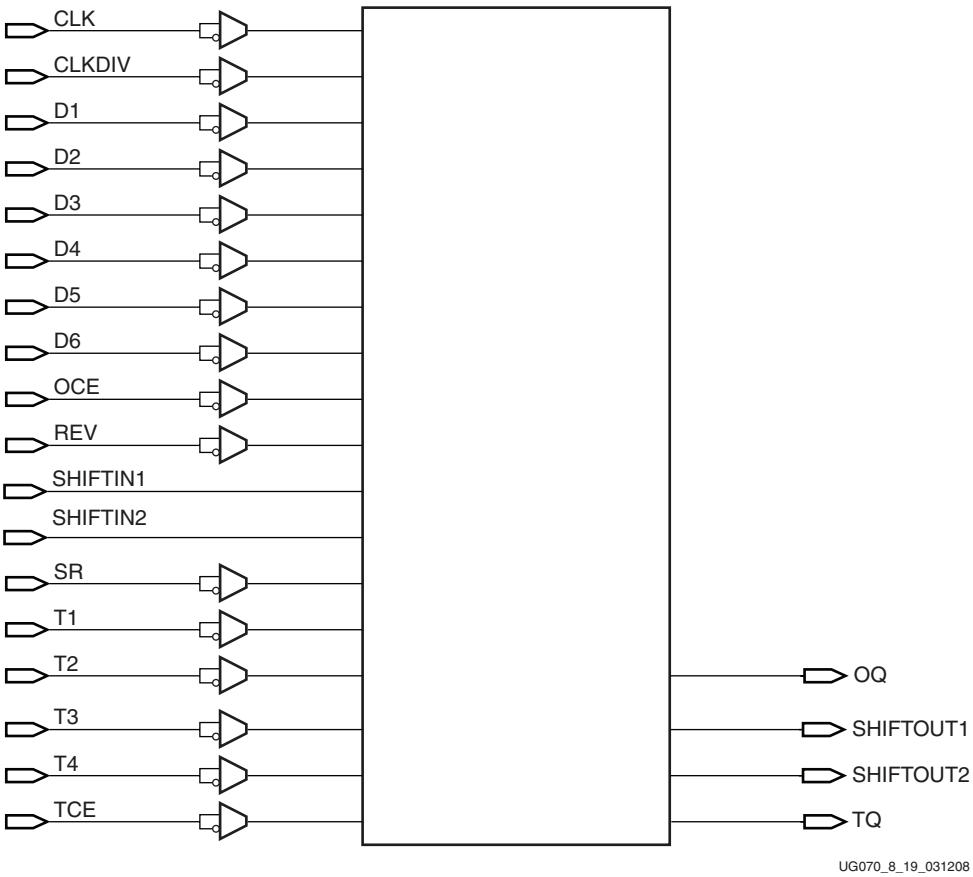


Figure 8-15: **OSERDES Primitive**

OSERDES Ports

Table 8-7 lists the available ports in the OSERDES primitive.

Table 8-7: **OSERDES Port List and Definitions**

Port Name	Type	Width	Description
OQ	Output	1	Data path output.
SHIFTOUT1	Output	1	Carry out for data width expansion. Connect to SHIFTIN1 of master OSERDES. See "OSERDES Width Expansion".
SHIFTOUT2	Output	1	Carry out for data width expansion. Connect to SHIFTIN2 of master OSERDES. See "OSERDES Width Expansion".
TQ	Output	1	3-state control output.
CLK	Input	1	High-speed clock input. Clocks serialized data to OQ output.
CLKDIV	Input	1	Divided clock input. Clocks parallel data at D1-D6 inputs into OSERDES.
D1 – D6	Input	1 (each)	Parallel data inputs.

Table 8-7: **OSERDES Port List and Definitions *(Continued)***

Port Name	Type	Width	Description
OCE	Input	1	Output data clock enable.
REV	Input	1	Reverse SR pin. Not available in the OSERDES block; connect to GND.
SHIFTIN1	Input	1	Carry input for data width expansion. Connect to SHIFTOUT1 of slave OSERDES. See "OSERDES Width Expansion".
SHIFTIN2	Input	1	Carry input for data width expansion. Connect to SHIFTOUT2 of slave OSERDES. See "OSERDES Width Expansion".
SR	Input	1	Active High reset.
T1 to T4	Input	1 (each)	Parallel 3-state inputs.
TCE	Input	1	3-state clock enable.

Data Path Output – OQ

The OQ port is the data output port of the OSERDES module. Data at the input port D1 appears first at OQ. This port connects the output of the data parallel-to-serial converter to the data input of the IOB.

3-state Control Output – TQ

This port is the 3-state control output of the OSERDES module. When used, this port connects the output of the 3-state parallel-to-serial converter to the control/3-state input of the IOB.

High-Speed Clock Input – CLK

This high-speed clock input drives the serial side of the parallel-to-serial converters.

Divided Clock Input – CLKDIV

This divided high-speed clock input drives the parallel side of the parallel-to-serial converters. This clock is the divided version of the clock connected to the CLK port.

Parallel Data Inputs – D1 to D6

All incoming parallel data enters the OSERDES module through ports D1 to D6. These ports are connected to the FPGA fabric, and can be configured from two to six bits (i.e., a 6:1 serialization). Bit widths greater than six (up to 10) can be supported by using a second OSERDES in SLAVE mode (see "OSERDES Width Expansion"). Refer to Figure 8-3, page 369 for bit ordering at the inputs and output of the OSERDES along with the corresponding bit order of the ISERDES.

Output Data Clock Enable – OCE

OCE is an active High clock enable for the data path.

Parallel 3-State Inputs – T1 to T4

All parallel 3-state signals enter the OSERDES module through ports T1 to T4. The ports are connected to the FPGA fabric, and can be configured as one, two, or four bits.

3-State Signal Clock Enable – TCE

TCE is an active High clock enable for the 3-state control path.

Reset Input – SR

The reset input causes the outputs of all data flip-flops in the CLK and CLKDIV domains to be driven LOW asynchronously. For circuits in the OSERDES running on the CLK domain where timing is critical, there is an internal, dedicated circuit to re-time the SR input to produce a reset signal synchronous to the CLK domain. Similarly, there is also a dedicated circuit to re-time the SR input to produce a reset signal synchronous to the CLKDIV domain. Because there are circuits in the OSERDES that re-time the SR input, the user is only required to provide a reset pulse to the SR input that meets timing on the CLKDIV frequency domain (synchronous to CLKDIV). Therefore, SR should be driven High for a minimum of one CLKDIV cycle.

When building an interface consisting of multiple OSERDES, it may be important that all OSERDES in the interface are synchronized to one another. The internal re-timing of the SR input guarantees that all OSERDES that receive the same reset pulse come out of reset in sync with one another. The reset timing of multiple OSERDES is shown in Figure 8-16.

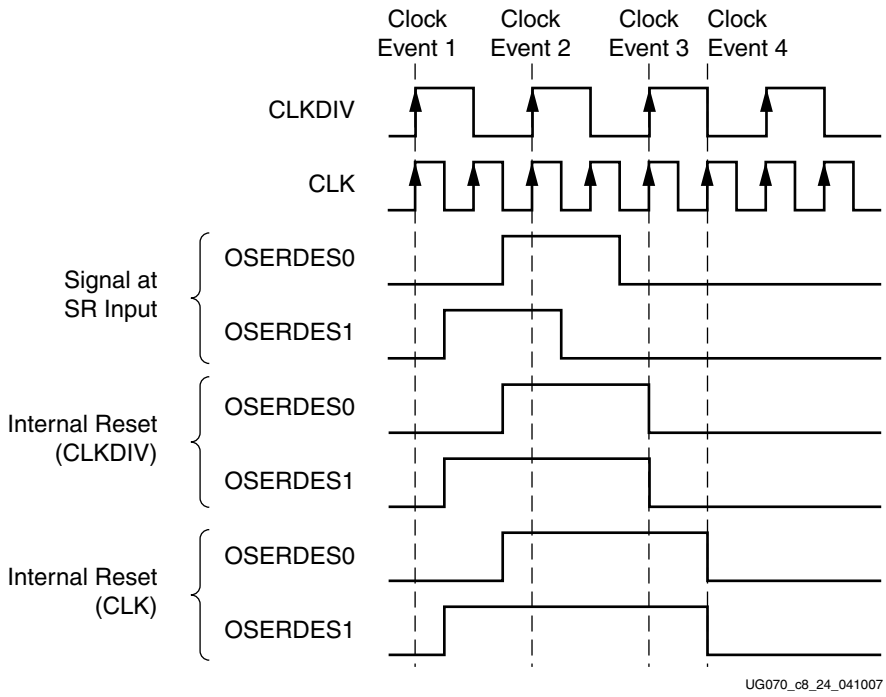


Figure 8-16: **Two OSERDES Coming Out of Reset Synchronously with One Another**

Clock Event 1

A reset pulse is generated on the rising edge of CLKDIV. Because the pulse must take two different routes to get to OSERDES0 and OSERDES1, there are different propagation delays for both paths. The difference in propagation delay is emphasized in Figure 8-16. The path to OSERDES0 is very long and the path to OSERDES1 is very short, such that each OSERDES receives the reset pulse in a different CLK cycle. The internal resets for both CLK and CLKDIV go into reset asynchronously when the SR input is asserted.

Clock Event 2

The reset pulse is deasserted on the rising edge of CLKDIV. The difference in propagation delay between the two OSERDES causes the SR input to come out of reset in two different CLK cycles. If there were no internal re-timing, OSERDES1 would come out of reset one CLK cycle before OSERDES0, which would leave both OSERDES out of sync.

Clock Event 3

The release of the reset signal at the SR input is re-timed internally to CLKDIV. This brings OSERDES0 and OSERDES1 back into sync.

Clock Event 4

The release of the reset signal at the SR input is re-timed internally to CLK.

OSERDES Attributes

Table 8-8 lists and describes the various attributes that are available for the OSERDES primitive. The table includes the default values.

Table 8-8: **OSERDES Attribute Summary**

OSERDES Attribute	Description	Value	Default Value
DATA_RATE_OQ	Defines whether data (OQ) changes at every clock edge or every positive clock edge with respect to CLK.	String: “SDR” or “DDR”	DDR
DATA_RATE_TQ	Defines whether the 3-state (TQ) changes at every clock edge, every positive clock edge with respect to clock, or is set to buffer configuration.	String: “BUF”, “SDR”, or “DDR”	DDR
DATA_WIDTH	Defines the parallel-to-serial data converter width. This value also depends on the DATA_RATE_OQ value.	Integer: 2, 3, 4, 5, 6, 7, 8, or 10. If DATA_RATE_OQ = DDR, value is limited to 4, 6, 8, or 10. If DATA_RATE_OQ = SDR, value is limited to 2, 3, 4, 5, 6, 7, or 8.	4
SERDES_MODE	Defines whether the OSERDES module is a master or slave when using width expansion.	String: “MASTER” or “SLAVE”	MASTER
TRISTATE_WIDTH	Defines the parallel to serial 3-state converter width.	Integer: 1, 2, or 4 If DATA_RATE_TQ = DDR, value is limited to 2 and 4. If DATA_RATE_TQ = SDR or BUF, value is limited to 1.	4

DATA_RATE_OQ Attribute

The DATA_RATE_OQ attribute defines whether data is processed as single data rate (SDR) or double data rate (DDR). The allowed values for this attribute are BUF, SDR, and DDR. The default value is DDR. When this attribute is set to BUF, the path from the T1 input to the TQ output of the OSERDES is completely combinatorial.

DATA_RATE_TQ Attribute

The DATA_RATE_TQ attribute defines whether 3-state control is to be processed as single data rate (SDR) or double data rate (DDR). The allowed values for this attribute are SDR and DDR. The default value is DDR.

DATA_WIDTH Attribute

The DATA_WIDTH attribute defines the parallel data input width of the parallel-to-serial converter. The possible values for this attribute depend on the DATA_RATE_OQ attribute. When DATA_RATE_OQ is set to SDR, the possible values for the DATA_WIDTH attribute are 2, 3, 4, 5, 6, 7, and 8. When DATA_RATE_OQ is set to DDR, the possible values for the DATA_WIDTH attribute are 4, 6, 8, and 10.

When the DATA_WIDTH is set to widths larger than six, a pair of OSERDES must be configured into a master-slave configuration, and the DATA_WIDTH value of both OSERDES must be set to the desired width. For example, for a width of 8, both MASTER and SLAVE must have DATA_WIDTH set to 8. See "OSERDES Width Expansion".

SERDES_MODE Attribute

The SERDES_MODE attribute defines whether the OSERDES module is a master or slave when using width expansion. The possible values are MASTER and SLAVE. The default value is MASTER. See "OSERDES Width Expansion".

TRISTATE_WIDTH Attribute

The TRISTATE_WIDTH attribute defines the parallel 3-state input width of the 3-state control parallel-to-serial converter. The possible values for this attribute depend on the DATA_RATE_TQ attribute. When DATA_RATE_TQ is set to SDR or BUF, the TRISTATE_WIDTH attribute can only be set to 1. When DATA_RATE_TQ is set to DDR, the possible values for the TRISTATE_WIDTH attribute are 2 or 4.

TRISTATE_WIDTH cannot be set to widths larger than four. The TRISTATE_WIDTH and DATA_RATE_TQ attribute settings do not impose any limits on the DATA_RATE_OQ and DATA_WIDTH settings, except when TRISTATE_WIDTH is set to 4 and DATA_RATE_TQ is set to DDR. In this case, the only allowable DATA_RATE_OQ and DATA_WIDTH settings are:

DATA_RATE_OQ =DDR; DATA_WIDTH = 4
DATA_RATE_OQ = SDR; DATA_WIDTH = 2

OSERDES Width Expansion

Two OSERDES modules are used to build a parallel-to-serial converter larger than 6:1. In every I/O tile there are two OSERDES modules; one master and one slave. By connecting the SHIFTIN ports of the master OSERDES to the SHIFTOUT ports of the slave OSERDES, the parallel-to-serial converter can be expanded to up to 10:1(DDR) and 8:1 (SDR).

If the output is differential, the master OSERDES must be on the positive side of the differential output pair. If the output is not differential, the output buffer associated with the slave OSERDES is not available for use.

Complementary single-ended standards (e.g., DIFF_HSTL, DIFF_SSTL) cannot be used when using the OSERDES with width expansion. This is because the complementary single-ended standards use both OLOGIC blocks in an I/O tile to transmit both legs of the signal, leaving no OLOGIC blocks to use for width expansion.

Figure 8-17 illustrates a block diagram of a 10:1 DDR parallel-to-serial converter using the master and slave OSERDES modules. Ports D3–D6 are used for the last four bits of the parallel interface on the slave OSERDES (LSB to MSB).

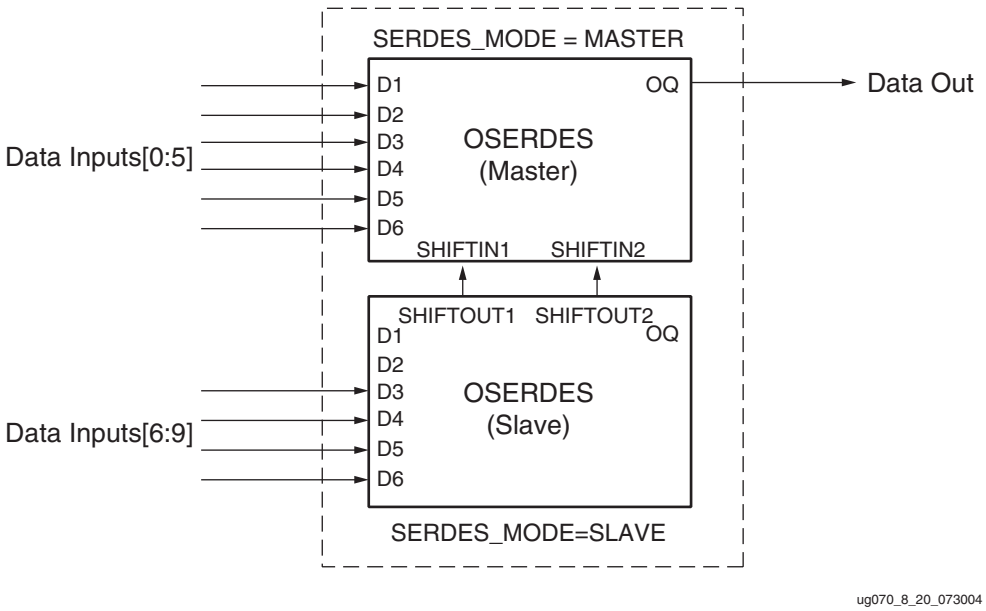


Figure 8-17: **Block Diagram of OSERDES Width Expansion**

Table 8-9 lists the data width availability for SDR and DDR mode.

Table 8-9: **OSERDES SDR/DDR Data Width Availability**

SDR Data Widths	2, 3, 4, 5, 6, 7, 8
DDR Data Widths	4, 6, 8, 10

Guidelines for Expanding the Parallel-to-Serial Converter Bit Width

1. Both the OSERDES modules must be adjacent master and slave pairs.
2. Set the SERDES_MODE attribute for the master OSERDES to MASTER and the slave OSERDES to SLAVE (see "SERDES_MODE Attribute").
3. The user must connect the SHIFTIN ports of the MASTER to the SHIFTOUT ports of the SLAVE.
4. The SLAVE only uses the ports D3 to D6 as an input.
5. Master and Slave have the same DATA_WIDTH attribute value.

Table 8-10 shows the slave inputs used for data widths requiring width expansion.

Table 8-10: **Slave Inputs Used for Data Width Expansion**

Data Width	Slave Inputs Used
7	D3
8	D3–D4
10	D3–D6

OSERDES Latencies

The input to output latencies of OSERDES blocks depend on the DATA_RATE and DATA_WIDTH attributes. Latency is defined as a period of time between the following two events: (a) when the rising edge of CLKDIV clocks the data at inputs D1–D6 into the OSERDES, and (b) when the first bit of the serial stream appears at OQ. Table 8-11 summarizes the various OSERDES latency values.

Table 8-11: **OSERDES Latencies**

DATA_RATE	DATA_WIDTH	Latency
SDR	2:1	1 CLK cycle
	3:1	3 CLK cycles
	4:1	4 CLK cycles
	5:1	4 CLK cycles
	6:1	5 CLK cycles
	7:1	5 CLK cycles
	8:1	6 CLK cycles
DDR	4:1	1 CLK cycle
	6:1	3 CLK cycles
	8:1	4 CLK cycles
	10:1	4 CLK cycles

OSERDES Timing Model and Parameters

This section discusses all timing models associated with the OSERDES primitive. Table 8-12 describes the function and control signals of the OSERDES switching characteristics in the *Virtex-4 Data Sheet*.

Table 8-12: **OSERDES Switching Characteristics**

Symbol	Description
Setup/Hold	
T_{OSDCK_D}/T_{OSCKD_D}	D input setup/hold with respect to CLKDIV
T_{OSDCK_T}/T_{OSCKD_T}	T input setup/hold with respect to CLK
T_{OSDCK_T}/T_{OSCKD_T}	T input setup/hold with respect to CLKDIV
$T_{OSCCK\_OCE}/T_{OSCKC\_OCE}$	OCE input setup/hold with respect to CLK
$T_{OSCCK\_TCE}/T_{OSCKC\_TCE}$	TCE input setup/hold with respect to CLK
Sequential Delays	
T_{OSCKO_OQ}	Clock to Out from CLK to OQ
T_{OSCKO_TQ}	Clock to Out from CLK to TQ
Combinatorial	
T_{OSCO_OQ}	Asynchronous Reset to OQ
T_{OSCO_TQ}	Asynchronous Reset to TQ

Timing Characteristics of 2:1 SDR Serialization

In Figure 8-18, the timing of a 2:1 SDR data serialization is illustrated.

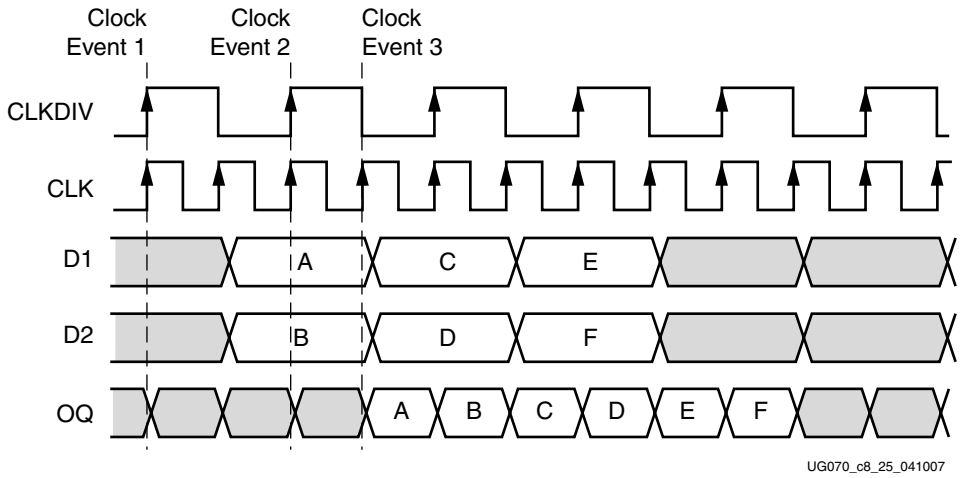


Figure 8-18: **OSERDES Data Flow and Latency in 2:1 SDR Mode**

Clock Event 1

On the rising edge of CLKDIV, the word *AB* is driven from the FPGA fabric to the D1 and D2 inputs of the OSERDES (after some propagation delay).

Clock Event 2

On the rising edge of CLKDIV, the word *AB* is sampled into the OSERDES from the D1 and D2 inputs.

Clock Event 3

The data bit A appears at OQ one CLK cycle after *AB* is sampled into the OSERDES. This latency is consistent with Table 8-11, which states that the latency of an OSERDES in 2:1 SDR mode is one CLK cycle.

Timing Characteristics of 8:1 DDR Serialization

In Figure 8-19, the timing of an 8:1 DDR data serialization is illustrated. In contrast to the 2:1 SDR example, a second OSERDES is required to achieve a serialization of 8:1. The two OSERDES are connected and configured using the methods of section "OSERDES Width Expansion," page 392. Six of the eight bits are connected to D1–D6 of the master OSERDES, while the remaining two bits are connected to D3–D4 of the slave OSERDES.

Clock Event 1

On the rising edge of CLKDIV, the word *ABCDEFGH* is driven from the FPGA fabric to the D1–D6 inputs of the master OSERDES and D3–D4 of the slave OSERDES (after some propagation delay).

Clock Event 2

On the rising edge of CLKDIV, the word *ABCDEFGH* is sampled into the master and slave OSERDES from the D1–D6 and D3–D4 inputs, respectively.

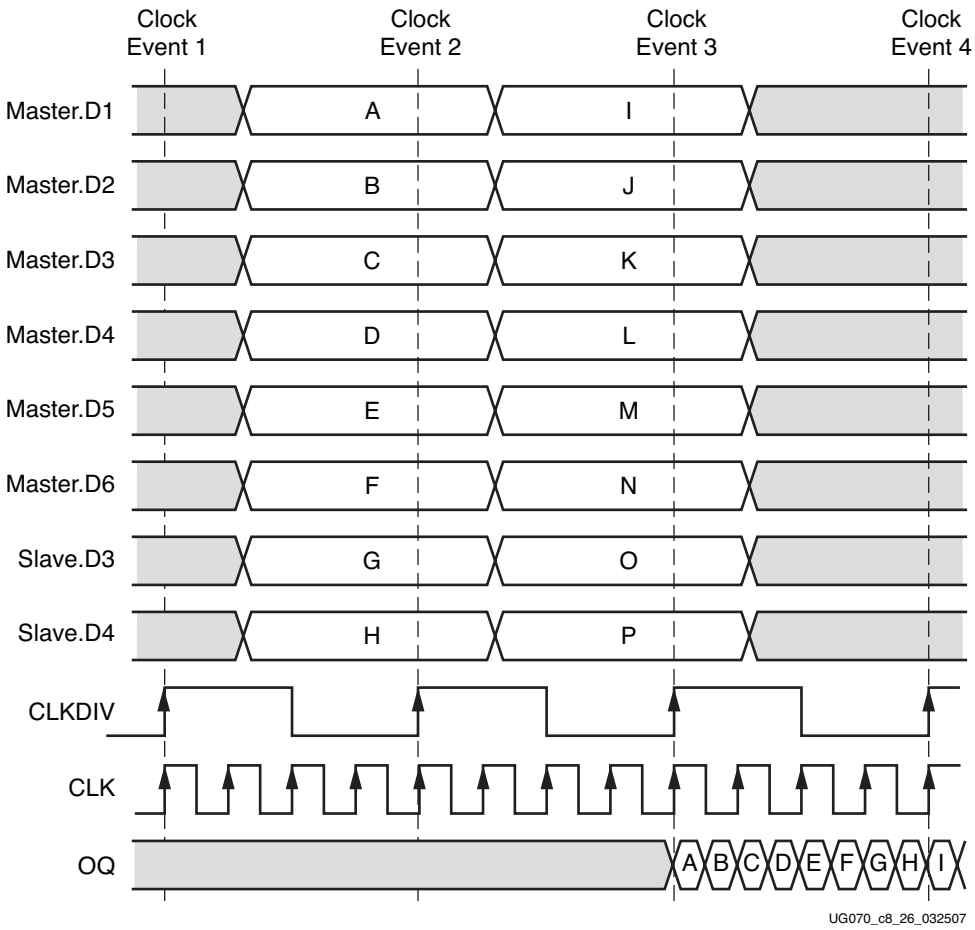


Figure 8-19: **OSERDES Data Flow and Latency in 8:1 DDR Mode**

Clock Event 3

The data bit *A* appears at OQ four CLK cycles after *ABCDEFGH* is sampled into the OSERDES. This latency is consistent withTable 8-11, which states that the latency of an OSERDES in 8:1 DDR mode is four CLK cycles.

The second word *IJKLMNOP* is sampled into the master and slave OSERDES from the D1–D6 and D3–D4 inputs, respectively.

Clock Event 4

Between Clock Events 3 and 4, the entire word *ABCDEFGH* is transmitted serially on OQ, a total of eight bits transmitted in one CLKDIV cycle.

The data bit *I* appears at OQ four CLK cycles after *IJKLMNOP* is sampled into the OSERDES. This latency is consistent with Table 8-11, which states that the latency of an OSERDES in 8:1 DDR mode is four CLK cycles.

Timing Characteristics of 4:1 DDR 3-State Controller Serialization

The operation of the 3-State Controller is illustrated in Figure 8-20. The example is a 4:1 DDR case shown in the context of a bidirectional system in which the IOB must be frequently 3-stated.

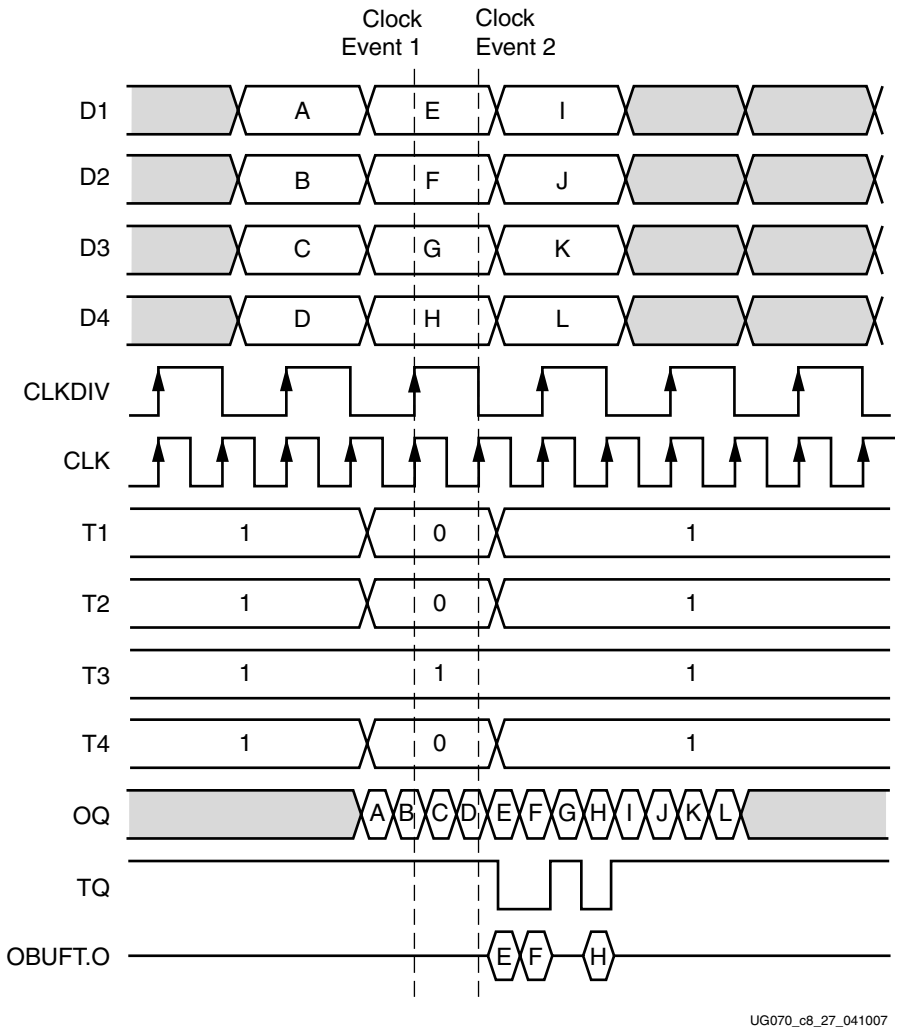


Figure 8-20: **3-State Control Serialization in 4:1 DDR Mode**

Clock Event 1

T1, T2, and T4 are driven Low to release the 3-state condition. The serialization paths of T1–T4 and D1–D4 in the OSERDES are identical (including latency), such that the bits *EFGH* are always aligned with the `0010` presented at the T1–T4 pins during Clock Event 1.

Clock Event 2

The data bit *E* appears at OQ one CLK cycle after *EFGH* is sampled into the OSERDES. This latency is consistent with Table 8-11, which states that the latency of an OSERDES in 4:1 DDR mode is one CLK cycle.

The 3-state bit 0 at T1 during Clock Event 1 appears at TQ one CLK cycle after `0010` is sampled into the OSERDES 3-state block. This latency is consistent with Table 8-11, which states that the latency of an OSERDES in 4:1 DDR mode is one CLK cycle.

OSERDES VHDL and Verilog Instantiation Templates

The following examples illustrate the instantiation of the OSERDES module in VHDL and Verilog.

OSERDES VHDL Template

```
--Example OSERDES Component Declaration

component OSERDES
  generic(
         DATA_RATE_OQ: string:= "DDR";
         DATA_RATE_TQ: string:= "DDR";
         DATA_WIDTH: integer:= 4;
         INIT_OQ: bit:= '0';
         INIT_TQ: bit:= '0';
         SERDES_MODE: string:= "MASTER";
         SRVAL_OQ: bit:= '0';
         SRVAL_TQ: bit:= '0';
         TRISTATE_WIDTH: integer:= 4
    );

  port(
      OQ: out std_ulogic;
      SHIFTOUT1: out std_ulogic;
      SHIFTOUT2: out std_ulogic;
      TQ: out std_ulogic;

      CLK: in std_ulogic;
      CLKDIV: in std_ulogic;
      D1: in std_ulogic;
      D2: in std_ulogic;
      D3: in std_ulogic;
      D4: in std_ulogic;
      D5: in std_ulogic;
      D6: in std_ulogic;
      OCE: in std_ulogic;
      REV        : in std_ulogic;
      SHIFTIN1: in std_ulogic;
      SHIFTIN2: in std_ulogic;
      SR         : in std_ulogic;
      T1: in std_ulogic;
      T2: in std_ulogic;
      T3: in std_ulogic;
      T4: in std_ulogic;
      TCE: in std_ulogic
    );
  end component;

--Example OSERDES instantiation

U_OSERDES : OSERDES
Port map (
      OQ => user_oq,
      SHIFTOUT1 => user_shiftout1,
      SHIFTOUT2 => user_shiftout2,
      TQ => user_tq,
```

```
    CLK => user_clk,
    CLKDIV => user_clkdiv,
    D1 => user_d1,
    D2 => user_d2,
    D3 => user_d3,
    D4 => user_d4,
    D5 => user_d5,
    D6 => user_d6,
    OCE => user_oce,
    REV => user_rev,
    SHIFTIN1 => user_shiftin1,
    SHIFTIN2 => user_shiftin2,
    SR => user_sr,
    T1 => user_t1,
    T2 => user_t2,
    T3 => user_t3,
    T4 => user_t4
    TCE => user_tce
  );
```

OSERDES Verilog Template

```
//Example OSERDES module declaration

module OSERDES (OQ, SHIFTOUT1, SHIFTOUT2, TQ, CLK, CLKDIV, D1, D2, D3,
D4, D5, D6, OCE, REV, SHIFTIN1, SHIFTIN2, SR, T1, T2, T3, T4, TCE);

    parameter DATA_RATE_OQ = "DDR";
    parameter DATA_RATE_TQ = "DDR";
    parameter DATA_WIDTH = 4;
    parameter INIT_OQ = 1'b0;
    parameter INIT_TQ = 1'b0;
    parameter SERDES_MODE = "MASTER";
    parameter SRVAL_OQ = 1'b0;
    parameter SRVAL_TQ = 1'b0;
    parameter TRISTATE_WIDTH = 4;

    output OQ;
    output SHIFTOUT1;
    output SHIFTOUT2;
    output TQ;

    input CLK;
    input CLKDIV;
    input D1;
    input D2;
    input D3;
    input D4;
    input D5;
    input D6;
    tri0 GSR = glbl.GSR;
    input OCE;
    input REV;
    input SHIFTIN1;
    input SHIFTIN2;
    input SR;
    input T1;
    input T2;
    input T3;
```

```
    input T4;
    input TCE;

endmodule;

//Example OSERDES instantiation

OSERDES U_OSERDES (
.OQ(user_oq),
.SHIFTOUT1(user_shiftout1),
.SHIFTOUT2(user_shiftout2),
.TQ(user_tq),
.CLK(user_clk),
.CLKDIV(user_clkdiv),
.D1(user_d1),
.D2(user_d2),
.D3(user_d3),
.D4(user_d4),
.D5(user_d5),
.D6(user_d6),
.OCE(user_oce),
.REV(user_rev),
.SHIFTIN1(user_shiftin1),
.SHIFTIN2(user_shiftin2),
.SR(user_sr),
.T1(user_t1),
.T2(user_t2),
.T3(user_t3),
.T4(user_t4),
.TCE(user_tce)
);
```

XILINX®

Chapter 9

Temperature Sensing Diode

Temperature-Sensing Diode (TDP/TDN)

The Virtex®-4 FPGA temperature-sensing diode is accessible through the TDP (anode) and TDN (cathode) pins. The TDP and TDN pins are wired internally to a diode-connected transistor, which creates a remote temperature sensor.

TDP and TDN are dedicated pins attached to the substrate/die and cannot be accessed through the software tools. TDP and TDN are always available, and no special design is necessary. The TDP and TDN pins are unconnected when this feature is not used.

The temperature-sensing diode is one part of a two-part system. A temperature sensor interface device is also required. Most temperature sensor interface devices provide corresponding pins to connect directly to the Virtex-4 FPGA TDP and TDN pins. Once the upper and lower temperature limits are set, an output signal is created when these bounds are exceeded. This output can be an interrupt to turn off the clock, turn on a fan, or perform another operation to reduce heat.

The accuracy of the temperature measurement achieved by this two-part system does not depend on the temperature-sensing diode (TDN/TDP pins). The voltage-versus-temperature curve is determined by the physical nature of the diode. Numerical readout accuracy relies on the temperature sensor interface device to translate the IV-versus-temperature curves into an actual temperature reading. The accuracy specifications are listed in the specific temperature sensor data sheets.

Temperature Sensor Examples

Maxim Remote/Local Temperature Sensors

General information on these devices is available from Maxim at:

http://www.maxim-ic.com.

Links to the specific data sheets for these devices:

- http://pdfserv.maxim-ic.com/ds/en/MAX1617.pdf
- http://www.maxim-ic.com/quick_view2.cfm/qv_pk/3000
- http://pdfserv.maxim-ic.com/en/ds/MAX6627-MAX6628.pdf

The *PC Board Layout* section of these data sheets include important design considerations.

Texas Instruments Remote/Local Temperature Sensor

The links below point to information from Texas Instruments on the Burr-Brown Products TMP401 temperature sensor:

- http://focus.ti.com/docs/prod/folders/print/tmp401.html (Overview)
- http://focus.ti.com/lit/ds/symlink/tmp401.pdf (Data Sheet)

Refer to the "Applications Information" section of the data sheet for important design considerations. For more information on this device, refer to http://www.ti.com.

National Semiconductor (LM83 or LM86)

These National Semiconductor devices are triple-diode input and local digital temperature sensors with a two-wire interface. General information on these devices is available at the National Semiconductor website: http://www.national.com.

Links to the specific data sheets for these devices:

- http://www.national.com/ds/LM/LM83.pdf
- http://www.national.com/ds/LM/LM86.pdf

The *Application Hints* section of these data sheets include important design considerations.

Index

H

I

L

M

N

O

P

R

S

T

W