XILINX®

Chapter 2

Design Considerations

Summary

This chapter covers the following topics:

- Using Global Clock Networks
- Using Digital Clock Managers (DCMs)
- Using Block SelectRAM™ Memory
- Using Distributed SelectRAM Memory
- Using Look-Up Tables as Shift Registers (SRLUTs)
- Designing Large Multiplexers
- Implementing Sum of Products (SOP) Logic
- Using Embedded Multipliers
- Using Single-Ended SelectI/O Resources
- Using Digitally Controlled Impedance (DCI)
- Using Double-Data-Rate (DDR) I/O
- Using LVDS I/O
- Using Bitstream Encryption
- Using the CORE Generator System

Introduction

This chapter describes how to take advantage of the many special features of the Virtex-II architecture to achieve maximum density and performance. In many cases, the functions described can be automatically generated using the Xilinx CORE Generator™ tool. This is noted throughout the chapter, in the following sections specifically:

- Using Block SelectRAM™ Memory
- Using Distributed SelectRAM Memory
- Using Look-Up Tables as Shift Registers (SRLUTs)
- Designing Large Multiplexers
- Using Embedded Multipliers

Using Global Clock Networks

Introduction

Virtex-II devices support very high frequency designs and thus require low-skew advanced clock distribution. With device density up to 10 million system gates, numerous global clocks are necessary in most designs. Therefore, to provide a uniform and portable solution (soft-IP), all Virtex-II devices from XC2V40 to XC2V8000 have 16 global clock buffers and support 16 global clock domains. Up to eight of these clocks can be used in any quadrant of the device by the synchronous logic elements (that is, registers, 18Kb block RAM, pipeline multipliers) and the IOBs. The software tools place and route these global clocks automatically.

If the design uses between 8 and 16 clocks, it must be partitioned into quadrants, with up to 8 clocks per quadrant. If more than 16 clocks are required, the backbone (24 horizontal and vertical long lines routing resources) can be used as additional clock network.

In addition to clock distribution, the 16 clock buffers are also “glitch-free” synchronous 2:1 multiplexers. These multiplexers are capable of switching between two asynchronous (or synchronous) clocks at any time. No particular phase relations between the two clocks are needed. The clock multiplexers can also be configured as a global clock buffer with a clock enable. The clock can be stopped High or Low at the clock buffer output.

Clock Distribution Resources

The various resources available to manage and distribute the clocks include:

- 16 clock pads that can be used as regular user I/Os if not used as clock inputs. The 16 clock pads can be configured for any I/O standard, including differential standards (for example, LVDS, LVPECL, and so forth).
- 16 “IBUFG” elements that represent the clock inputs in a VHDL or Verilog design.
- 8 “IBUFGDS” elements (that is, attributes LVPECL_33, LVDS_25, LVDS_33, LDT_25, or ULVDS_25) that represent the differential clock input pairs in a VHDL or Verilog design. Each IBUFGDS replaces two IBUFG elements.
- 4 to 12 Digital Clock Managers (DCMs), depending on the device size, to de-skew and generate the clocks. For more information on DCMs, see "Using Digital Clock Managers (DCMs)" on page 175.
- 16 “BUFGMUX” elements that can consist of up to 16 global clock buffers (BUFG), global clock buffers with a clock enable (BUFGCE), or global clock multiplexers (BUFGMUX).

Figure 2-1 illustrates the placement of these clock resources in Virtex-II devices (the XC2V250 through the XC2V2000) that have eight DCMs.

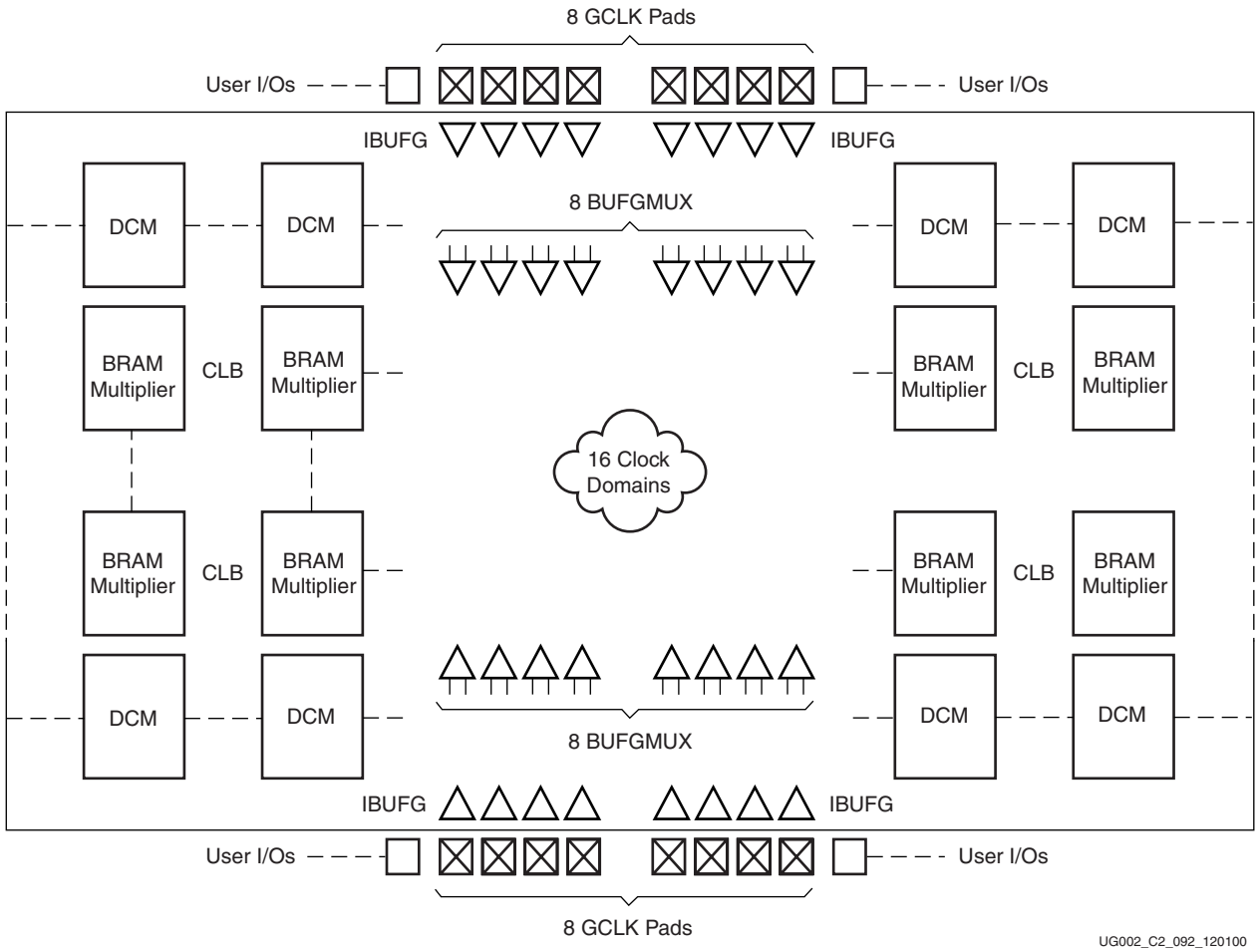


Figure 2-1: **Clock Resources in Virtex-II Devices**

The simple scheme to distribute an external clock in the device is to implement a clock pad with an IBUFG input buffer connected to a BUFG global buffer, as shown in Figure 2-2 and Figure 2-3. The primary (GCLKP) and secondary (GCLKS) clock pads have no relationship with the P-side and N-side of differential clock inputs. In banks 0 and 1, the GCLKP corresponds to the N-side, and the GCLKS corresponds to the P-side of a differential clock input. In banks 4 and 5, this correspondence is reversed.

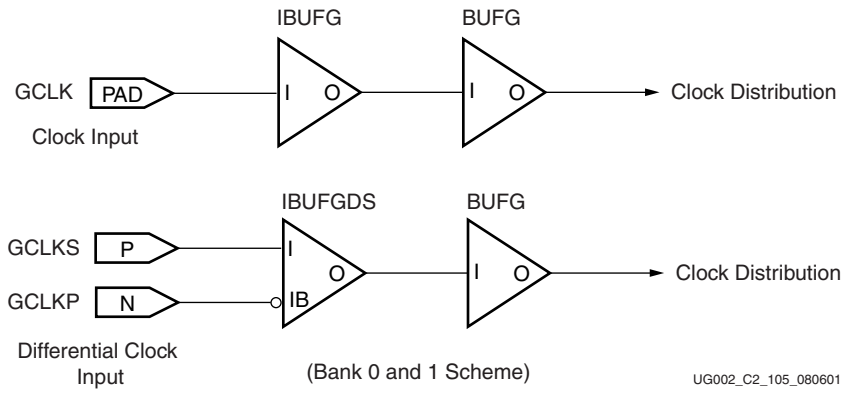


Figure 2-2: **Simple Clock Distribution (Bank 0 and 1 Scheme)**

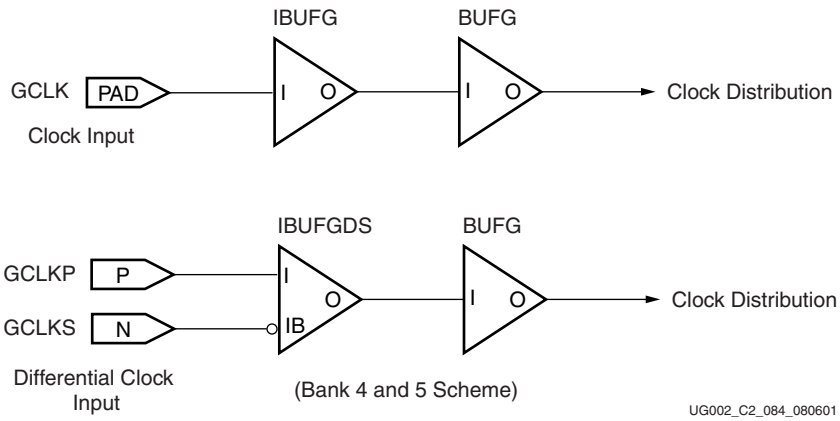


Figure 2-3: **Simple Clock Distribution (Bank 4 and 5 Scheme)**

Major synthesis tools automatically infer the IBUFG and BUFG when the corresponding input signal is used as a clock in the VHDL or Verilog code.

A high frequency or adapted (frequency, phase, and so forth) clock distribution with low skew is implemented by using a DCM between the output of the IBUFG and the input of the BUFG, as shown in Figure 2-4. "Using Digital Clock Managers (DCMs)" on page 175 provides details about DCMs and their use.

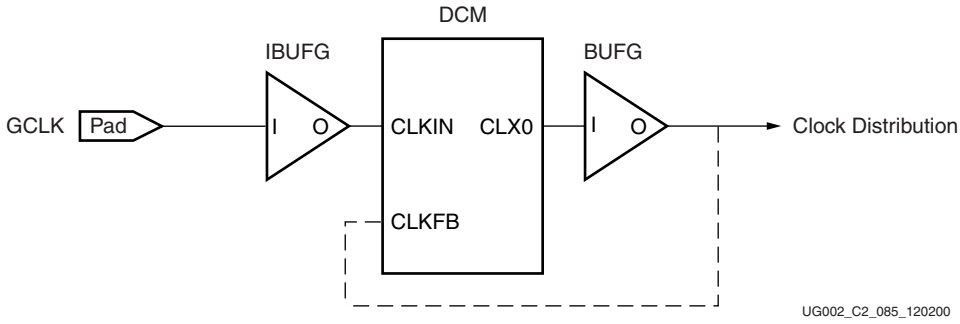


Figure 2-4: **Clock Distribution with DCM**

Clock distribution from internal sources is also possible with a BUFG only or with a DCM, as shown in Figure 2-5.

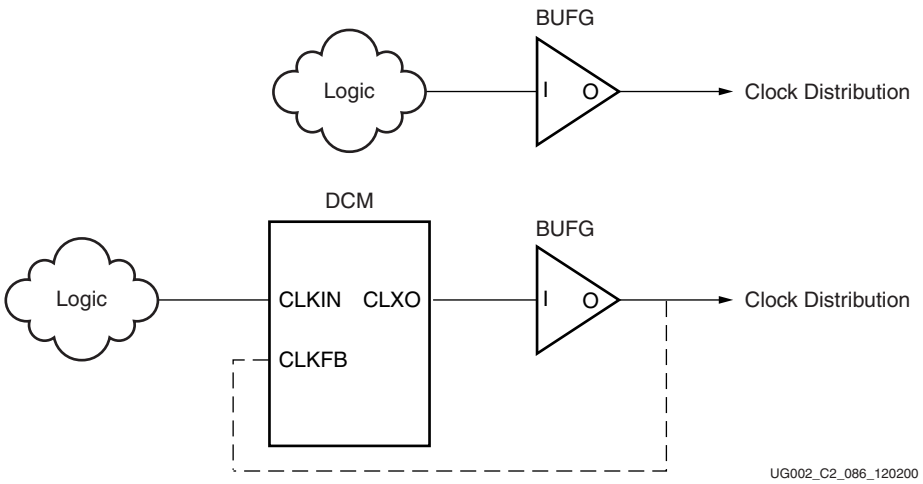


Figure 2-5: **Internal Logic Driving Clock Distribution**

Global Clock Inputs

The clock buffer inputs are fed either by one of the 16 clock pads (refer to the **Virtex-II Data Sheet**), by the outputs of the DCM, or by local interconnect. Each clock buffer can be a synchronous "glitch-free" 2:1 multiplexer with two clock inputs and one select input. Internal logic (or alternatively a regular IOB) can feed the clock inputs. Any internal or external signal can drive the select input or clock enable input.

The possible inputs driving a global clock buffer or multiplexer are summarized in Table 2-1.

Table 2-1: **Inputs Driving Global Clock Buffers or DCMs**

Source	Destination				
	BUFG(I) or BUFGCE(I)	BUFGCE (CE)	BUFGMUX (I0 or I1)	BUFGMUX (S)	DCM (CLKIN)
External Clock via IBUFG(O)	Dedicated in same quadrant[1]	NA	Dedicated in same quadrant[1]	NA	Same edge
DCM Clock Outputs	Same edge (top or bottom)[2]	NA	Same edge (top or bottom)[2]	NA	General interconnect[3]
Internal Logic	General interconnect	General interconnect	General interconnect	General interconnect	General interconnect[3]
User I/O Pad via IBUF(O) (not IBUFG)	General interconnect	General interconnect	General interconnect	General interconnect	General interconnect[3]
BUFG(O)	NA	NA	NA	NA	Global clock net
BUFGMUX(O)	NA	NA	General interconnect	NA	Global clock net

Notes:

1. Not all IBUFGs in the quadrant have a dedicated connection to a specific BUFG. Others would require general interconnect to be hooked up.
2. Same edge (top or bottom) enables use of dedicated routing resources.
3. Pad to DCM input skew is not compensated.

All BUFG (BUFGCE, BUFGMUX) outputs are available at the quadrant boundaries.

The output of the global clock buffer can be routed to non-clock pins.

Primary and Secondary Global Multiplexers

Each global clock buffer is a self-synchronizing circuit called a clock multiplexer.

The 16 global clock buffers or multiplexers are divided as follows:

- Eight primary clock multiplexers
- Eight secondary clock multiplexers

No hardware difference exists between a primary and a secondary clock multiplexer. However, some restrictions apply to primary/secondary multiplexers, because they share input connections, as well as access to a quadrant.

Each Virtex-II device is divided into four quadrants: North-West, South-West, North-East, and South-East. Each quadrant has two primary and two secondary clock multiplexers. The clock multiplexers are indexed 0 to 7, with one primary and one secondary for each index, alternating on the top and on the bottom (i.e., clock multiplexer "0P" at the bottom is facing clock multiplexer "0S" at the top).

In each device, the eight top/bottom clock multiplexers are divided into four primary and four secondary, indexed 0 to 7, as shown in Figure 2-6.

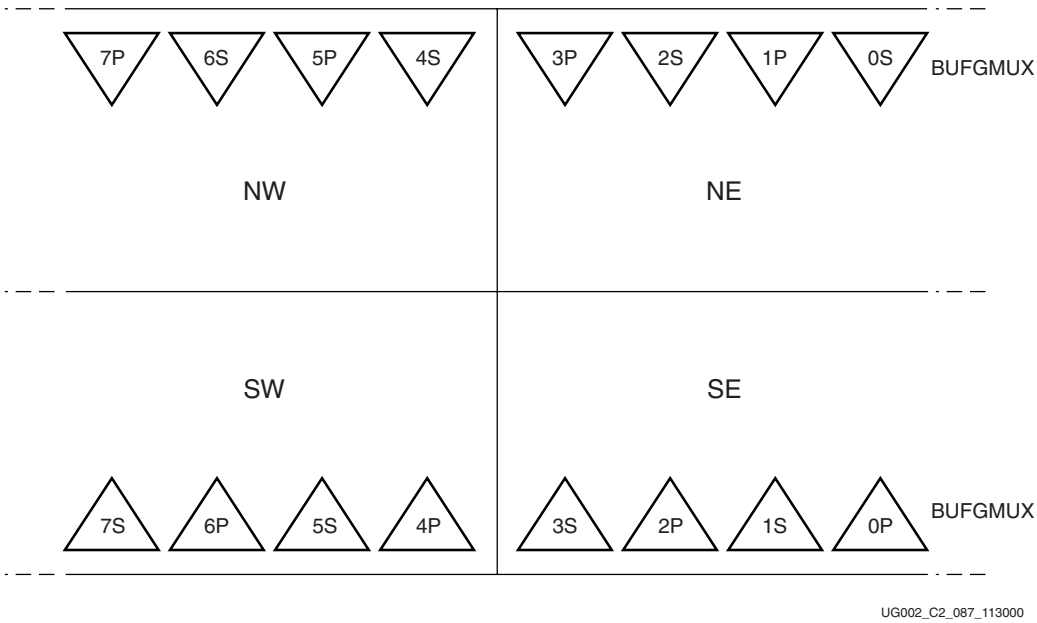


Figure 2-6: **Primary and Secondary Clock Multiplexer Locations**

Primary/Secondary: Rule 1

Considering two "facing" clock multiplexers (BUFG#P and BUFG#S), one or the other of these clock outputs can enter any quadrant of the chip to drive a clock within that quadrant, as shown in Figure 2-7. Note that the clock multiplexers "xP" and "xS" compete for quadrant access. For example, BUFG0P output cannot be used in the same quadrant as BUFG0S.

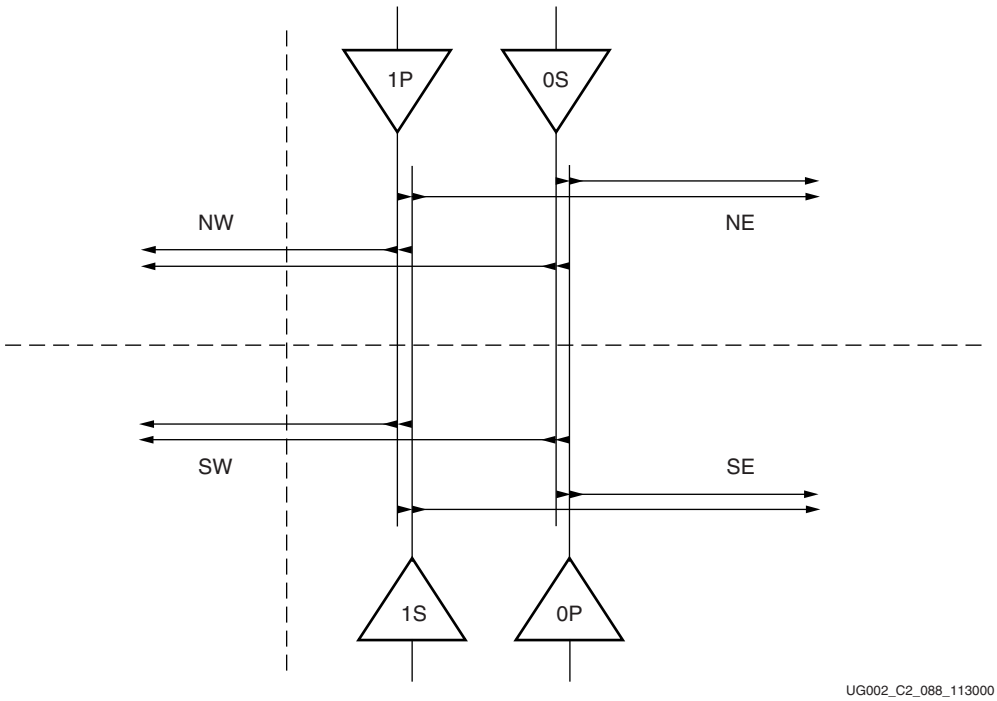


Figure 2-7: **Facing BUFG#P and BUFG#S Connections**

Primary/Secondary: Rule 2

In a BUFGCE or BUFGMUX configuration, shared inputs have to be considered. Any two adjacent clock multiplexers share two inputs, as shown in Figure 2-8. The clock multiplexer "1P" and "0S" have common I0/I1 and I1/I0 inputs.

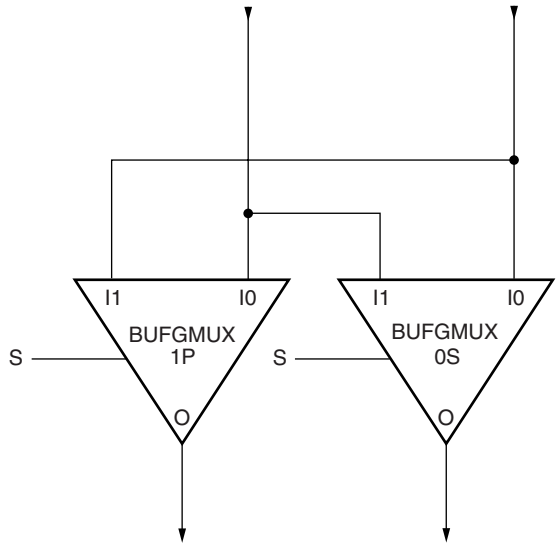


Figure 2-8: **Clock Multiplexer Pair Sharing Clock Multiplexer Inputs**

Table 2-2 lists the clock multiplexer pairs in any Virtex-II device. The primary multiplexer inputs I1/I0 are common with the corresponding secondary multiplexer inputs I0/I1 (i.e., Primary I1 input is common with secondary I0 input, and primary I0 input is common with secondary I1 input).

Table 2-2: **Top Clock Multiplexer Pairs**

Primary I1/I0	1P	3P	5P	7P
Secondary I0/I1	0S	2S	4S	6S

Table 2-3: **Bottom Clock Multiplexer Pairs**

Primary I1/I0	0P	2P	4P	6P
Secondary I0/I1	1S	3S	5S	7S

Primary/Secondary Usage

For up to eight global clocks, it is safe to use the eight primary global multiplexers (1P, 3P, 5P, 7P on the top and 0P, 2P, 4P, 6P on the bottom). Because of the shared inputs, a maximum of eight independent global clock multiplexers can be used in a design, as shown in Figure 2-9.

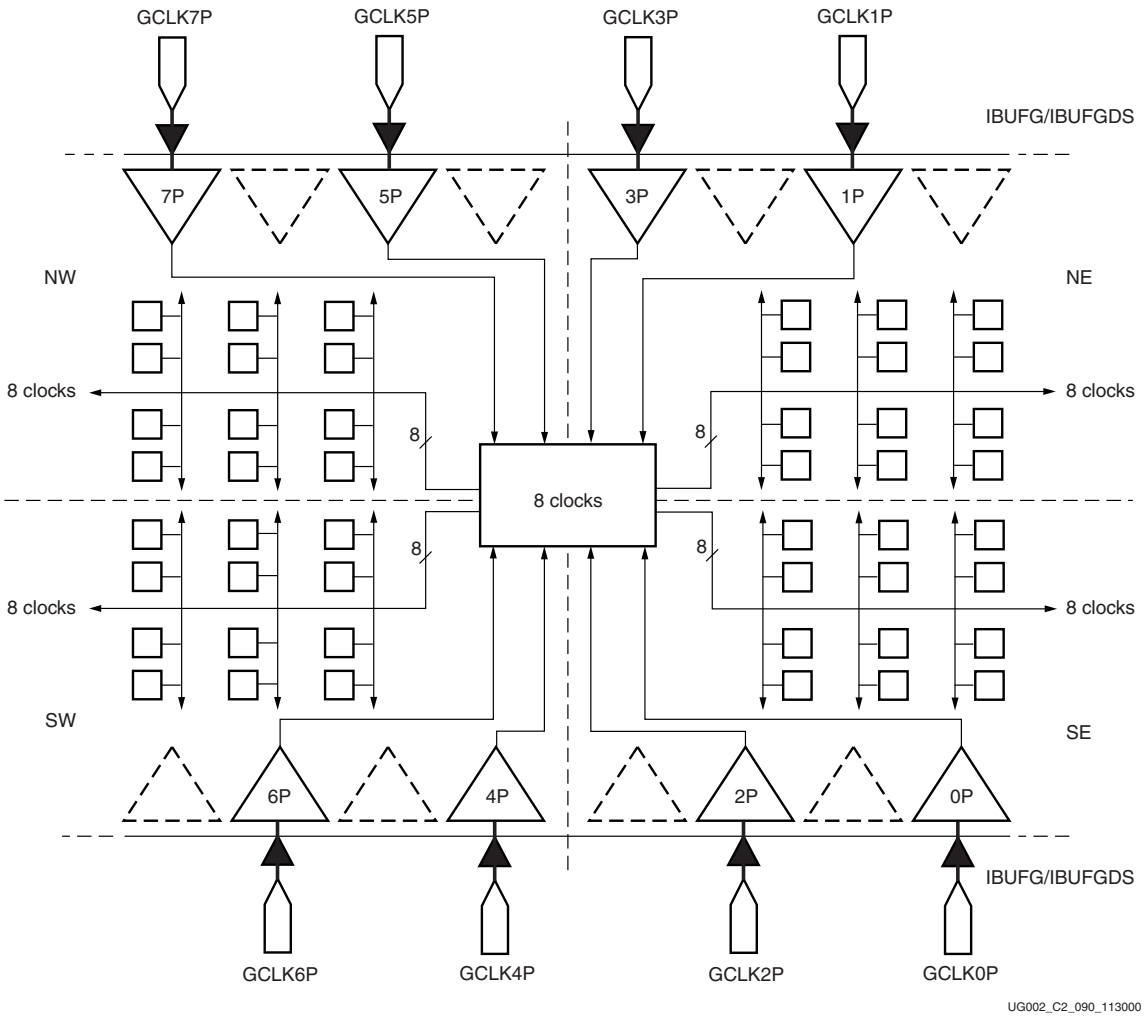


Figure 2-9: **Eight Global Clocks Design**

DCM Clocks

The four clock pins (IBUFG) in a quadrant can feed all DCMs in the same edge of the device. The clock-to-out and setup times are identical for all DCMs. Up to four clock outputs per DCM can be used to drive any clock multiplexer on the same edge (top or bottom), as shown in Figure 2-10.

BUFG Exclusivity

Each DCM has a restriction on the number of BUFGs it can drive on its (top or bottom) edge. Pairs of buffers with shared dedicated routing resources exist such that only one buffer from each dedicated pair can be driven by a single DCM. The exclusive pairs for each edge are: 1:5, 2:6, 3:7, and 4:8.

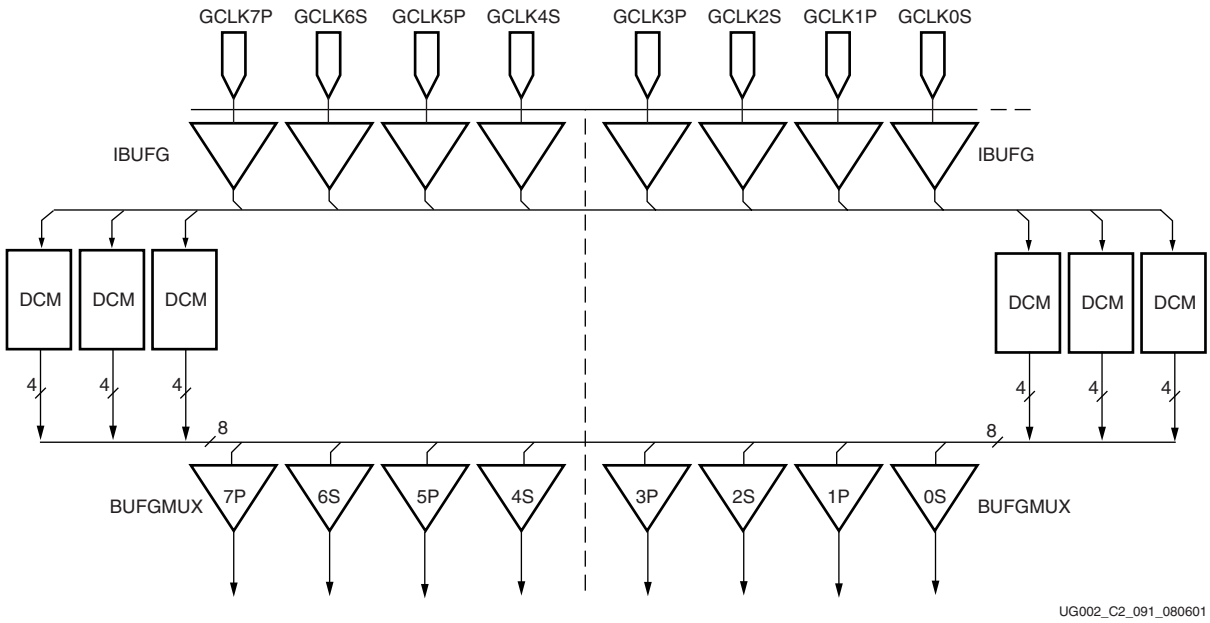


Figure 2-10: **DCM Clocks**

Clock Output

The clock distribution is based on eight clock trees per quadrant. Each clock multiplexer output is driving one global clock net. The Virtex-II device has eight dedicated low-skew clock nets. The device is divided into four quadrants (NW, NE, SW and SE) with eight global clocks available per quadrant.

Eight clock buffers are in the middle of the top edge and eight are in the middle of the bottom edge. Any of these 16 clock buffer outputs can be used in any quadrant, up to a maximum of eight clocks per quadrant, as illustrated in Figure 2-11, provided there is not a primary vs. secondary conflict.

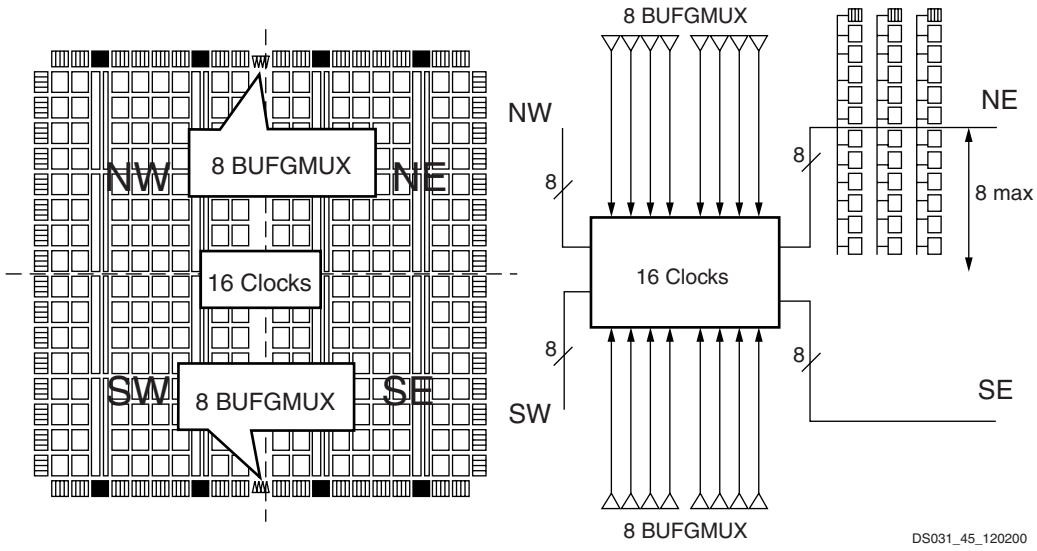


Figure 2-11: **Clock Buffer Outputs per Quadrant**

Designs with more than eight clocks must be floorplanned manually or automatically, distributing the clocks in each quadrant. As an example, a design with 16 clocks can be floorplanned as shown in Figure 2-12.

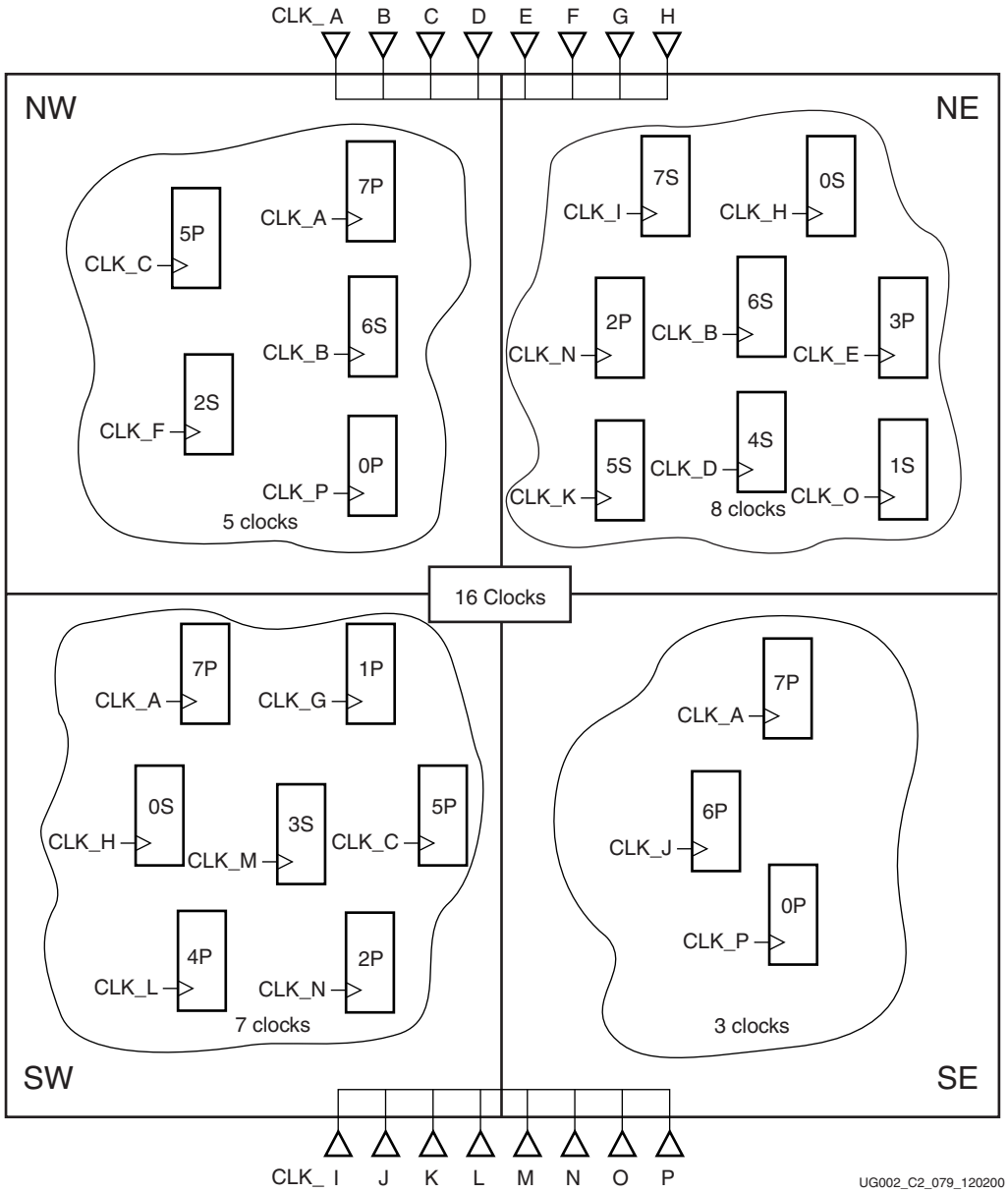


Figure 2-12: **16-Clock Floorplan**

The clock nets and clock buffers in this example are associated as shown in Table 2-4.

Table 2-4: **Clock Net Association With Clock Buffers**

Clock Net (top edge)	CLK_A	CLK_B	CLK_C	CLK_D	CLK_E	CLK_F	CLK_G	CLK_H
BUFG	7P	6S	5P	4S	3P	2S	1P	0S
Clock Net (bottom edge)	CLK_I	CLK_J	CLK_K	CLK_L	CLK_M	CLK_N	CLK_O	CLK_P
BUFG	7S	6P	5S	4P	3S	2P	1S	0P
Quadrant NW	CLK_A	CLK_B	CLK_C	–	–	CLK_F	–	CLK_P
Quadrant SW	CLK_A	–	CLK_C	CLK_L	CLK_M	CLK_N	CLK_G	CLK_H
Quadrant NE	CLK_I	CLK_B	CLK_K	CLK_D	CLK_E	CLK_N	CLK_O	CLK_H
Quadrant SE	CLK_A	CLK_J	–	–	–	–	–	CLK_P

CLK_A is used in three quadrants, and the other clocks are used in one or two quadrants, regardless of the position of the clock buffers (multiplexers), as long as they are not competing to access the same quadrant. (That is, CLK_A (BUFG7P) cannot be used in the same quadrant with CLK_I (BUFG7S). Refer to "Primary/Secondary: Rule 1" on page 160.) In other words, two buffers with the same index (0 to 7) cannot be used in the same quadrant.

Each register, block RAM, registered multiplier, or DDR register (IOB) can be connected to any of the eight clock nets available in a particular quadrant.

Note that if a global clock (primary buffer) is used in four quadrants, the corresponding secondary buffer is not available.

Power Consumption

Clock trees have been designed for low skew and low-power operation. Any unused branch is disconnected, as shown in Figure 2-13.

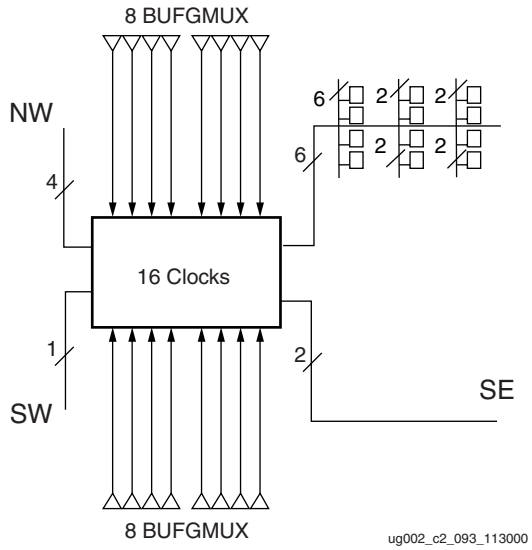


Figure 2-13: **Low-Power Clock Network**

Also available to reduce overall power consumption are the BUFGCE feature, for dynamically driving a clock tree only when the corresponding module is used, and the BUFGMUX feature, for switching from a high-frequency clock to a low-frequency clock.

The frequency synthesizer capability of the DCM can generate the low (or high) frequency clock from a single source clock, as illustrated in Figure 2-14. (See "Using Digital Clock Managers (DCMs)" on page 175).

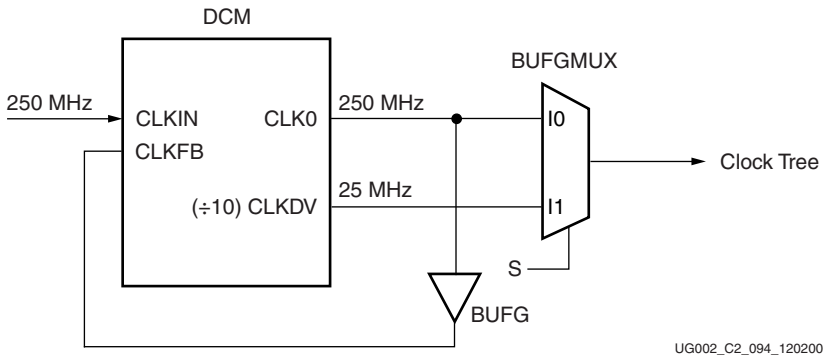


Figure 2-14: **Dynamic Power Reduction Scheme**

The primitives in Table 2-5 are available with the input, output, and control pins listed.

Table 2-5: **Clock Primitives**

Primitive	Input	Output	Control
IBUFG	I	O	–
IBUFGDS	I, IB	O	–
BUFG	I	O	–
BUFGMUX	I0, I1	O	S
BUFGMUX_1	I0, I1	O	S

Refer to "Using Single-Ended SelectI/O Resources" on page 258 for a list of the attributes available for IBUFG and Refer to "Using LVDS I/O" on page 317 for a list of the attributes available for IBUFGDS.

The submodules in Table 2-6 are available with the input, output, and control pins listed.

Table 2-6: **Clock Submodules**

Submodule	Input	Output	Control
BUFGCE	I	O	CE
BUFGCE_1	I	O	CE

Primitive Functions

IBUFG

IBUFG is an input clock buffer with one clock input and one clock output.

IBUFGDS

IBUFGDS is a differential input clock buffer with two clock inputs (positive and negative polarity) and one clock output.

BUFG

All Virtex-II devices have 16 global clock buffers (each of which can be used as BUFG, BUFGMUX, or BUFGCE).

BUFG is a global clock buffer with one clock input and one clock output, driving a low-skew clock distribution network. The output follows the input, as shown in Figure 2-15.

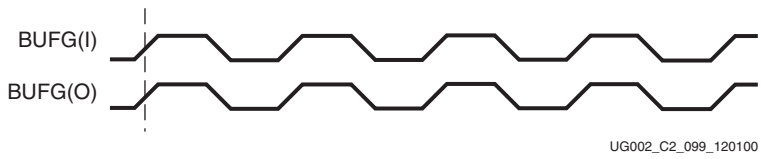


Figure 2-15: **BUFG Waveforms**

BUGMUX and BUFGMUX_1

BUFGMUX (see Figure 2-16) can switch between two unrelated, even asynchronous clocks. Basically, a Low on S selects the CLK0 input, a High on S selects the S1 input. Switching from one clock to the other is done in such a way that the output High and Low time is never shorter than the shortest High or Low time of either input clock. As long as the presently selected clock is High, any level change of S has no effect .

BUFGMUX is the preferred circuit for rising edge clocks, while BUFGMUX_1 is preferred for falling edge clocks.

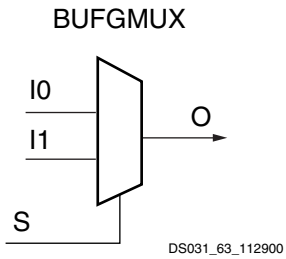


Figure 2-16: **Virtex-II BUFGMUX or BUFGMUX_1 Function**

Operation of the BUFGMUX Circuit

If the presently selected clock is Low while S changes, or if it goes Low after S has changed, the output is kept Low until the other ("to-be-selected") clock has made a transition from High to Low. At that instant, the new clock starts driving the output.

The two clock inputs can be asynchronous with regard to each other, and the S input can change at any time, except for a short setup time prior to the rising edge of the presently selected clock; that is, prior to the rising edge of the BUFGMUX output O. Violating this setup time requirement can result in an undefined runt pulse output.

Figure 2-17 shows a switchover from CLK0 to CLK1.

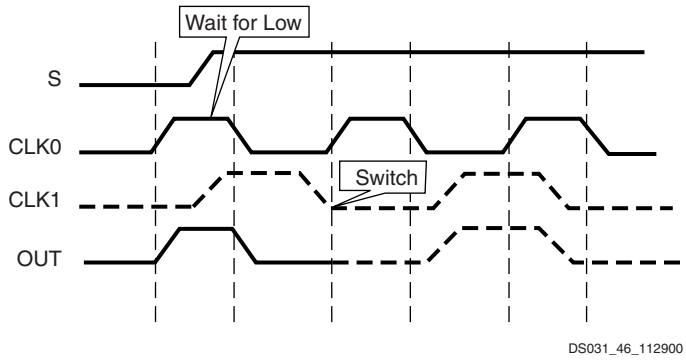


Figure 2-17: **BUFGMUX Waveform Diagram**

- The current clock is CLK0.
- S is activated High.
- If CLK0 is currently High, the multiplexer waits for CLK0 to go Low.
- Once CLK0 is Low, the multiplexer output stays Low until CLK1 transitions High to Low.
- When CLK1 transitions from High to Low, the output switches to CLK1.
- No glitches or short pulses can appear on the output.

Operation of the BUFGMUX_1 Circuit

If the presently selected clock is High while S changes, or if it goes High after S has changed, the output is kept High until the other ("to-be-selected") clock has made a transition from Low to High. At that instant, the new clock starts driving the output.

The two clock inputs can be asynchronous with regard to each other, and the S input can change at any time, except for a short setup time prior to the falling edge of the presently selected clock; that is, prior to the falling edge of the BUFGMUX output O. Violating this setup time requirement can result in an undefined runt pulse output.

Figure 2-18 shows a switchover from CLK0 to CLK1.

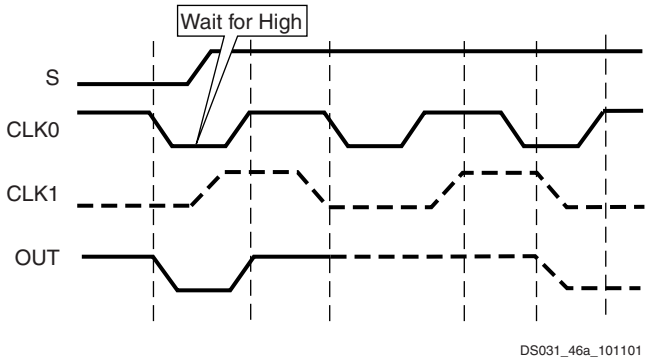


Figure 2-18: **BUFGMUX_1 Waveform Diagram**

- The current clock is CLK0.
- S is activated High.
- If CLK0 is currently Low, the multiplexer waits for CLK0 to go High.
- Once CLK0 is High, the multiplexer output stays High until CLK1 transitions Low to High.
- When CLK1 transitions from Low to High, the output switches to CLK1.
- No glitches or short pulses can appear on the output.

Submodules

BUFGCE and BUFGCE_1

BUFGCE and BUFGCE_1 are submodules based on BUFGMUX and BUFGMUX_1, respectively. BUFGCE and BUFGCE_1 are global clock buffers incorporating a smart enable function that avoids output glitches or runt pulses. The select signal must meet the setup time for the clock.

BUFGCE is the preferred circuit for clocking on the rising edge, while BUFGCE_1 is preferred when clocking on the falling edge.

Operation of the BUFGCE Circuit

If the CE input (see Figure 2-19) is active (High) prior to the incoming rising clock edge, this Low-to-High-to-Low clock pulse passes through the clock buffer. Any level change of CE during the incoming clock High time has no effect.

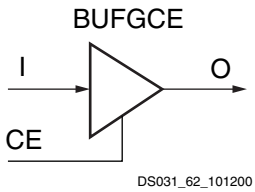


Figure 2-19: **Virtex-II BUFGCE or BUFGCE_1 Function**

If the CE input is inactive (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 time has no effect. CE must not change during a short setup window just prior to the rising clock edge on the BUFGCE input I. Violating this setup time requirement can result in an undefined runt pulse output.

This means the output stays Low when the clock is disabled, but it completes the clock-High pulse when the clock is being disabled, as shown in Figure 2-20.

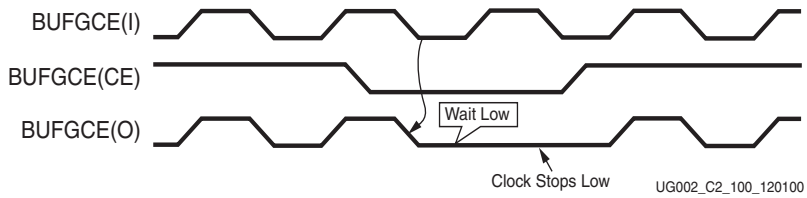


Figure 2-20: **BUFGCE Waveforms**

Operation of the BUFGCE_1 circuit

If the CE input is active (High) prior to the incoming falling clock edge, this High-to-Low-to-High clock pulse passes through the clock buffer. Any level change of CE during the incoming clock Low time has no effect.

If the CE input is inactive (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 time has no effect. CE must not change during a short setup window just prior to the falling clock edge on the BUFGCE input I. Violating this setup time requirement can result in an undefined runt pulse output.

This means the output stays High when the clock is disabled, but it completes the clock-Low pulse when the clock is being disabled, as shown in Figure 2-21.

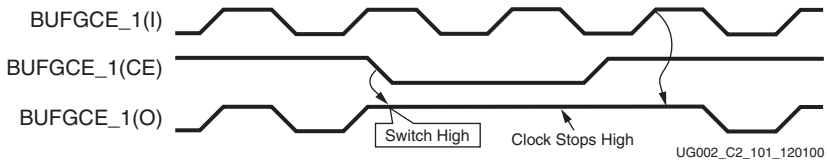


Figure 2-21: **BUFGCE_1 Waveforms**

When BUFGCE (or BUFGCE_1) is used with DCM outputs, a second BUFG can be used for clock feedback. Buffer sharing the inputs with BUFGCE is the preferred solution.

Summary

Table 2-7 shows the maximum resources available per Virtex-II device.

Table 2-7: **Resources per Virtex-II Device (from XC2V40 to XC2V8000)**

Resource	Maximum Number
Single-ended IBUFG (pads)	16
Differential IBUFGDS (pairs)	8
BUFG (Global Clock Buffer)	16
BUFGCE (or BUFGCE_1)	8
BUFGMUX (or BUFGMUX_1)	8

Characteristics

The following are characteristics of global clocks in Virtex-II devices:

- Low-skew clock distribution.
- Synchronous “glitch-free” multiplexer that avoids runt pulses. Switching between two asynchronous clock sources is usually considered unsafe, but it is safe with the Virtex-II global clock multiplexer.

- Any level change on S must meet a setup time requirement with respect to the signal on the output O (rising edge for BUFGMUX, falling edge for BUFGMUX_1). Any level change on CE must meet a setup time requirement with respect to the signal on the Input I (rising edge for BUFGCE, falling edge for BUFGCE_1).
- Two BUFGMUX (or BUFGMUX_1) resources can be cascaded to create a 3 to 1 clock multiplexer.

Location Constraints

BUFGMUX and BUFGMUX_1 (primitives) and IBUFG (IBUFGDS) instances can have LOC properties attached to them to constrain placement. The LOC properties use the following form to constrain a clock net:

```
NET "clock_name" LOC="BUFGMUX#P/S";
```

Each clock pad (or IBUFG) has a direct connection with a specific global clock multiplexer (input I0). A placement that does not conform to this rule causes the software to send a warning.

If the clock pad (or IBUFG) has LOC properties attached, the DCM allows place and route software maximum flexibility, as compared to a direct connection to the global clock buffer (BUFG).

Secondary Clock Network

If more clocks are required, the 24 horizontal and vertical long lines in Virtex-II devices can be used to route additional clock nets. Skew is minimized by the place and route software, if the USELOWSKEWLINES constraint is attached to the net.

VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available as examples (see "VHDL and Verilog Templates" on page 170) 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.

VHDL and Verilog Templates

The following are templates for primitives:

- BUFGMUX_INST
- BUFGMUX_1_INST

The following are templates for submodules:

- BUFGCE_SUBM
- BUFGCE_1_SUBM

As examples, the BUFGMUX_INST.vhd, BUFGMUX_1_INST.vhd, BUFGCE_SUBM.vhd, and BUFGCE_1_SUBM.vhd VHDL templates are shown. In addition, the BUFGMUX_INST.v, BUFGMUX_1_INST.v, BUFGCE_1_SUBM.v, and BUFGCE_SUBM.v Verilog templates are shown.

VHDL Template

```
-- Module: BUFGMUX_INST
-- Description: VHDL instantiation template
-- Global Clock Multiplexer (Switch Low)
-- Device: Virtex-II Family
------------------------------------------------------------------------
-- Component Declarations:
--
```

```
component BUFGMUX
  port (
  I0   : in std_logic;
        I1   : in std_logic;
        S    : in std_logic;
        O    : out std_logic
  );
end component;
--
-- Architecture  section:
--
-- Global Clock Buffer Instantiation
U_BUFGMUX: BUFGMUX
  port map (
  I0     => , -- insert clock input used when select (S) is Low
  I1     => , -- insert clock input used when select (S) is High
  S      => , -- insert Mux-Select input
  O      =>   -- insert clock output
  );
--
-------------------------------------------------------------------------
-- Module: BUFGMUX_1_INST
-- Description: VHDL instantiation template
--  Global Clock Multiplexer (Switch High)
--
-- Device: Virtex-II Family
-------------------------------------------------------------------------
-- Component Declarations:
component BUFGMUX_1
  port (
  I0   : in std_logic;
        I1   : in std_logic;
        S    : in std_logic;
        O    : out std_logic
  );
end component;
--
-- Architecture  section:
--
-- Global Clock Buffer Instantiation
U_BUFGMUX_1: BUFGMUX_1
  port map (
  I0     => , -- insert clock input used when select (S) is Low
  I1     => , -- insert clock input used when select (S) is High
  S      => , -- insert Mux-Select input
  O      =>   -- insert clock output
  );
--
-------------------------------------------------------------------------
-- Module: BUFGCE_SUBM
-- Description: VHDL instantiation template
-- Global Clock Buffer with Clock Enable:
-- Input Clock Buffer to BUFGMUX - Clock disabled = Low
-- Device: Virtex-II Family
-------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
```

```
--
entity BUFGCE_SUBM is
  port (
        I:  in std_logic;
        CE: in std_logic;
        O:  out std_logic
       );
end BUFGCE_SUBM;
--
architecture BUFGCE_SUBM_arch of BUFGCE_SUBM is
--
-- Component Declarations:
component BUFGMUX
  port (
  I0   : in std_logic;
        I1   : in std_logic;
        S    : in std_logic;
        O    : out std_logic
  );
end component;
--
-- signal declarations
signal GND : std_logic;
signal CE_B : std_logic;
--
begin
GND <= '0';
--
CE_B <= not CE;
--
-- Global Clock Buffer Instantiation
U_BUFGMUX: BUFGMUX
  port map (
  I0     => I,
  I1     => GND,
  S      => CE_B,
  O      => O
  );
--
end BUFGCE_SUBM_arch;
------------------------------------------------------------------------
-- Module: BUFGCE_1_SUBM
-- Description: VHDL instantiation template
-- Global Clock Buffer with Clock Enable:
-- Input Clock Buffer to BUFGMUX_1 - Clock disabled = High
-- Device: Virtex-II Family
------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
--
entity BUFGCE_1_SUBM is
  port (
        I:  in std_logic;
        CE: in std_logic;
        O:  out std_logic
       );
end BUFGCE_1_SUBM;
```

```
--
architecture BUFGCE_1_SUBM_arch of BUFGCE_1_SUBM is
--
-- Component Declarations:
component BUFGMUX_1
  port (
  I0   : in std_logic;
         I1   : in std_logic;
         S    : in std_logic;
         O    : out std_logic
  );
end component;
--
-- signal declarations
signal VCC : std_logic;
--
signal CE_B : std_logic;
--
begin
VCC <= '1';
--
CE_B <= not CE;
--
-- Global Clock Buffer Instantiation
U_BUFGMUX_1: BUFGMUX_1
  port map (
  I0     => I,
  I1     => VCC,
  S      => CE_B,
  O      => O
  );
--
end BUFGCE_1_SUBM_arch;
```


Verilog Template

```
//------------------------------------------------------------------------
// Module:      BUFGMUX_INST
// Description: Verilog Instantiation Template
//  Global Clock Multiplexer (Switch Low)
//
//
// Device: Virtex-II Family
//------------------------------------------------------------------------
//
//BUFGMUX Instantiation
BUFGMUX  U_BUFGMUX
             (.I0(),  // insert clock input used when select(S) is Low
              .I1(),  // insert clock input used when select(S) is High
               .S(),   // insert Mux-Select input
               .O()    // insert clock output
              );
//------------------------------------------------------------------------
// Module:      BUFGMUX_1_INST
// Description: Verilog Instantiation Template
//  Global Clock Multiplexer (Switch High)
//
//
// Device: Virtex-II Family
//------------------------------------------------------------------------
//
//BUFGMUX_1 Instantiation
BUFGMUX_1  U_BUFGMUX_1
```

```
                (.I0(),  // insert clock input used when select(S) is Low
                .I1(),  // insert clock input used when select(S) is High
                 .S(),   // insert Mux-Select input
                 .O()    // insert clock output
                );
//-----------------------------------------------------------------------
// Module:      BUFGCE_SUBM
// Description: Verilog Submodule
//  Global Clock Buffer with Clock Enable:
//  Input Clock Buffer to BUFGMUX - Clock disabled = Low
//
// Device: Virtex-II Family
//-----------------------------------------------------------------------
module BUFGCE_SUBM (I,
                    CE,
                    O);
input   I,
        CE;

output  O;

wire GND;

assign GND = 1'b0;

BUFGMUX U_BUFGMUX
             (.I0(I),
              .I1(GND),
              .S(~CE),
              .O(O)
             );
//
endmodule
//-----------------------------------------------------------------------
// Module: BUFGCE_1_SUBM
// Description: Verilog Submodule
// Global Clock Buffer with Clock Enable:
// Input Clock Buffer to BUFGMUX_1 - Clock disabled = High
//
// Device: Virtex-II Family
//-----------------------------------------------------------------------

module BUFGCE_1_SUBM (I,
                      CE,
                      O);
input   I,
        CE;

output  O;

wire VCC;

assign VCC = 1'b1;

BUFGMUX_1 U_BUFGMUX_1
             (.I0(I),
              .I1(VCC),
              .S(~CE),
              .O(O)
             );
//
endmodule
```

Using Digital Clock Managers (DCMs)

Overview

Virtex-II devices have 4 to 12 DCMs, and each DCM provides a wide range of powerful clock management features:

- **Clock De-skew**: The DCM contains a digitally-controlled feedback circuit (delay-locked loop) that can completely eliminate clock distribution delays. Clock de-skew works as follows:

  The incoming clock drives a long chain of delay elements (individual small buffers). A wide multiplexer selects any one of these buffers as an output. A controller drives the select inputs of this multiplexer. The phase detector in this controller compares the incoming clock signal (CLKIN) against a feedback input (CLKFB), which must be another version of the same clock signal, usually from the far end of the internal clock distribution network (but it can also be from an output pin).

  The phase detector steers the controller to adjust the tap selection, and thus the through-delay in the DCM, in such a way that the two inputs to the phase comparator coincide.(This is a typical servo loop.) The tap controller adds exactly the right amount of delay to the clock distribution network to give it a total delay of one full clock period. For a repetitive clock signal, this effectively eliminates the clock distribution delay completely.

- **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 (÷1.5, ÷2, ÷2.5, and so forth, up to ÷15 and ÷16.)

  Two other outputs (CLKFX and CLKFX180) provide an output frequency that is 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-II Data Sheet**. An internal calculator figures out the appropriate tap selection, so that the output edge coincides with the input clock whenever that is 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 every 5 input periods = every 9 output periods.

- **Phase Shifting**: Three outputs drive the same frequency as CLCK0 but are delayed by 1/4, 1/2, and 3/4 of a clock period. An additional control optionally shifts all nine clock outputs by a fixed fraction of the clock period (defined during configuration, and described in multiples of the clock period divided by 256).

  The user can also dynamically and repetitively move the phase forwards or backwards by one unit of the clock period divided by 256. Note that any such phase shift is always invoked as a specific fraction of the clock period, but is always implemented by moving delay taps with a resolution of DCM_TAP (see DCM Timing Parameters in the **Virtex-II Data Sheet**).

- **General Control Signals**: The RST input, when High, resets the entire DCM. The LOCKED output is High when all enabled DCM circuits have locked. The active High STATUS outputs indicate the following:
  - Phase Shift Overflow (STATUS[0])
  - CLKIN Stopped (STATUS[1])
  - CLKFX Stopped (STATUS[2])

Clock De-Skew

The Virtex-II Digital Clock Manager (DCM) offers a fully digital, dedicated on-chip de-skew circuit providing zero propagation delay, low clock skew between output clock signals distributed throughout the device, and advanced clock domain control. These features can be used to implement several circuits that improve and simplify system level design.

Any four of the nine outputs of the DCM can be used to drive a global clock network. All DCM outputs can drive general interconnect at the same time; for example, DCM output can be used to generate board-level clocks. The well-buffered global clock distribution network minimizes clock skew caused by loading differences. By monitoring a sample of the output clock (CLK0 or CLK2X), the de-skew 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.

Figure 2-22 shows all of the inputs and outputs relevant to the DCM de-skew feature.

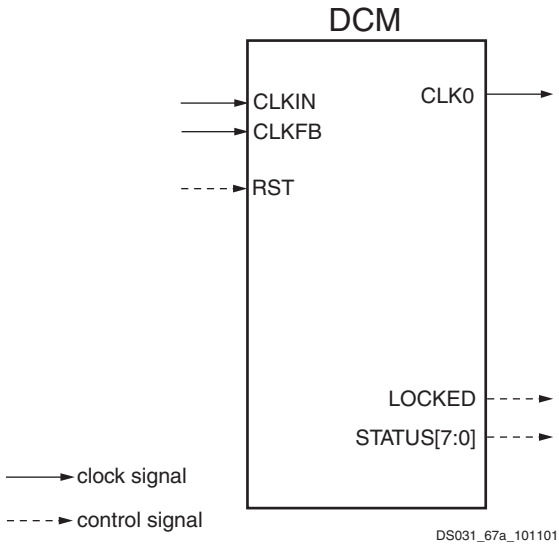


Figure 2-22: **Clock De-Skew Outputs**

The de-skew feature can also act as a clock mirror. By driving the CLK0 or CLK2X output off-chip and then back in again, the de-skew feature can be used to de-skew a board-level clock serving multiple devices.

By taking advantage of the de-skew circuit to remove on-chip clock delay, the designer can greatly simplify and improve system level design involving high-fanout, high-performance clocks.

Operation

A de-skew circuit in its simplest form consists of variable delay line and control logic. The delay line produces a delayed version of the input clock (CLKIN). The clock distribution network routes the clock to all internal registers and to the clock feedback CLKFB pin. The control logic samples the input clock, as well as the feedback clock, and adjusts the delay line.

For optimum performance, the Virtex-II DCM uses a discrete digital delay line, which is a series of buffer elements each with an intrinsic delay of less than DCM_TAP (see AC characteristics in the **Virtex-II Data Sheet**).

A de-skew circuit works by inserting delay between the input clock and the feedback clock until the two rising edges align, putting the two clocks 360 degrees out of phase, which means they are in phase. When the edges from the input clock line up with the edges from the feedback clock, the DCM achieves "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.

Input Clock Requirements

The clock input of the DCM can be driven either by an IBUFG, an IBUF, or a BUFGMUX. An LVDS clock can also be used as input.

The output clock signal of a DCM, essentially a delayed version of the input clock signal, reflects any instability on the input clock in the output waveform. A DCM cannot improve the input jitter. The DCM input clock requirements are specified in the **Virtex-II Data Sheet**.

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

Input Clock Changes

Changing the period of the input clock beyond the maximum drift amount requires a manual reset of the DCM. Failure to reset the DCM produces an unreliable lock signal and output clock.

It is possible to stop the input clock with little impact to the de-skew circuit. The clock should be stopped for no more than 100 ms to minimize the effect of device cooling, which would change the tap delays. The clock should be stopped during a Low phase, and when restored, must generate a full High half-period. During this time, LOCKED stays High and remains High when the clock is restored. So a High on LOCKED does not necessarily mean that a valid clock is available.

When the clock is being stopped, one to four more clock cycles are still generated as the delay line is flushed. When the clock is restarted, the output clock cycles are not generated for one to four clocks as the delay line is filled. The most common case is two or three clocks. In a similar manner, a phase shift of the input clock is also possible. The phase shift propagates to the output one to four clocks after the original shift, with no disruption to the DCM control.

Output Clocks

Some restrictions apply regarding the connectivity of the output pins. The DCM clock outputs can each drive an OBUF, a global clock buffer BUFGMUX, or they can route directly to the clock input of a synchronous element. The DCM clock outputs can drive BUFGMUXs that are on the same edge of the device (top or bottom).

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 and can exhibit glitches, spikes, or other spurious movement.

Characteristics of the De-Skew Circuit

- Can eliminate clock distribution delay by effectively adding one clock period delay. Clocks are de-skewed to within CLKOUT_PHASE, specified in the **Virtex-II Data Sheet**.
- Can be used to eliminate on-chip as well as off-chip clock delay.
- Has 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, cannot tolerate large variations of the input frequency.
- De-skew circuit is part of the DCM, which also includes phase adjustment, frequency synthesis, and spread spectrum techniques that are described in this document.
- Does not eliminate jitter. The de-skew circuit output jitter is the sum of input jitter and some jitter value that the de-skew circuit might add.
- The completion of configuration can be delayed until after DCM locks to guarantee the system clock is established prior to initiating the device.

Port Signals

Source Clock Input — CLKIN

The CLKIN pin provides the user source clock (the clock signal on which the de-skew circuit operates) to the DCM. The CLKIN frequency must fall in the ranges specified in the **Virtex-II Data Sheet**. The clock input signal can be provided by one of the following:

IBUF — Input buffer

IBUFG — Global clock input buffer on the same edge of the device (top or bottom)

BUFGMUX — Internal global clock buffer

Feedback Clock Input — CLKFB

A reference or feedback signal is required to delay-compensate the output. Connect only the CLK0 or CLK2X DCM outputs to the feedback clock input (CLKFB) pin to provide the necessary feedback to the DCM. The feedback clock input signal can be driven by an internal global clock buffer (BUFGMUX), one of the global clock input buffers (IBUFG) on the same edge of the device (top or bottom), or IBUF (the input buffer.)

If an IBUFG sources the CLKFB pin, the following special rules apply:

1. An external input port must source the signal that drives the IBUFG input pin.
2. That signal must directly drive only OBUFs and nothing else.

Reset Input — RST

When the reset pin is activated, the LOCKED signal deactivates within four source clock cycles. The RST pin, active High, must either connect to a dynamic signal or be tied to ground. As the DCM delay taps reset to zero, glitches can occur on the DCM clock output pins. Activation of the RST pin can also severely affect the duty cycle of the clock output pins. Furthermore, the DCM output clocks no longer de-skew with respect to one another. For these reasons, use the reset pin only when reconfiguring the device or changing the input frequency. The reset input signal is asynchronous and should be held HIGH for at least 2 ns. It takes approximately 120 µs for the DCM to achieve lock after a reset in the slowest frequency range. The DCM locks faster at higher frequencies. See the LOCK_DLL teiming parameter in the **Virtex-II Data Sheet**.

Locked Output — LOCKED

In order to achieve lock, the DCM may need to sample several thousand clock cycles. After the DCM achieves lock, the LOCKED signal goes High. The DCM timing parameter section of the **Virtex-II Data Sheet** provides estimates for locking times.

To guarantee that the system clock is established prior to the device "waking up," the DCM can delay the completion of the device configuration process until after the DCM locks. The STARTUP_WAIT attribute activates this feature.

Until the LOCKED signal activates, 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.

Status - STATUS

The STATUS output is an 8-bit output, of which STATUS[1] reveals the loss of the input clock, CLKIN to the DCM.

Attributes

The following attributes provide access to some of the Virtex-II series de-skew features, (for example, clock division and duty cycle correction).

Frequency Mode

The de-skew feature of the DCM is achieved with a delay-locked loop (DLL). This attribute specifies either the high or low-frequency mode of the DLL. The default is low-frequency mode. In high-frequency mode, the only outputs available from the DLL are the CLK0, CLK180, CLKDV, and LOCKED. (CLK90, CLK270, CLK2X, and CLK2X180 are not available in high-frequency mode.) The frequency ranges for both frequency modes are specified in the **Virtex-II Data Sheet**. To set the DLL to high-frequency mode, attach the DLL_FREQUENCY_MODE=HIGH attribute in the source code or schematic.

Feedback Input

This attribute specifies the feedback input to the DCM (CLK0, or CLK2x). CLK0 is the default feedback. When both the CLK0 and the CLK2x outputs are used internally or externally to the device, the feedback input can be either the CLK0 or CLK2x. In order to set the feedback to CLK2X, attach the CLOCK_FEEDBACK=2X attribute in the source code or schematic.

Duty Cycle Correction

The 1x clock outputs, CLK0, CLK90, CLK180, and CLK270, use the duty cycle corrected default such that they exhibit a 50/50 duty cycle. The DUTY_CYCLE_CORRECTION attribute (by default TRUE) controls this feature.

To deactivate the DCM duty cycle correction for the 1x clock outputs, attach the DUTY_CYCLE_CORRECTION=FALSE attribute in the source code or schematic. This makes the output clocks have the same duty cycle as the source clock.

Startup Delay

The default value of the STARTUP_WAIT attribute 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 specified cycle until the DCM locks. For details, see Chapter 3: Configuration and Appendix B: BitGen and PROMGen Switches and Options.

Legacy Support

The Virtex/Virtex-E library primitives/sub modules are supported in Virtex-II for legacy purposes. The following are supported primitives/submodules:

- CLKDLL
- CLKDLLE
- CLKDLLHF
- BUFGDLL

Library Primitive

Only a single library primitive is available for the DLL, a part of the DCM. It is labeled the 'DCM' primitive.

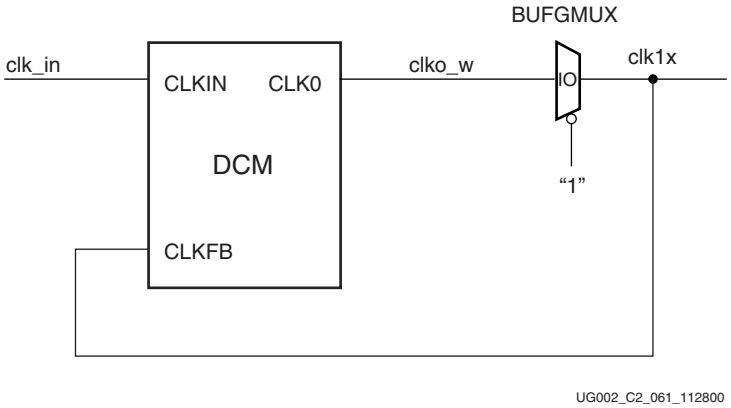


Figure 2-23: **BUFG_CLK0_SUBM**

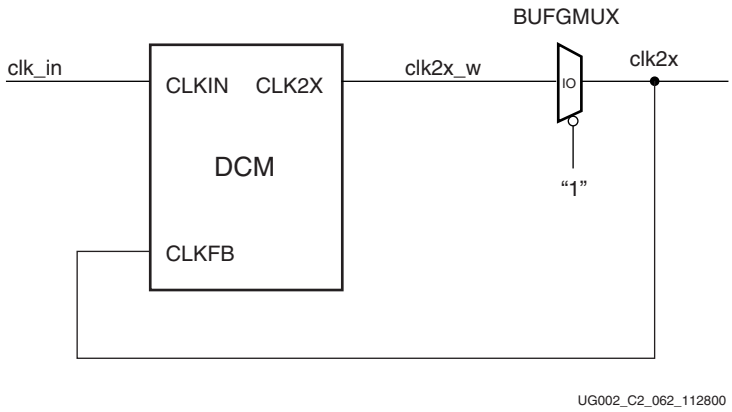


Figure 2-24: **BUFG_CLK2X_SUBM**

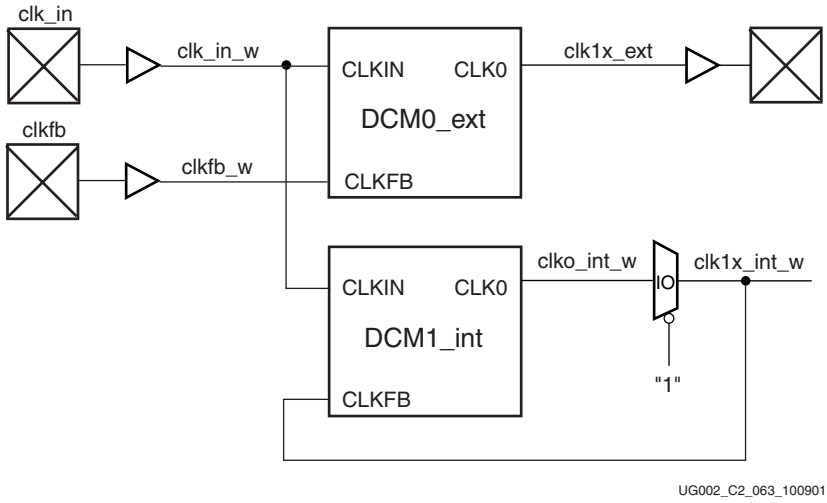


Figure 2-25: **BUFG_CLK0_FB_SUBM**

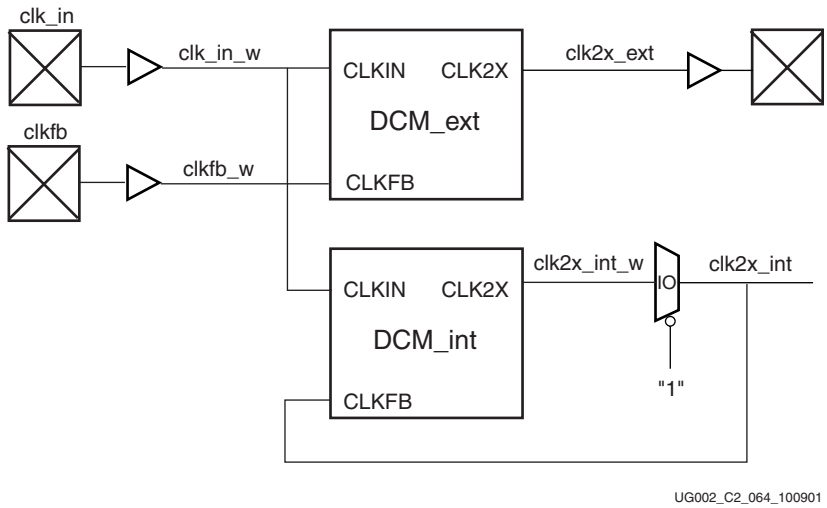


Figure 2-26: **BUFG_CLK2X_FB_SUBM**

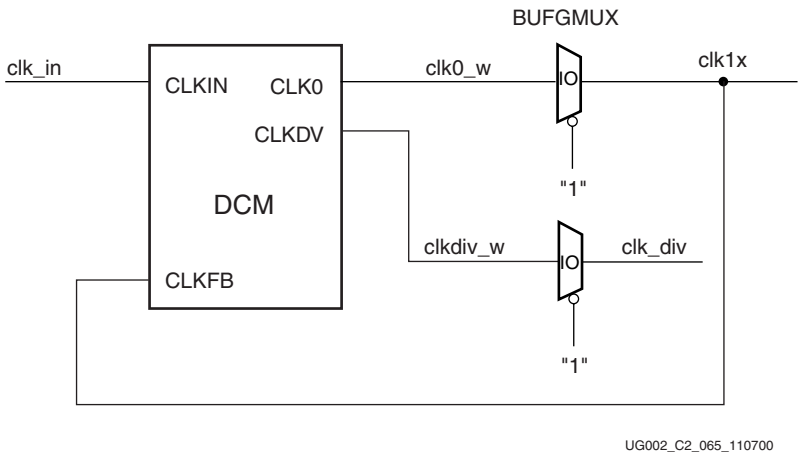


Figure 2-27: **BUFG_CLKDV_SUBM**

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 divided output clocks 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 a fully digital, dedicated Frequency Synthesizer output (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 multipler (numerator) and D is the divisor (denominator).

The two counter-phase frequency synthesized outputs can drive global clock routing networks within the device. The well-buffered global clock distribution network minimizes clock skew due to differences in distance or loading. See Figure 2-28.

Operation

The DCM clock output CLKFX is any M/D product 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-II Data Sheet**. The frequency synthesizer output is

phase aligned to the clock output, CLK0, only if feedback is provided to the CLKFB input of the DCM.

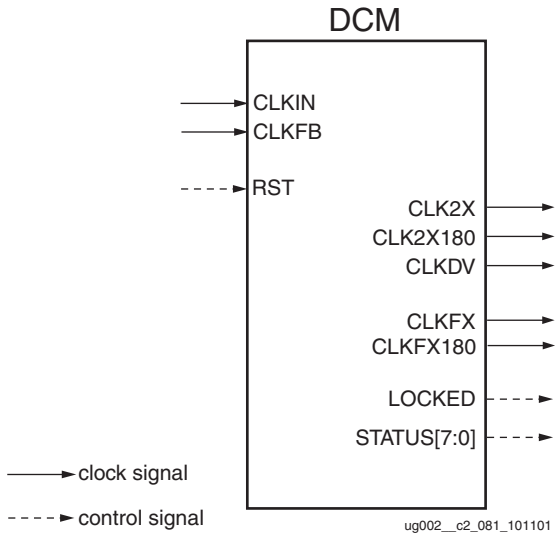


Figure 2-28: **Frequency Synthesis Outputs**

The internal operation of the frequency synthesizer is complex and beyond the scope of this document. The frequency synthesizer multiplies the incoming frequencies by the pre-calculated quotient M/D and generates the correct output frequencies as long as it is within the range specified in the **Virtex-II Data Sheet**.

For example, assume input frequency = 50 MHz, M = 25, and D = 8 (note that M and D values have no common factors and hence cannot be reduced). The output frequency is correctly 156.25 MHz, although 25 x 50 MHz = 1.25 GHz and 50 MHz / 8 = 6.25 MHz, and both of these values are far 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. The user should divide M and D by the largest common factor.
- The outputs are phase aligned with CLK0 when CLKFB is connected.

Port Signals

Source Clock Input — CLKIN

The CLKIN pin provides the user source clock to the DCM. The CLKIN frequency must fall in the ranges specified in the **Virtex-II Data Sheet**. The clock input signal can be provided by one of the following:

- IBUF — Input buffer
- IBUFG — Global clock input buffer
- BUFGMUX — Internal global clock buffer

2x Clock Output — CLK2X

The CLK2X output provides a frequency-doubled clock with an automatic 50/50 duty-cycle correction. This output is not available in high-frequency mode.

Until the DCM has achieved lock, 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 source clock.

Clock Divide Output — CLKDV

The clock divide output pin CLKDV provides a lower frequency version of the source clock. The CLKDV_DIVIDE property controls CLKDV such that the source clock is divided by N where N is either 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.

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 all integer values of the division factor N in high-frequency mode.

Frequency Synthesized Clock Output - CLKFX

The CLKFX output provides a frequency-synthesized clock (M/D * CLKIN) with a 50/50 duty cycle. For the CLKFX output to be phase-aligned with CLKIN, the clock feedback (CLK0) must be provided at the CLKFB input. With M and D adjusted such that they have no common factor, the alignment occurs only once every D input clock cycles.

Frequency Synthesized Clock Output 180º Phase Shifted - CLKFX180

The CLKFX180 output is a 180º phase shifted version of the CLKFX clock output, also with a 50/50 duty cycle.

Locked Output — LOCKED

The LOCKED signal is activated after the DCM has achieved the parameter values set by the user parameters. To guarantee that the system clock is established prior to the device "waking up," the DCM can delay the completion of the device configuration process until after the DCM locks. The STARTUP_WAIT attribute activates this feature. Until the LOCKED signal activates, the DCM output clocks are not valid and can exhibit glitches, spikes, or other spurious signals.

Reset Input — RST

When the reset pin activates, the LOCKED signal deactivates within four source clock cycles. The M and D values at configuration are maintained after the reset. The RST pin, active High, must either connect to a dynamic signal or be tied to ground. Activation of the RST pin can also severely affect the duty cycle of the clock output pins. For this reason, activate the reset pin only when reconfiguring the device or changing the input frequency. The reset input signal is asynchronous and should be held High for at least 2 ns.

Status - STATUS

The STATUS output is an 8-bit output:

- STATUS[1] indicates the loss of the input clock, CLKIN, only when CLKFB is connected.
- STATUS[2] indicates loss of CLKFX and CLKFX180 even though LOCKED might still be High. Note that this "CLKFX stopped" status functions only when CLKIN is present.

Attributes

The following attributes provide access to some of the Virtex-II series frequency synthesis features, (for example, clock multiplication, clock division).

Clock Divide

The CLKDV_DIVIDE attribute specifies how the signal on the CLKDV pin is frequency divided with respect to the CLK0 pin. The values allowed for this attribute 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.

Frequency Mode for Frequency Synthesis

This attribute specifies either the high or low-frequency mode of the frequency synthesizer. The default is low-frequency mode. The frequency ranges for both frequency modes are specified in the **Virtex-II Data Sheet**.

To set the frequency synthesizer to high-frequency mode, attach the DFS_FREQUENCY_MODE=HIGH attribute in the source code or schematic.

Multiply/Divide Attribute

The M and D values can be set using the CLKFX_MULTIPLY and the CLKFX_DIVIDE attributes. The default settings are M = 4 and D = 1.

Startup Delay

The default value of the STARTUP_WAIT attribute 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 specified cycle until the DCM locks. For details, see Chapter 3: Configuration and Appendix B: BitGen and PROMGen Switches and Options.

Submodules

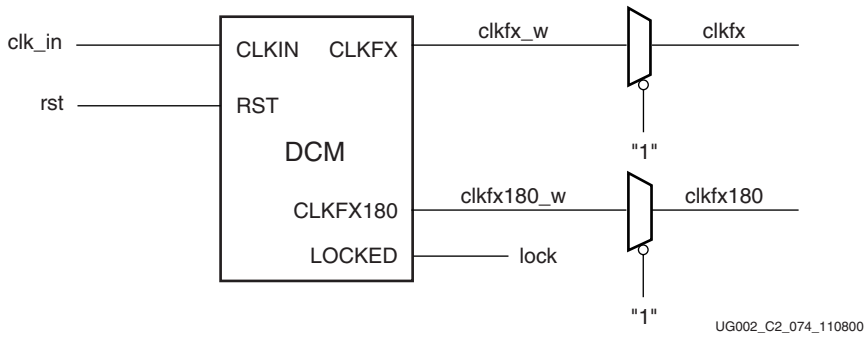


Figure 2-29: **BUFG_DFS_SUBM**

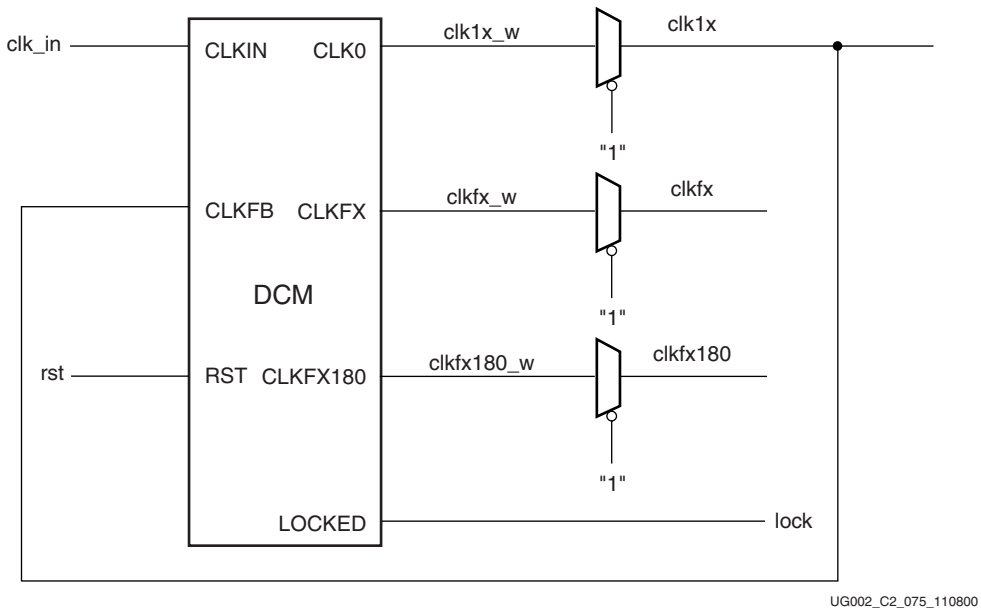


Figure 2-30: **BUFG_DFS_FB_SUBM**

Phase Shifting

The DCM can also provide coarse and fine-grained phase shifting. The CLK0, CLK90, CLK180, and CLK270 outputs are each phase shifted by ¼ of the input clock period relative to each other, providing coarse phase control. Note that CLK90 and CLK270 are not available in high-frequency mode.

Operation

Figure 2-31 shows a block diagram of the DCM and all of the outputs affected by the circuitry of the phase shift feature.

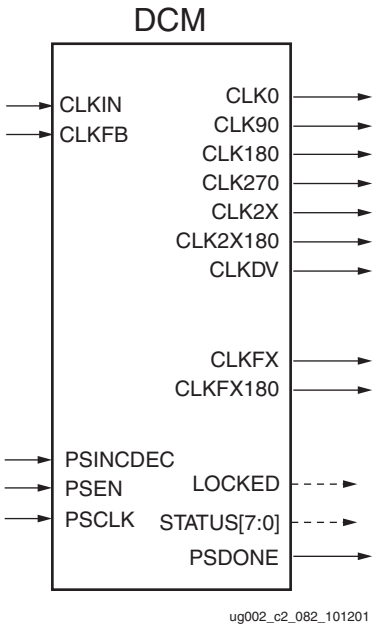


Figure 2-31: **Phase Shift Outputs**

Fine-phase adjustment affects all nine DCM output clocks. When activated, the phase shift between the rising edges of CLKIN and CLKFB is a specified fraction of the input clock period.

In variable mode, the PHASE_SHIFT value can also be dynamically incremented or decremented as determined by PSINCDEC synchronously to PSCLK, when the PSEN input is active. Figure 2-32 illustrates the effects of fine-phase shifting.

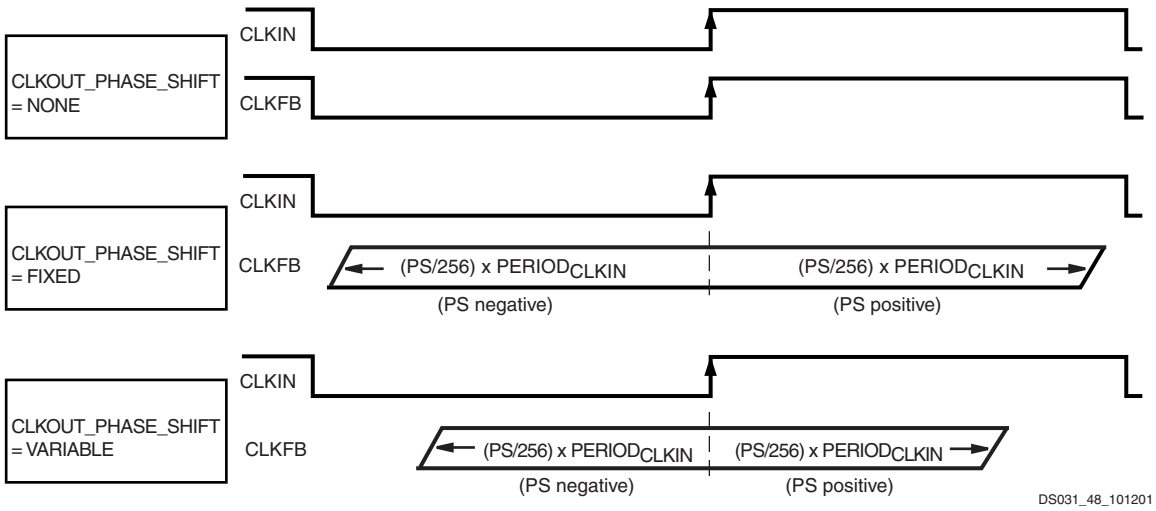


Figure 2-32: **Phase Shift Effects**

Two separate components of the phase shift range must be understood:

- PHASE_SHIFT attribute range
- FINE_SHIFT_RANGE DCM timing parameter range

The PHASE_SHIFT attribute is the numerator in the following equation:

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

The full range of this attribute is always -255 to +255, but its practical range varies with CLKIN frequency, as constrained by the FINE_SHIFT_RANGE component, which 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. Across process, voltage, and temperature, this absolute range is guaranteed to be as specified in the DCM Timing Parameters section of the **Virtex-II Data Sheet**.

Absolute range (fixed mode) = ± FINE_SHIFT_RANGE

Absolute range (variable mode) = ± FINE_SHIFT_RANGE/2

The reason for the difference between fixed and variable modes is as follows. For variable mode to allow symmetric, dynamic sweeps from -255/256 to +255/256, the DCM sets the "zero phase skew" point as the middle of the delay line, thus dividing the total delay line range in half. In fixed mode, since the PHASE_SHIFT value never changes after configuration, the entire delay line is available for insertion into either the CLKIN or CLKFB path (to create either positive or negative skew).

Taking both of these components into consideration, the following are some usage examples:

- If $\text{PERIOD}_{\text{CLKIN}}$ = two times FINE_SHIFT_RANGE, then PHASE_SHIFT in fixed mode is limited to ± 128, and in variable mode it is limited to ± 64.
- If $\text{PERIOD}_{\text{CLKIN}}$ = FINE_SHIFT_RANGE, then PHASE_SHIFT in fixed mode is limited to ± 255, and in variable mode it is limited to ± 128.
- If $\text{PERIOD}_{\text{CLKIN}} \leq$ half of the FINE_SHIFT_RANGE, then PHASE_SHIFT is limited to ± 255 in either mode.

In variable mode, the phase factor can be changed by activating PSEN for one period of PSCLK. Increments or decrements to the phase factor can be made by setting the PSINCDEC pin to a High or Low, respectively. When the de-skew circuit has completed an increment or decrement operation, the signal PSDONE goes High for a single PSCLK cycle. This indicates to the user that the next change may be made.

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.

Phase Shift Characteristics

- Offers fine-phase adjustment with a resolution of ±1/256 of the clock period (or ± one DCM_TAP, whichever is greater) by configuration and also dynamically under user control.
- The phase shift settings affect all nine DCM outputs.
- V_{CC} and temperature do not affect the phase shift.

1x Clock Outputs — CLK[0|90|180|270]

The 1x clock output pin CLK0 represents a delay-compensated version of the source clock (CLKIN) signal. In low-frequency mode, the DCM provides three phase-shifted versions of the CLK0 signal (CLK90, CLK180, and CLK270), whereas in high-frequency mode, only the 180 phase-shifted version is provided. All four (including CLK0) of the phase shifted outputs can be used simultaneously in low-frequency mode. The relationship between phase shift and the corresponding period shift appears in Table 2-8. The timing diagrams in Figure 2-33 illustrate the DLL clock output characteristics.

Table 2-8: **Relationship of Phase-Shifted Output Clock to Period Shift**

Phase (degrees)	% Period Shift
0	0%
90	25%
180	50%
270	75%

By default, the DCM provides a 50/50 duty cycle correction on all 1x clock outputs. The DUTY_CYCLE_CORRECTION attribute (TRUE by default), controls this feature. Attach the DUTY_CYCLE_CORRECTION=FALSE property to the DCM symbol in order to deactivate the DCM duty cycle correction. With duty cycle correction deactivated, the output clocks have the same duty cycle as the source clock.

The DCM clock outputs can drive an OBUF, a BUFGMUX, or they can route directly to the clock input of a synchronous element.

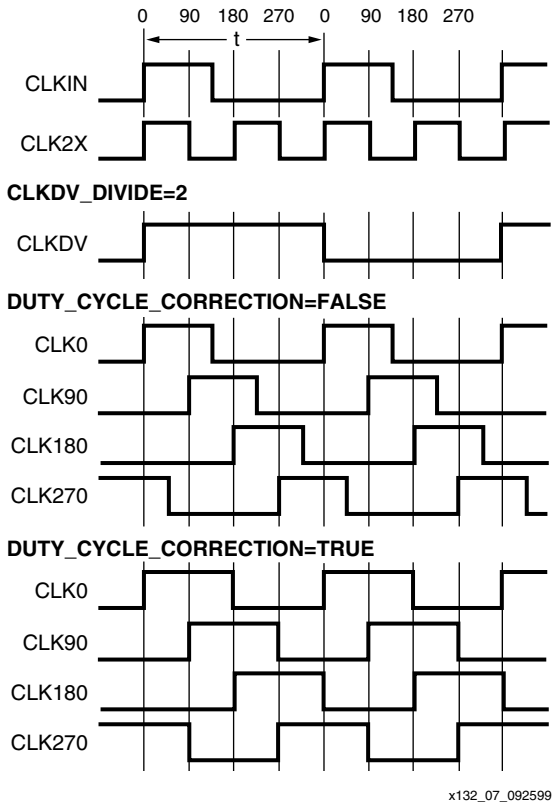


Figure 2-33: **DLL Output Characteristics**

Source Clock Input — CLKIN

The CLKIN pin provides the user source clock to the DCM. The CLKIN frequency must fall in the ranges specified in the **Virtex-II Data Sheet**. The clock input signal can be provided by one of the following:

- IBUF — Input buffer
- IBUFG — Global clock input buffer
- BUFGMUX — Internal global clock buffer

Feedback Clock Input — CLKFB

A DCM requires a reference or feedback signal to provide delay-compensated output. Connect only the CLK0 or CLK2X DCM outputs to the feedback clock input (CLKFB) pin to provide the necessary feedback to the DCM. The feedback clock input signal can be driven by an internal global clock buffer (BUFGMUX), one of the global clock input buffers (IBUFG) on the same edge of the device (top or bottom), or IBUF (the input buffer.)

If an IBUFG sources the CLKFB pin, the following special rules apply:

1. An external input port must source the signal that drives the IBUFG input pin.
2. That signal must directly drive only OBUFs and nothing else.

Phase Shift Clock - PSCLK

The PSCLK input can be sourced by the CLKIN signal to the DCM, or it can be a lower or higher frequency signal provided from any clock source (external or internal). The frequency range of PSCLK is defined by PSCLK_FREQ_LF/HF (see the **Virtex-II Data Sheet**). This input has to be tied to ground when the CLKOUT_PHASE_SHIFT attribute is set to NONE or FIXED.

Phase Shift Increment/Decrement - PSINCDEC

The PSINCDEC signal is synchronous to PSCLK and is used to increment or decrement the phase shift factor. In order to increment or decrement the phase shift by 1/256 of clock period, the PSINCDEC signal must be High for increment or Low for decrement. This input has to be tied to ground when the CLKOUT_PHASE_SHIFT attribute is set to NONE or FIXED.

Phase Shift Enable - PSEN

To initiate a variable phase-shift operation, the PSEN input must be activated for one period of PSCLK. The phase change becomes effective after up to 100 CLKIN pulse cycles plus three PSCLK cycles, and is indicated by a High pulse on PSDONE. During the phase transition there are no sporadic changes or glitches on any output. PSEN must be tied to ground when the CLKOUT_PHASE_SHIFT attribute is set to NONE or FIXED.

Reset Input — RST

When the reset pin is activated, the LOCKED signal deactivates within four source clock cycles. After reset, the phase shift value is set to its value at configuration in both the fixed and variable modes. The RST pin, active High, must either connect to a dynamic signal or be tied to ground. Activation of the RST pin can also severely affect the duty cycle of the clock output pins. For this reason, activate the reset pin only when reconfiguring the device or changing the input frequency. The reset input signal is asynchronous and should be held High for at least 2 ns.

Locked Output — LOCKED

The LOCKED signal activates after the DCM has achieved lock. To guarantee that the system clock is established prior to the device “waking up,” the DCM can delay the completion of the device configuration process until after the DCM locks. The STARTUP_WAIT attribute activates this feature. Until the LOCKED signal activates, the DCM output clocks are not valid and can exhibit glitches, spikes, or other spurious movement. For details, refer to Chapter 3: Configuration.

Phase Shift DONE - PSDONE

The PSDONE signal is synchronous to PSCLK and it indicates, by pulsing High for one period of PSCLK, that the requested phase shift was achieved. This signal also indicates to the user that a new change to the phase shift numerator can be made. This output signal is not valid if the phase shift feature is not being used or is in FIXED mode.

Status - STATUS

STATUS[0] indicates the overflow of the phase shift numerator and that the absolute delay range of the phase shift delay line is exceeded.

Attributes

The following attributes provide access to the Virtex-II fine-phase adjustment capability.

Clock Out Phase Shift

The CLKOUT_PHASE_SHIFT attribute controls the use of the PHASE_SHIFT value. It can be set to NONE, FIXED, or VARIABLE. By default, this attribute is set to NONE, indicating that the phase shift feature is not being used. When this attribute is set to NONE, the PHASE_SHIFT value has no effect on the DCM outputs. 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. The effects of the CLKOUT_PHASE_SHIFT attribute are shown in Figure 2-32.

PHASE_SHIFT

This attribute specifies the phase shift numerator as any value from -255 to 255.

Submodules

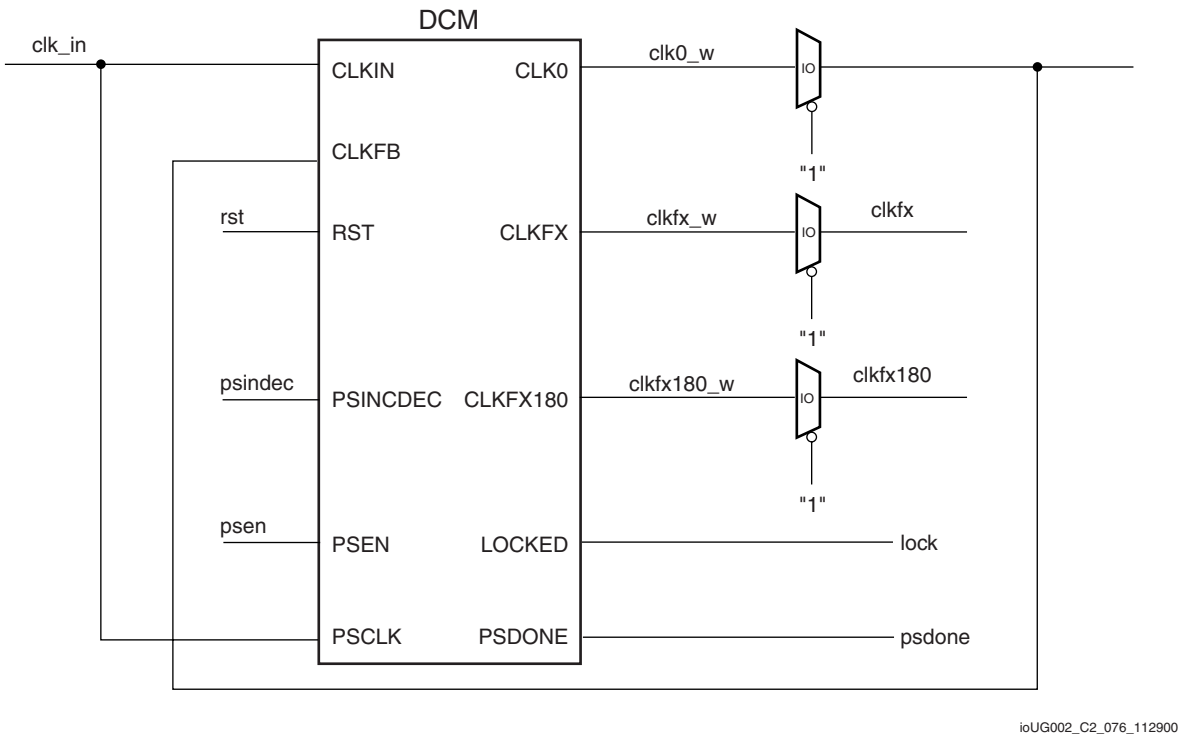


Figure 2-34: **BUFG_PHASE_CLKFX_FB_SUBM**

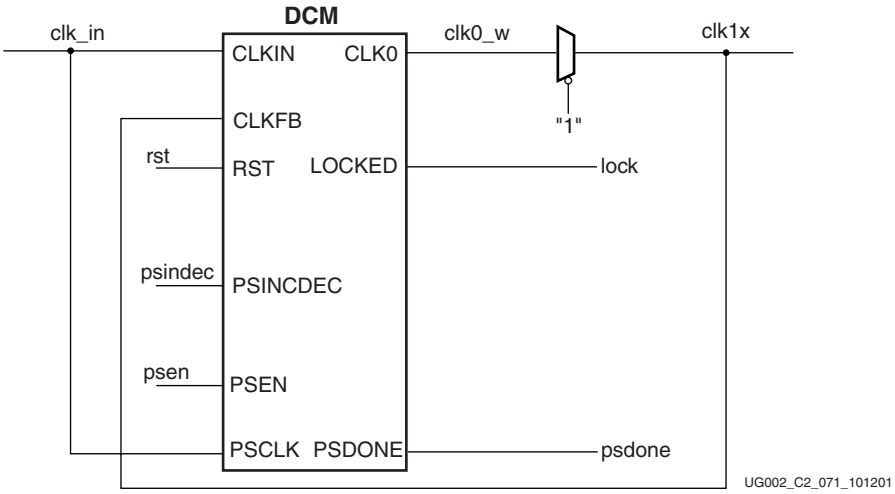


Figure 2-35: **BUFG_PHASE_CLK0_SUBM**

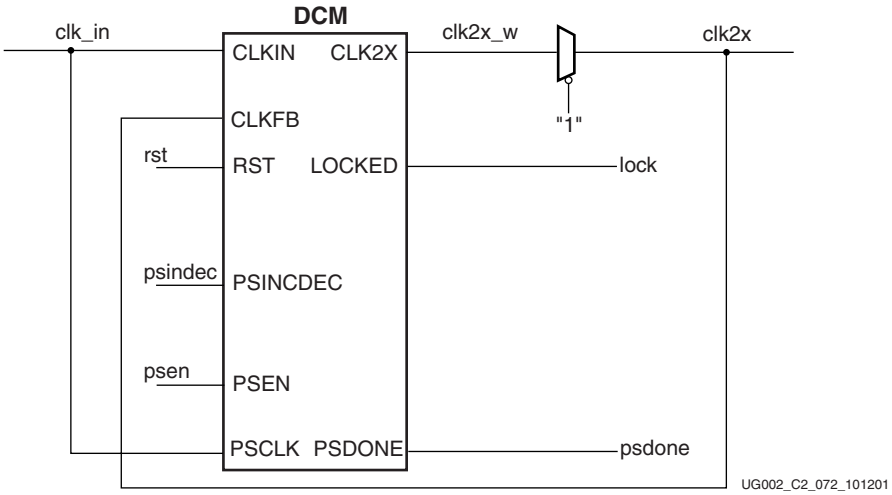


Figure 2-36: **BUFG_PHASE_CLK2X_SUBM**

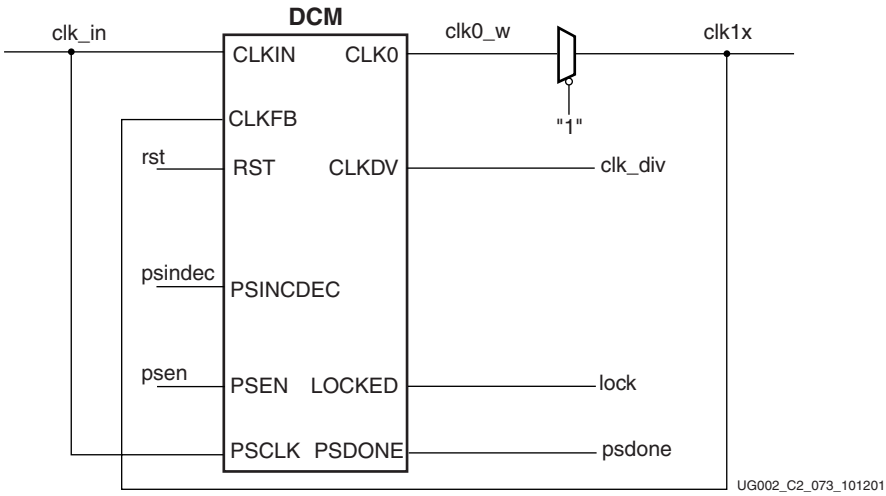


Figure 2-37: **BUFG_PHASE_CLKDV_SUBM**

VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available as examples (see "VHDL and Verilog Templates" on page 191) for all 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.

VHDL and Verilog Templates

The following submodules described in this section are available:

- BUFG_CLK0_SUBM
- BUFG_CLK2X_SUBM
- BUFG_CLK0_FB_SUBM
- BUFG_CLK2X_FB_SUBM
- BUFG_CLKDV_SUBM
- BUFG_DFS_SUBM
- BUFG_DFS_FB_SUBM
- BUFG_PHASE_CLKFX_FB_SUBM
- BUFG_PHASE_CLK0_SUBM
- BUFG_PHASE_CLK2X_SUBM
- BUFG_PHASE_CLKDV_SUBM

The corresponding submodules must be synthesized with the design. The BUFG_CLK0_SUBM submodule is provided in VHDL and Verilog as an example.

VHDL Template

```
-- Module: BUFG_CLK0_SUBM
-- Description: VHDL submodule
-- DCM with CLK0 deskew
-- Device: Virtex-II Family
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
--
entity BUFG_CLK0_SUBM is
  port (
        CLK_IN : in std_logic;
        RST    : in std_logic;
        CLK1X  : out std_logic;
        LOCK   : out std_logic
       );
end BUFG_CLK0_SUBM;
--
architecture BUFG_CLK0_SUBM_arch of BUFG_CLK0_SUBM is
-- Components Declarations:
component BUFG
  port (
        I : in std_logic;
        O : out std_logic
      );
end component;
component DCM
```

```
-- pragma translate_off
    generic (
             DLL_FREQUENCY_MODE : string := "LOW";
             DUTY_CYCLE_CORRECTION : boolean := TRUE;
             STARTUP_WAIT : boolean := FALSE
            );
-- pragma translate_on
    port ( CLKIN     : in  std_logic;
           CLKFB     : in  std_logic;
           DSSEN     : in  std_logic;
           PSINCDEC  : in  std_logic;
           PSEN      : in  std_logic;
           PSCLK     : in  std_logic;
           RST       : in  std_logic;
           CLK0      : out std_logic;
           CLK90     : out std_logic;
           CLK180    : out std_logic;
           CLK270    : out std_logic;
           CLK2X     : out std_logic;
           CLK2X180  : out std_logic;
           CLKDV     : out std_logic;
           CLKFX     : out std_logic;
           CLKFX180  : out std_logic;
           LOCKED    : out std_logic;
           PSDONE    : out std_logic;
           STATUS    : out std_logic_vector(7 downto 0)
          );
end component;
-- Attributes
attribute DLL_FREQUENCY_MODE : string;
attribute DUTY_CYCLE_CORRECTION : string;
attribute STARTUP_WAIT : string;
attribute DLL_FREQUENCY_MODE of U_DCM: label is "LOW";
attribute DUTY_CYCLE_CORRECTION of U_DCM: label is "TRUE";
attribute STARTUP_WAIT of U_DCM: label is "FALSE";
-- Signal Declarations:
signal GND : std_logic;
signal CLK0_W: std_logic;
signal CLK1X_W: std_logic;
begin
GND <= '0';
CLK1X <= CLK1X_W;
-- DCM Instantiation
U_DCM: DCM
  port map (
            CLKIN =>    CLK_IN,
            CLKFB =>    CLK1X_W,
            DSSEN =>    GND,
            PSINCDEC => GND,
            PSEN =>     GND,
            PSCLK =>    GND,
            RST =>      RST,
            CLK0 =>     CLK0_W,
            LOCKED =>   LOCK
    );
-- BUFG Instantiation
U_BUFG: BUFG
  port map  (
       I => CLK0_W,
       O => CLK1X_W
            );
end BUFG_CLK0_SUBM_arch;
```

Verilog Template

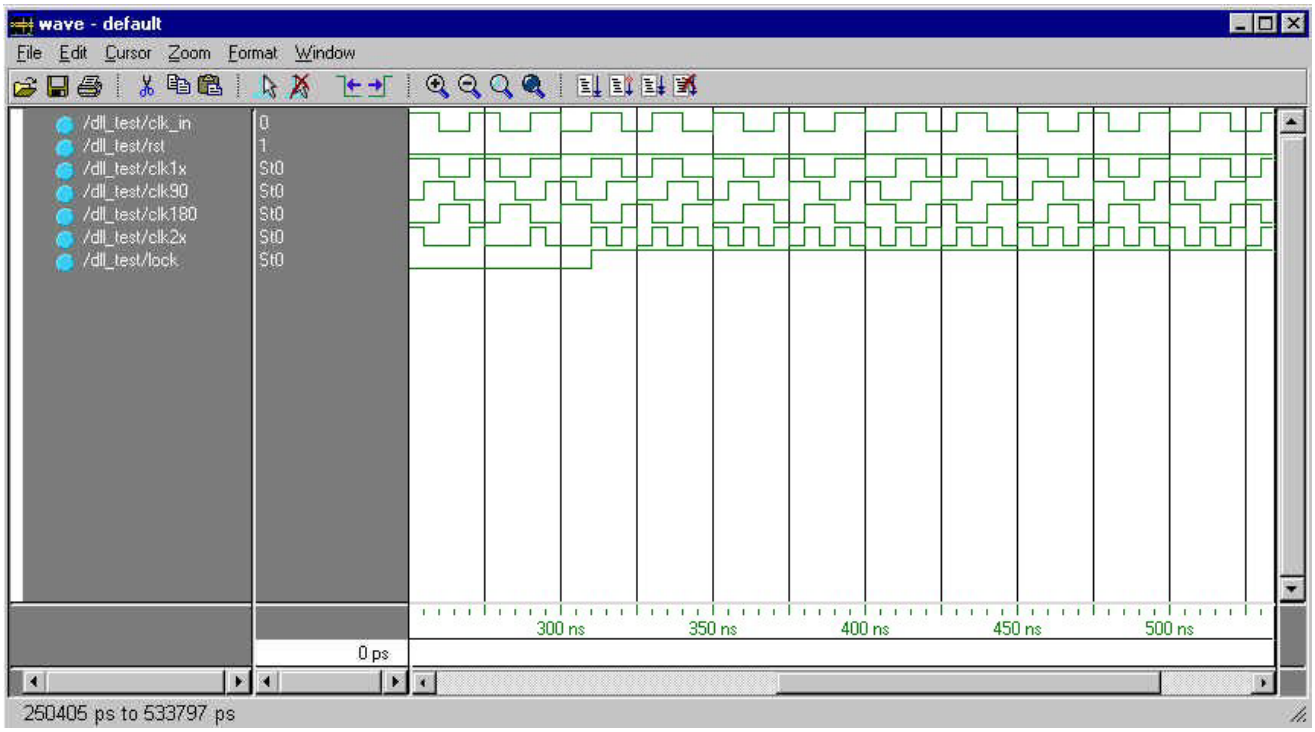
```
// Module:      BUFG_CLK0_SUBM
// Description: Verilog Submodule
//  DCM with CLK0 deskew
//
// Device: Virtex-II Family
//-----------------------------------------------------------------------

module BUFG_CKL0_SUBM (
                        CLK_IN,
                        RST,
                        CLK1X,
                        LOCK
                       );

    input CLK_IN;
    input RST;

    output CLK1X;
    output LOCK;

    wire CLK0_W;
    wire GND;

    assign GND = 1'b0;

//BUFG Instantiation
//
BUFG  U_BUFG
            (.I(CLK0_W),
             .O(CLK1X)
            );

// Attributes for functional simulation//
// synopsys translate_off
       defparam U_DCM.DLL_FREQUENCY_MODE = "LOW";
       defparam U_DCM.DUTY_CYCLE_CORRECTION = "TRUE";
       defparam U_DCM.STARTUP_WAIT = "FALSE";
// synopsys translate_on

// Instantiate the DCM primitive//
  DCM U_DCM (
                    .CLKFB(CLK1X),
                    .CLKIN(CLK_IN),
                    .DSSEN(GND),
                    .PSCLK(GND),
       .PSEN(GND),
                    .PSINCDEC(GND),
                    .RST(RST),
                    .CLK0(CLK0_W),
       .LOCKED(LOCK)
                   );

// synthesis attribute declarations
  /* synopsys attribute

  DLL_FREQUENCY_MODE "LOW"
  DUTY_CYCLE_CORRECTION "TRUE"
  STARTUP_WAIT "FALSE"
  */
endmodule
```

DCM Waveforms

The DCM waveforms shown below are the results of functional simulation using Model Technology's ModelSim EE/Plus 5.3a_p1 simulator. Note that the time scale for these simulations were set to 1ns/1ps. It is important to set the unused inputs of the DCM to logic 0 and to set the attribute values to the correct data types. For example, the PHASE_SHIFT, CLKFX_DIVIDE, and CLKFX_MULTIPLY attributes are integers and should be set to values as shown.

```
defparam U_DCM.DFS_FREQUENCY_MODE = "LOW";
defparam U_DCM.CLKFX_DIVIDE = 1; (this value's range is specified under
Frequency Synthesis in the Virtex-II Data Sheet)
defparam U_DCM.CLKFX_MULTIPLY = 4; (this value's range is specified
under Frequency Synthesis in the Virtex-II Data Sheet)
defparam U_DCM.CLKOUT_PHASE_SHIFT = "FIXED";
defparam U_DCM.PHASE_SHIFT = 150;       (Any value from 1 to 255)
defparam U_DCM.STARTUP_WAIT = "FALSE";
```

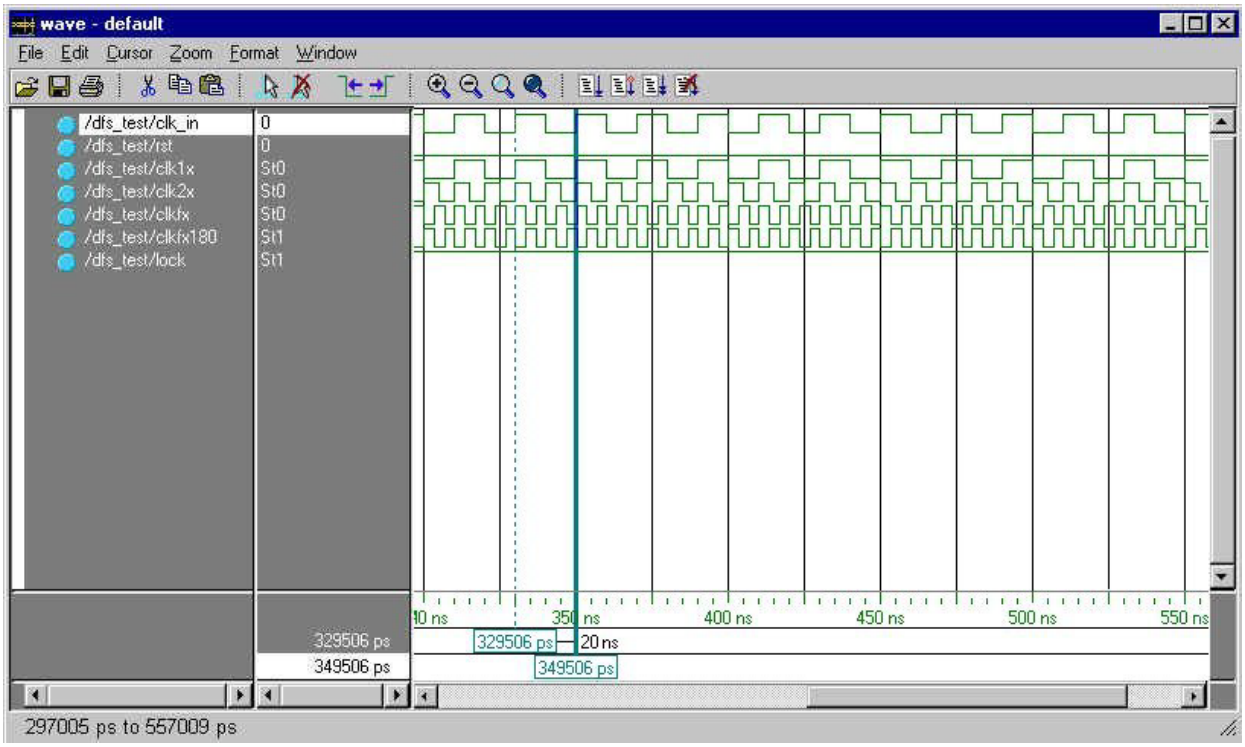
The input clock, 'clk_in' (CLKIN input of DCM) in all these waveforms is 50 MHz. The DCM_DLL waveforms in Figure 2-38 shows four DCM outputs, namely, clk1x (CLK0 output of DCM), clk2x (CLK2X output of DCM), clk90 (CLK90 output of DCM), and clk180 (CLK180 output of DCM).



ug002_c2_095_113000

Figure 2-38: **DCM_DLL Waveforms**

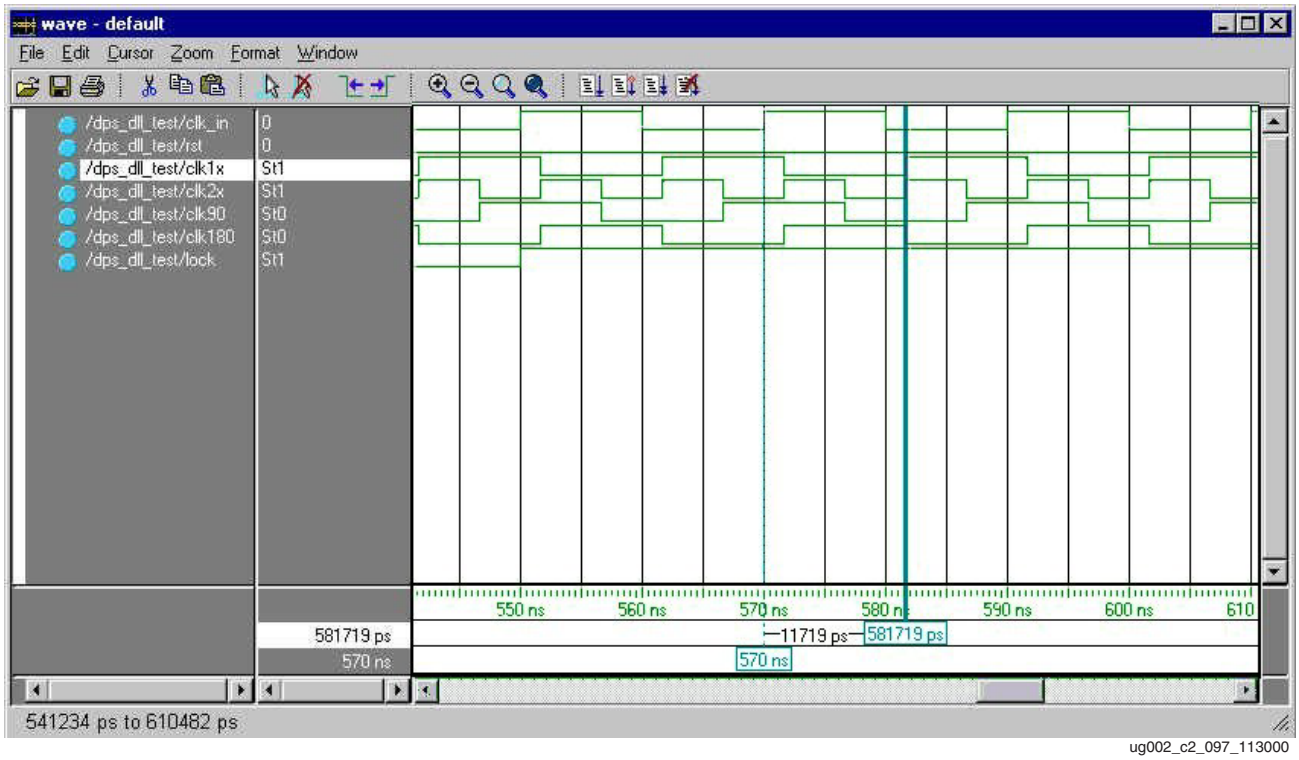
The DCM_DFS Waveforms in Figure 2-39 shows four DCM outputs namely, clk1x (CLK0 output of DCM), clk2x (CLK2X output of DCM), clkfx (CLKFX output of DCM), and clkfx180 (CLKFX180 output of DCM). In this case the attributes, CLKFX_DIVIDE = 1, and the CLKFX_MULTIPLY = 3.



ug002_c2_096_113000

Figure 2-39: **DCM_DFS Waveforms**

The DCM_DPS waveforms in Figure 2-40 shows four DCM outputs, namely, clk1x (CLK0 output of DCM), clk2x (CLK2X output of DCM), clk90 (CLK90 output of DCM), and clk180 (CLK180 output of DCM). In this case, the attribute PHASE_SHIFT = 150 which translates to a phase shift of (150 x 20 ns)/256 = 11.719 ns, where 20 ns is the clock period.



ug002_c2_097_113000

Figure 2-40: **DCM_DPS Waveforms**

The DCM_DPS_DFS waveforms in Figure 2-41 shows four DCM outputs namely, clk1x (CLK0 output of DCM), clk90 (CLK90 output of DCM), clkfx (CLKFX output of DCM), and clkfx180 (CLKFX180 output of DCM). In this case, the attributes, CLKFX_DIVIDE = 1, and the CLKFX_MULTIPLY = 4. The attribute, PHASE_SHIFT = 150 which translates to a phase shift of (150 x 20 ns)/256 = 11.719 ns, where 20 ns is the clock period.

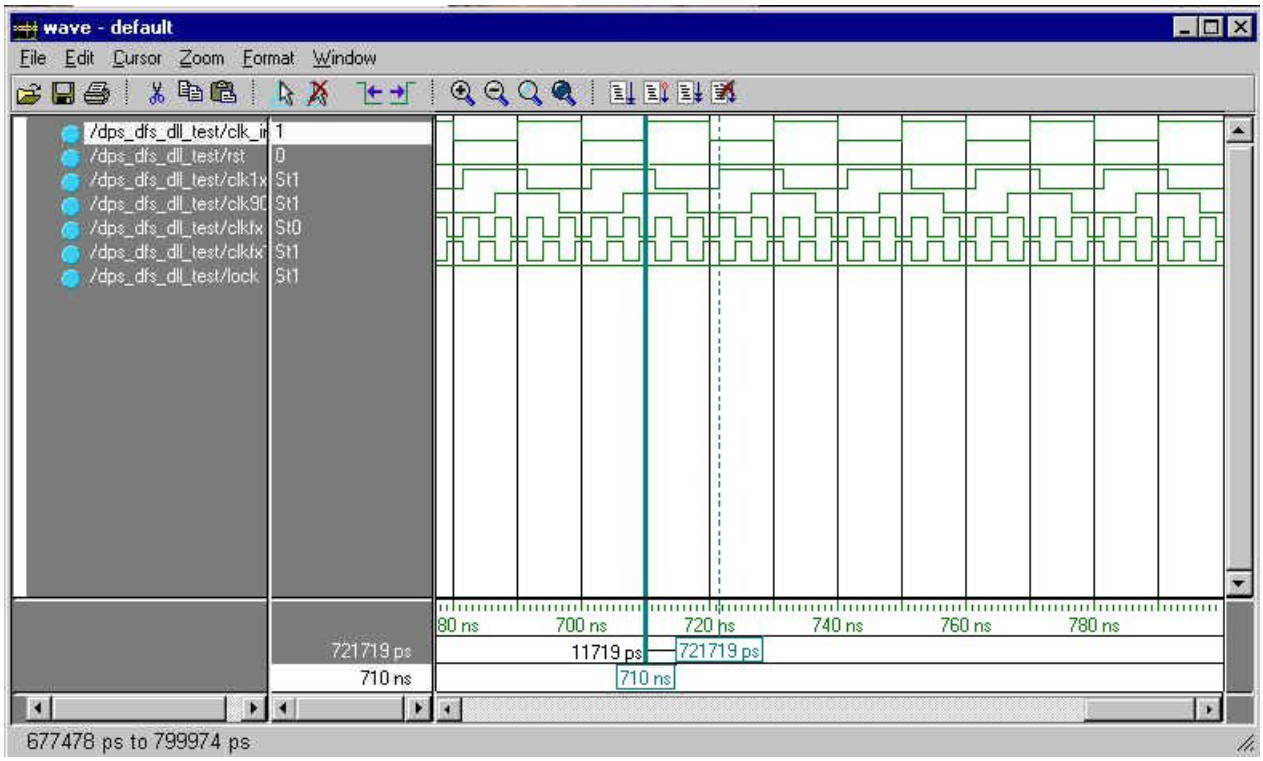


Figure 2-41: **DCM_DPS_DFS Waveforms**

Using Block SelectRAM™ Memory

Introduction

In addition to distributed SelectRAM memory, Virtex-II devices feature a large number of 18 Kb block SelectRAM memories. The block SelectRAM memory is a True Dual-Port™ RAM, offering fast, discrete, and large blocks of memory in the device. The memory is organized in columns, and the total amount of block SelectRAM memory depends on the size of the Virtex-II device. The 18 Kb blocks are cascadable to enable a deeper and wider memory implementation, with a minimal timing penalty incurred through specialized routing resources.

Embedded dual- or single-port RAM modules, ROM modules, synchronous and asynchronous FIFOs, and data width converters are easily implemented using the Xilinx CORE Generator "Block Memory" modules. Asynchronous FIFOs can also be generated using the CORE Generator Asynchronous FIFO module. Starting with IP Update #3, the designer can also generate synchronous FIFOs using Block Memory.

Synchronous Dual-Port and Single-Port RAM

Data Flow

The 18Kb block SelectRAM 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.

Data can be written to either port and can be read from the same or the other port. Each port is synchronous, with its own clock, clock enable, and write enable. Note that the read operation is also synchronous and requires a clock edge.

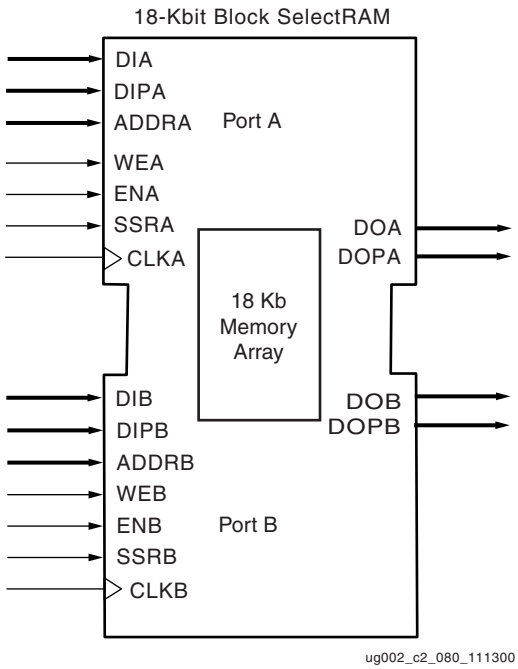


Figure 2-42: **Dual-Port Data Flows**

As described below, there are three options for the behavior of the data output during a write operation on its port. There is no dedicated monitor to arbitrate the result 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.

Operating Modes

To maximize utilization of the True Dual-Port memory at each clock edge, the block SelectRAM memory supports three different write modes for each port. The "read during write" mode offers the flexibility of using the data output bus during a write operation on the same port. Output behavior is determined by the configuration. This choice increases the efficiency of block SelectRAM memory at each clock cycle and allows designs that use maximum bandwidth.

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 interval passes.

Write Operations

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.

Three different modes are used to determine data available on the output latches after a write clock edge.

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 2-43.

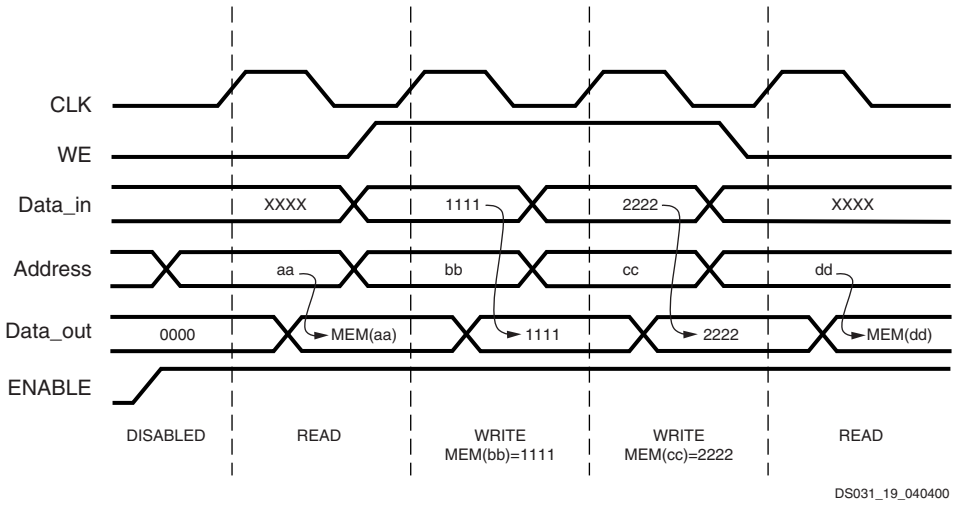


Figure 2-43: **WRITE_FIRST Mode Waveforms**

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 2-44.

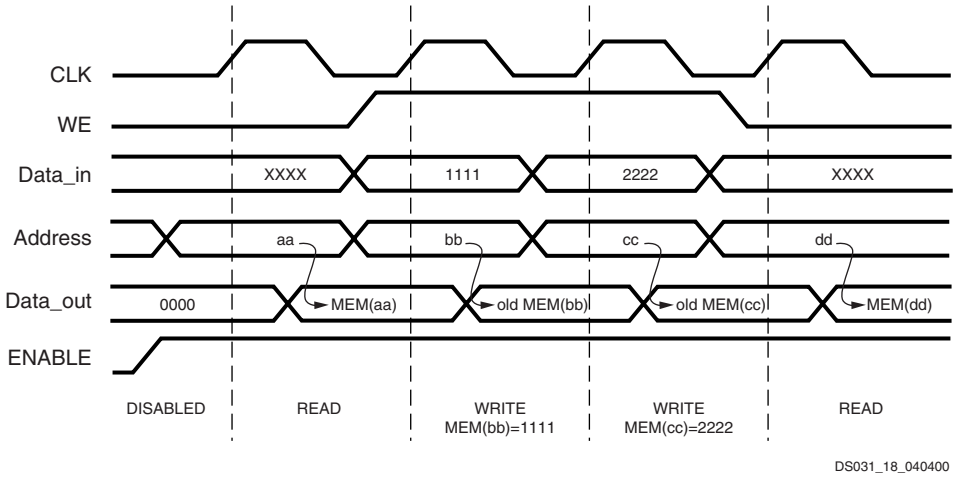


Figure 2-44: **READ_FIRST Mode Waveforms**

NO_CHANGE Mode

In NO_CHANGE mode, the output latches remain unchanged during a write operation. As shown in Figure 2-45, data output is still the last read data and is unaffected by a write operation on the same port.

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.

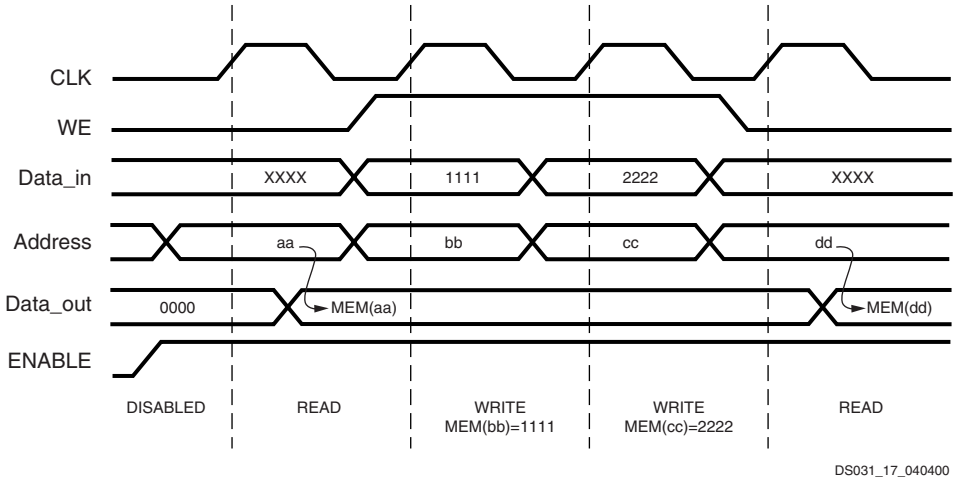


Figure 2-45: **NO_CHANGE Mode Waveforms**

Conflict Resolution

Virtex-II block SelectRAM memory is a True Dual-Port RAM that allows both ports to simultaneously access the same memory cell. When one port writes to a given memory cell, the other port must not address that memory cell (for a write or a read) within the clock-to-clock setup window. Figure 2-46 describes this asynchronous operation.

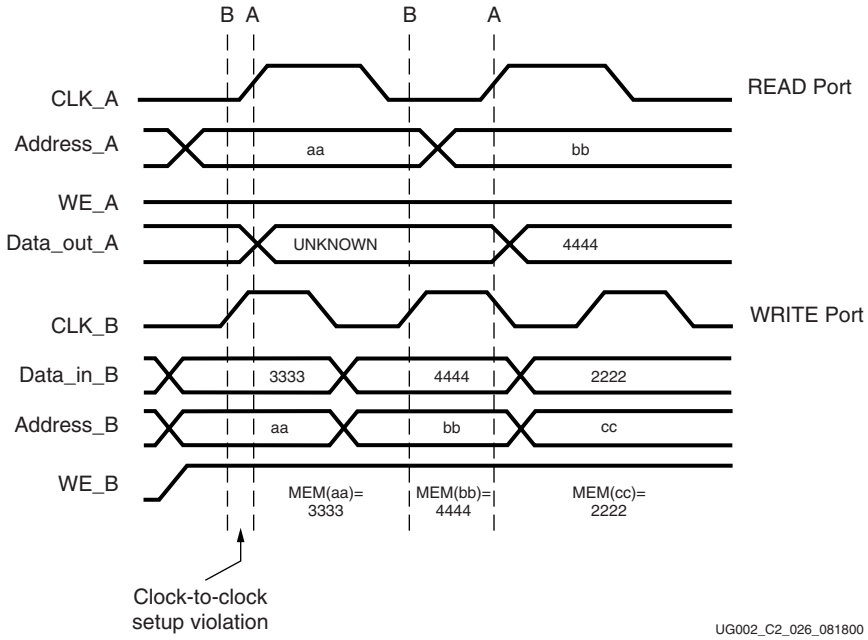


Figure 2-46: **READ-WRITE Conditions**

If port A and port B are configured with different widths, only the overlapping bits are invalid when conflicts occur.

Asynchronous Clocks

The first CLK_A clock edge violates the clock-to-clock setup parameter, because it occurs too soon after the last CLK_B clock edge. The write operation on port B is valid, and the read operation on port A is invalid.

At the second rising edge of the CLK_B pin, the write operation is valid. The memory location (bb) contains 4444. The second rising edge of CLK_A reads the new data at the same location (bb), which now contains 4444.

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

Synchronous Clocks

When both clocks are synchronous or identical, the result of simultaneous accesses from both ports to the same memory cell is best described in words:

- If both ports read simultaneously from the same memory cell:
  Both Data_out ports will have the same data.
- If both ports write simultaneously into the same memory cell:
  The data stored in that cell becomes invalid (unless both ports write identical data).
- If one port writes and the other one reads from the same memory cell:
  The write operation succeeds, and the write port's Data_out behaves as determined by the read output mode (write_first, read_first, or no_change).

  If the write port is in read_first mode, the read port's Data_out represents the previous content of the memory cell. If the write port is in write_first mode or in no_change mode, the read port's Data_out becomes invalid. Obviously, the read port's mode setting does not affect this operation.

Characteristics

- A write operation requires only one clock edge.
- A read operation requires only one clock edge.
- All inputs are registered with the port clock and have a setup-to-clock timing specification.
- All outputs have a read-through function or one of three read-during-write functions, depending on the state of the WE pin. The outputs relative to the port clock are available after the clock-to-out timing interval.
- Block SelectRAM cells are true synchronous RAM memories and do not have a combinatorial path from the address to the output.
- The ports are completely independent of each other (that is, clocking, control, address, read/write functions, initialization, and data width) without arbitration.
- Output ports are latched with a self-timed circuit, guaranteeing glitch-free reads. The state of the output port does not change until the port executes another read or write operation.
- Data input and output signals are always busses; that is, in a 1-bit width configuration, the data input signal is DI[0] and the data output signal is DO[0].

Library Primitives

The input and output data busses are represented by two busses 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.

Figure 2-47 shows the generic dual-port block RAM primitive. DIA, DIPA, ADDRA, DOA, DOPA, and the corresponding signals on port B are busses.

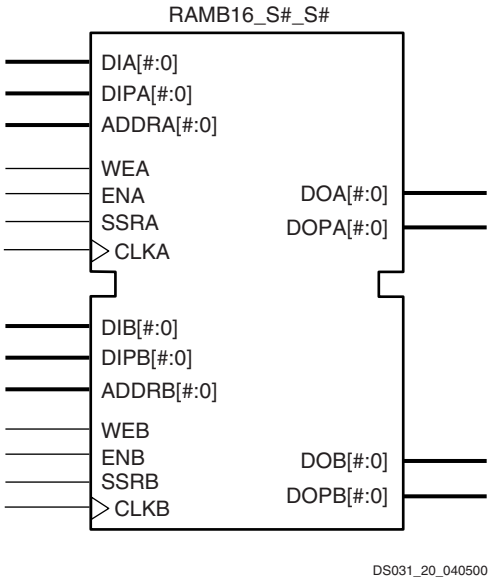


Figure 2-47: **Dual-Port Block RAM Primitive**

Table 2-9 lists the available dual-port primitives for synthesis and simulation.

Table 2-9: **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 2-48 shows the generic single-port block RAM primitive. DI, DIP, ADDR, DO, and DOP are busses.

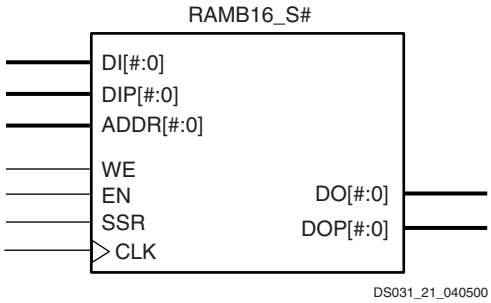


Figure 2-48: **Single-Port Block RAM Primitive**

Table 2-10 lists all of the available single-port primitives for synthesis and simulation.

Table 2-10: **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)

VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available as examples (see "VHDL and Verilog Templates" on page 207).

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

SelectRAM_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.

Port Signals

Each block SelectRAM port operates independently of the other while accessing the same set of 18K-bit 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 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]

Both EN and WE are active when the contents of the data input bus is written to memory at the address pointed to by the address bus. 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).

Set/Reset - SSR[A|B]

The SSR pin forces the data output latches to contain the value "SRVAL" (see "Attributes" on page 205). 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 memory cells and does not disturb write operations on the other port. Like 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).

Address Bus - ADDR[A|B]<#:0>

The address bus selects the memory cells for read or write. The width of the port determines the required address bus width, as shown in Table 2-11.

Table 2-11: **Port Aspect Ratio**

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

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

Data-in busses provide the new data value to be written into RAM. The regular data-in bus (DI) and the parity data-in bus (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 2-11.

Data-Out Busses - DO[A|B]<#:0> & DOP[A|B]<#:0>

Data-out busses 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 busses reflect either the data-in busses or the stored value before write. During a write operation in NO_CHANGE mode, data-out busses 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 2-11.

Inverting Control Pins

For each port, the four control pins (CLK, EN, WE, 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.

Unused Inputs

Non-connected Data and/or address inputs should be connected to logic “1”.

GSR

The global set/reset (GSR) signal of a Virtex-II device is an asynchronous global signal that is active at the end of device configuration. The GSR can also restore the initial Virtex-II state at any time. The GSR signal initializes the output latches to the INIT, or to the INIT_A and INIT_B value (see “Attributes” on page 205). 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 SelectRAM primitive).

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 2-12 shows low-order address mapping for each port width.

Table 2-12: **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																													

Attributes

Content Initialization - INIT_xx

INIT_xx attributes define the initial memory contents. By default block SelectRAM 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 2-13.

Table 2-13: **Block SelectRAM Initialization Attributes**

Attribute	Memory Cell	
	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 busses (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 and behaves like a regular INIT_xx attribute. 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 2-14. 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 2-14. These attributes are hex-encoded bit vectors and the default value is 0.

Table 2-14: **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

Initialization in VHDL or Verilog Codes

Block SelectRAM memory structures can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the block SelectRAM 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 XC2V_RAMB_1_PORT block SelectRAM instantiation code examples (in VHDL and Verilog) illustrate these techniques (see "VHDL and Verilog Templates" on page 207).

Location Constraints

Block SelectRAM instances can have LOC properties attached to them to constrain placement. Block SelectRAM 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 SelectRAM location on the device.

Applications

Creating Larger RAM Structures

Block SelectRAM columns have specialized routing to allow cascading 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 program offers the designer a painless way to generate wider and deeper memory structures using multiple block SelectRAM instances. This program outputs VHDL or Verilog instantiation templates and simulation models, along with an EDIF file for inclusion in a design.

Multiple RAM Organizations

The flexibility of block SelectRAM memories allows designs with various types of RAM in addition to regular configurations. Application notes at **www.xilinx.com** describe some of these designs, with VHDL and Verilog reference designs included.

Virtex-II block SelectRAM can be used as follows:

- Two independent single-port RAM resources
- One 72-bit single-port RAM resource
- One triple-port (1 Read/Write and 2 Read ports) RAM resource

Application notes with VHDL and Verilog reference designs at **www.xilinx.com** also describe other implementations using block SelectRAM memory, such as:

- **xapp258** "FIFOs Using Virtex-II Block RAM"
- **xapp260** "Fast Read/Write CAM Solution"

VHDL and Verilog Templates

VHDL and Verilog templates are available for all single-port and dual-port primitives. The A and B numbers indicate the width of the ports.

The following are single-port templates:

- SelectRAM_A1
- SelectRAM_A2
- SelectRAM_A4
- SelectRAM_A9
- SelectRAM_A18
- SelectRAM_A36

The following are dual-port templates:

- SelectRAM_A1_B1
- SelectRAM_A1_B2
- SelectRAM_A1_B4
- SelectRAM_A1_B9
- SelectRAM_A1_B18
- SelectRAM_A1_B36
- SelectRAM_A2_B2
- SelectRAM_A2_B4
- SelectRAM_A2_B9
- SelectRAM_A2_B18
- SelectRAM_A2_B36
- SelectRAM_A4_B4

- SelectRAM_A4_B9
- SelectRAM_A4_B18
- SelectRAM_A4_B36
- SelectRAM_A9_B9
- SelectRAM_A9_B18
- SelectRAM_A9_B36
- SelectRAM_A18_B18
- SelectRAM_A18_B36
- SelectRAM_A36_B36

VHDL Template

As an example, the **XC2V_RAMB_1_PORT.vhd** file uses the SelectRAM_A36 template:

```
-- Module: XC2V_RAMB_1_PORT
-- Description: 18Kb Block SelectRAM example
-- Single Port 512 x 36 bits
-- Use template "SelectRAM_A36.vhd"
--
-- Device: Virtex-II Family
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--
-- Syntax for Synopsys FPGA Express
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
--
entity XC2V_RAMB_1_PORT is
   port (
    DATA_IN : in std_logic_vector (35 downto 0);
        ADDRESS   : in std_logic_vector (8 downto 0);
        ENABLE: in std_logic;
        WRITE_EN     : in std_logic;
        SET_RESET    : in std_logic;
        CLK    : in std_logic;
        DATA_OUT     : out std_logic_vector (35 downto 0)
       );
end XC2V_RAMB_1_PORT;
--
architecture XC2V_RAMB_1_PORT_arch of XC2V_RAMB_1_PORT is
--
-- Components Declarations:
--
component BUFG
  port (
  I : in std_logic;
  O : out std_logic
  );
end component;
--
-- Syntax for Synopsys FPGA Express
component RAMB16_S36
-- pragma translate_off
  generic (
-- "Read during Write" attribute for functional simulation
  WRITE_MODE : string := "READ_FIRST" ; -- WRITE_FIRST(default)/
READ_FIRST/ NO_CHANGE
```

```
-- Output value after configuration
  INIT : bit_vector(35 downto 0)  := X"000000000";
-- Output value if SSR active
  SRVAL : bit_vector(35 downto 0)  := X"012345678";
-- Plus bits initial content
       INITP_00 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
       INITP_01 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_02 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_03 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_04 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_05 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_06 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_07 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
-- Regular bits initial content
       INIT_00 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
       INIT_01 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       ... (cut)
       INIT_3E : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3F : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000"
  );
-- pragma translate_on
  port (
        DI     : in std_logic_vector (31 downto 0);
        DIP    : in std_logic_vector (3 downto 0);
        ADDR   : in std_logic_vector (8 downto 0);
        EN     : in STD_LOGIC;
        WE     : in STD_LOGIC;
        SSR    : in STD_LOGIC;
        CLK    : in STD_LOGIC;
        DO     : out std_logic_vector (31 downto 0);
        DOP    : out std_logic_vector (3 downto 0)
  );
end component;
--
-- Attribute Declarations:
attribute WRITE_MODE : string;
attribute INIT: string;
attribute SRVAL: string;
--
attribute INITP_00: string;
attribute INITP_01: string;
attribute INITP_02: string;
attribute INITP_03: string;
attribute INITP_04: string;
attribute INITP_05: string;
attribute INITP_06: string;
attribute INITP_07: string;
--
```

```
attribute INIT_00: string;
attribute INIT_01: string;
attribute INIT_02: string;
… (cut)
attribute INIT_3E: string;
attribute INIT_3F: string;
--
-- Attribute "Read during Write mode" = WRITE_FIRST(default)/
READ_FIRST/ NO_CHANGE
attribute WRITE_MODE of U_RAMB16_S36: label is "READ_FIRST";
attribute INIT of U_RAMB16_S36: label is "000000000";
attribute SRVAL of U_RAMB16_S36: label is "012345678";
--
-- RAMB16 memory initialization for Alliance
-- Default value is "0" / Partial initialization strings are padded
-- with zeros to the left
attribute INITP_00 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
attribute INITP_01 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_02 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_03 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_04 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_05 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_06 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_07 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
--
attribute INIT_00 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
attribute INIT_01 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_02 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
... (cut)
attribute INIT_3E of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_3F of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
--
--
-- Signal Declarations:
--
-- signal VCC : std_logic;
-- signal GND : std_logic;
signal CLK_BUFG: std_logic;
signal INV_SET_RESET : std_logic;
--
begin
-- VCC <= '1';
-- GND <= '0';
--
-- Instantiate the clock Buffer
U_BUFG: BUFG
  port map (
  I => CLK,
  O => CLK_BUFG
```

```
  );
--
-- Use of the free inverter on SSR pin
INV_SET_RESET <= NOT SET_RESET;

-- Block SelectRAM Instantiation
U_RAMB16_S36: RAMB16_S36
  port map (
        DI     => DATA_IN (31 downto 0), -- insert 32 bits data-in bus
(<31 downto 0>)
        DIP    => DATA_IN (35 downto 32), -- insert 4 bits parity data-
in bus (or <35 downto 32>)
       ADDR   => ADDRESS (8 downto 0), -- insert 9 bits address bus
         EN     => ENABLE, -- insert enable signal
         WE     => WRITE_EN, -- insert write enable signal
         SSR    => INV_SET_RESET, -- insert set/reset signal
         CLK    => CLK_BUFG, -- insert clock signal
        DO     => DATA_OUT (31 downto 0), -- insert 32 bits data-out bus
(<31 downto 0>)
        DOP    => DATA_OUT (35 downto 32)  -- insert 4 bits parity data-
out bus (or <35 downto 32>)
  );
--
end XC2V_RAMB_1_PORT_arch;
----------------------------------------------------------------------
```

Verilog Template

```
// Module: XC2V_RAMB_1_PORT
// Description: 18Kb Block SelectRAM-II example
//  Single Port 512 x 36 bits
//  Use template "SelectRAM_A36.v"
//
// Device: Virtex-II Family
//----------------------------------------------------------------------

module XC2V_RAMB_1_PORT (CLK, SET_RESET, ENABLE, WRITE_EN, ADDRESS,
DATA_IN, DATA_OUT);


input CLK, SET_RESET, ENABLE, WRITE_EN;
input [35:0] DATA_IN;
input [8:0] ADDRESS;
output [35:0] DATA_OUT;

wire CLK_BUFG, INV_SET_RESET;

//Use of the free inverter on SSR pin
assign INV_SET_RESET = ~SET_RESET;


// initialize block ram for simulation
// synopsys translate_off
defparam
  //"Read during Write" attribute for functional simulation
  U_RAMB16_S36.WRITE_MODE = "READ_FIRST",   //WRITE_FIRST(default)/
READ_FIRST/ NO_CHANGE
  //Output value after configuration
  U_RAMB16_S36.INIT = 36'h000000000,
  //Output value if SSR active
  U_RAMB16_S36.SRVAL = 36'h012345678,
```

```
  //Plus bits initial content
  U_RAMB16_S36.INITP_00 =
256'h0123456789ABCDEF0000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_01 =
256'h0000000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_02 =
256'h0000000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_03 =
256'h0000000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_04 =
256'h0000000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_05 =
256'h0000000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_06 =
256'h0000000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_07 =
256'h0000000000000000000000000000000000000000000000000000000000000000000,

  //Regular bits initial content
  U_RAMB16_S36.INIT_00 =
256'h0123456789ABCDEF0000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INIT_01 =
256'h0000000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INIT_02 =
256'h0000000000000000000000000000000000000000000000000000000000000000000,
...<cut>
  U_RAMB16_S36.INIT_3E =
256'h0000000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INIT_3F =
256'h0000000000000000000000000000000000000000000000000000000000000000000;
 // synopsys translate_on

//Instantiate the clock Buffer
BUFG U_BUFG (  .I(CLK), .O(CLK_BUFG));

//Block SelectRAM Instantiation
RAMB16_S36 U_RAMB16_S36 (   .DI(DATA_IN[31:0]),
          .DIP(DATA_IN-PARITY[35:32]),
          .ADDR(ADDRESS),
          .EN(ENABLE),
          .WE(WRITE_EN),
          .SSR(INV_SET_RESET),
          .CLK(CLK_BUFG),
          .DO(DATA_OUT[31:0]),
          .DOP(DATA_OUT-PARITY[35:32]));

// synthesis attribute declarations
  /* synopsys attribute

  WRITE_MODE "READ_FIRST"
  INIT "000000000"
  SRVAL "012345678"

  INITP_00
"0123456789ABCDEF0000000000000000000000000000000000000000000000000000"
  INITP_01
"0000000000000000000000000000000000000000000000000000000000000000000"
  INITP_02
"0000000000000000000000000000000000000000000000000000000000000000000"
  INITP_03
"0000000000000000000000000000000000000000000000000000000000000000000"
```

```
  INITP_04
"0000000000000000000000000000000000000000000000000000000000000000"
  INITP_05
"0000000000000000000000000000000000000000000000000000000000000000"
  INITP_06
"0000000000000000000000000000000000000000000000000000000000000000"
  INITP_07
"0000000000000000000000000000000000000000000000000000000000000000"

  INIT_00
"0123456789ABCDEF0000000000000000000000000000000000000000000000000"
  INIT_01
"0000000000000000000000000000000000000000000000000000000000000000"
  INIT_02
"0000000000000000000000000000000000000000000000000000000000000000"
...<cut>
  INIT_3E
"0000000000000000000000000000000000000000000000000000000000000000"
  INIT_3F
"0000000000000000000000000000000000000000000000000000000000000000"
  */

endmodule
```

Using Distributed SelectRAM Memory

Introduction

In addition to 18Kb SelectRAM blocks, Virtex-II devices feature distributed SelectRAM modules. Each function generator or LUT of a CLB resource can implement a 16 x 1-bit synchronous RAM resource. Distributed SelectRAM memory writes synchronously and reads asynchronously. However, a synchronous read can be implemented using the register that is available in the same slice. This 16 x 1-bit RAM is cascadable for a deeper and/or wider memory implementation, with a minimal timing penalty incurred through specialized logic resources.

Distributed SelectRAM modules up to a size of 128 x 1 are available as primitives. Two 16 x 1 RAM resources can be combined to form a dual-port 16 x 1 RAM with one dedicated read/write port and a second read-only port. One port writes into both 16 x1 RAMs simultaneously, but the second port reads independently.

This section provides generic VHDL and Verilog reference code examples implementing *n*-bit-wide single-port and dual-port distributed SelectRAM memory.

Distributed SelectRAM memory enables many high-speed applications that require relatively small embedded RAM blocks, such as FIFOs, which are close to the logic that uses them.

Virtex-II Distributed SelectRAM memories can be generated using the CORE Generator Distributed Memory module (V2.0 or later). The user can also generate Distributed RAM-based Asynchronous and Synchronous FIFOs using the CORE Generator.

Single-Port and Dual-Port RAM

Data Flow

Distributed SelectRAM memory supports the following:

- Single-port RAM with synchronous write and asynchronous read
- Dual-port RAM with one synchronous write and two asynchronous read ports

As illustrated in the Figure 2-49, the dual port has one read/write port and an independent read port.

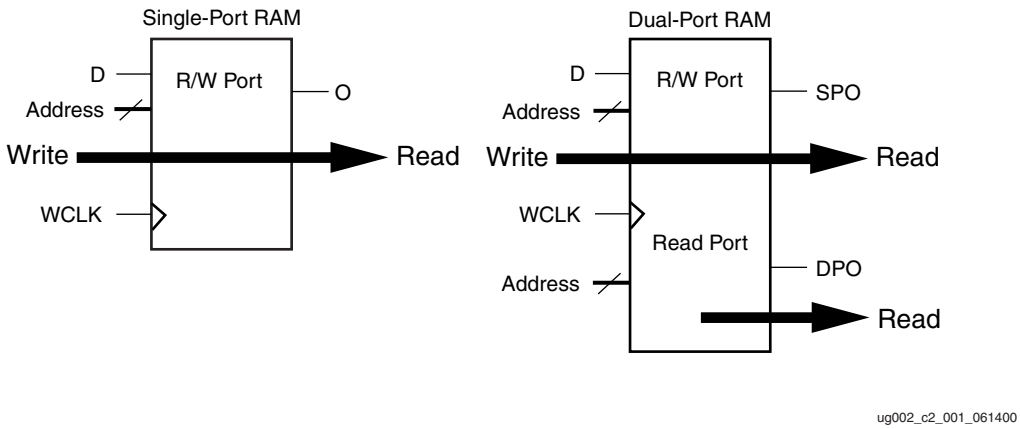


Figure 2-49: **Single-Port and Dual-Port Distributed SelectRAM**

Any read/write operation can occur simultaneously with and independently of a read operation on the other port.

Write Operations

The write operation is a single clock-edge operation, with a write enable that is active High by default. When the write enable is Low, no data is written into the RAM. When the write enable is High, the clock edge latches the write address and writes the data on D into the RAM.

Read Operation

The read operation is a combinatorial operation. The address port (single or dual port) is asynchronous with an access time equivalent to the logic delay.

Read During Write

When new data is synchronously written, the output reflects the data in the memory cell addressed (transparent mode). The timing diagram in Figure 2-50 illustrates a write operation, with the previous data read on the output port, before the clock edge and then the new data.

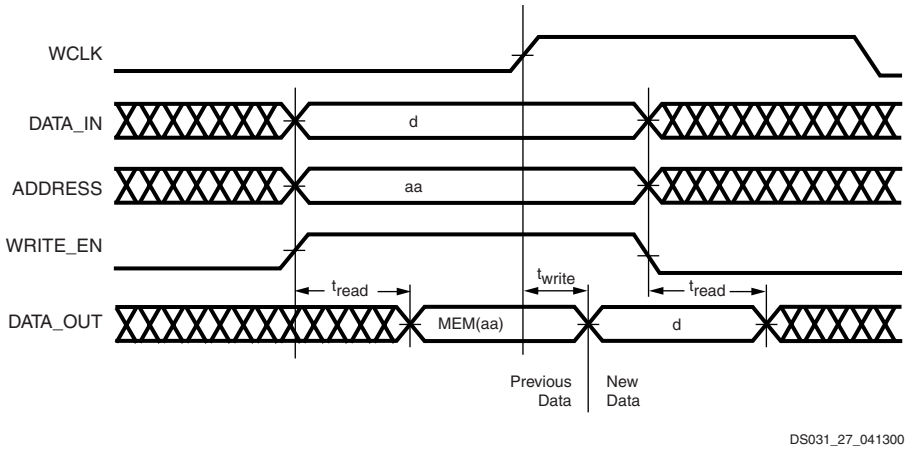


Figure 2-50: **Write Timing Diagram**

Characteristics

- A write operation requires only one clock edge.
- A read operation requires only the logic access time.
- Outputs are asynchronous and dependent only on the logic delay.
- Data and address inputs are latched with the write clock and have a setup-to-clock timing specification. There is no hold time requirement.
- For dual-port RAM, one address is the write and read address, the other address is an independent read address.

Library Primitives

Seven library primitives from 16 x 1-bit to 128 x 1-bit are available. Four primitives are single-port RAM and three primitives are True Dual-Port RAM, as shown in Table 2-15.

Table 2-15: **Single-Port and Dual-Port Distributed SelectRAM**

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, A3, A2, A1, A0
RAM128X1S	128 bits	single-port	A6, A4, A3, A2, A1, A0
RAM16X1D	16 bits	dual-port	A3, A2, A1, A0
RAM32X1D	32 bits	dual-port	A4, A3, A2, A1, A0
RAM64X1D	64 bits	dual-port	A5, A4, A3, A2, A1, A0

The input and output data are 1-bit wide. However, several distributed SelectRAM memories can be used to implement wide memory blocks.

Figure 2-51 shows generic single-port and dual-port distributed SelectRAM primitives. The A and DPRA signals are address busses.

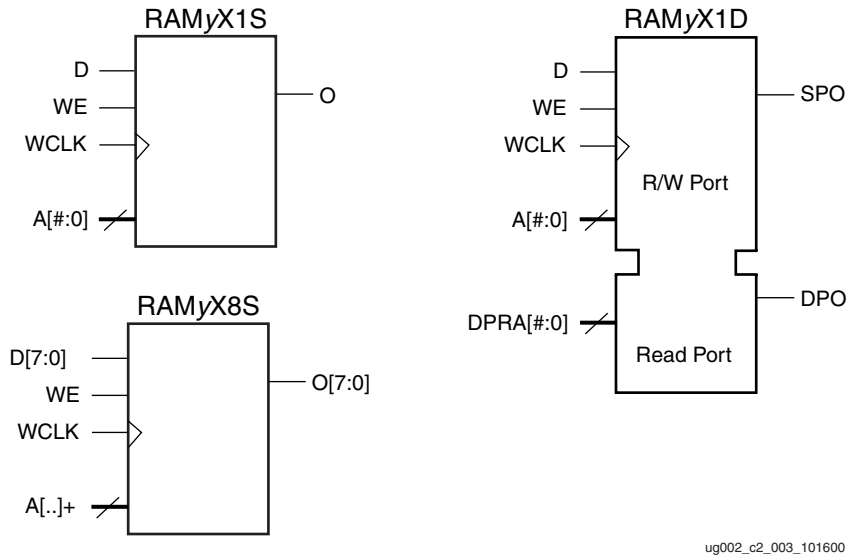


Figure 2-51: **Single-Port and Dual-Port Distributed SelectRAM Primitive**

As shown in Table 2-16, wider library primitives are available for 2-bit, 4-bit, and 8-bit RAM.

Table 2-16: **Wider Library 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
RAM64X2S	64 x 2-bit	D1, D0	A5, A4, A3, A2, A1, A0	O1, O0
RAM16X4S	16 x 4-bit	D3, D2, D1, D0	A3, A2, A1, A0	O3, O2, O1, O0
RAM32X4S	32 x 4-bit	D3, D2, D1, D0	A4,A3, A2, A1, A0	O3, O2, O1, O0
RAM16X8S	16 x 8-bit	D <7:0>	A3, A2, A1, A0	O <7:0>
RAM32X8S	32 x 8-bit	D <7:0>	A4,A3, A2, A1, A0	O <7:0>

VHDL and Verilog Instantiation

VHDL and Verilog instantiations templates are available as examples (see "VHDL and Verilog Templates" on page 221).

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 SelectRAM_*x*S templates (with *x* = 16, 32, 64, or 128) are single-port modules and instantiate the corresponding RAM*x*X1S primitive.

SelectRAM_*x*D templates (with *x* = 16, 32, or 64) are dual-port modules and instantiate the corresponding RAM*x*X1D primitive.

Ports Signals

Each distributed SelectRAM 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, A6)

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.

GSR

The global set/reset (GSR) signal does not affect distributed SelectRAM modules.

Attributes

Content Initialization - INIT

With the INIT attributes, users can define the initial memory contents after configuration. By default distributed SelectRAM memory 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 2-17 shows the length of the INIT attribute for each primitive.

Table 2-17: **INIT Attributes Length**

Primitive	Template	INIT Attribute Length
RAM16X1S	SelectRAM_16S	4 digits
RAM32X1S	SelectRAM_32S	8 digits
RAM64X1S	SelectRAM_64S	16 digits
RAM128X1S	SelectRAM_128S	32 digits
RAM16X1D	SelectRAM_16S	4 digits
RAM32X1D	SelectRAM_32S	8 digits
RAM64X1D	SelectRAM_64S	16 digits

Initialization in VHDL or Verilog Codes

Distributed SelectRAM memory structures can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the distributed SelectRAM 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 SelectRAM instantiation templates (in VHDL and Verilog) illustrate these techniques (see "VHDL and Verilog Templates" on page 221).

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 below: S

Slice S3	Slice S2	Slice S1	Slice S0
X1Y1	X1Y0	X0Y1	X0Y0

Distributed SelectRAM instances can have LOC properties attached to them to constrain placement. The RAM16X1S primitive fits in any LUT of slices S0 or S1.

For example, the instance `U_RAM16` is placed in slice X0Y0 with the following LOC properties:

```
INST "U_RAM16" LOC = "SLICE_X0Y0";
```

The RAM16X1D primitive occupies half of two slices, as shown in Figure 2-52. The first slice (output SPO) implements the read/write port with the same address A[3:0] for read

and write. The second slice implements the second read port with the address DPRA[3:0] and is written simultaneously with the first slice to the address A[3:0].

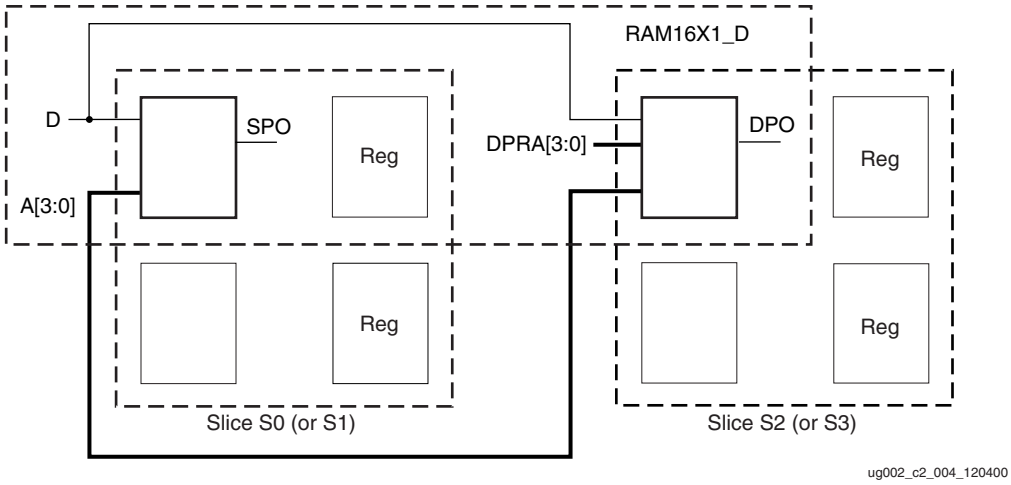


Figure 2-52: **RAM16X1_D Placement**

In the same CLB module, the dual-port RAM16X1_D either occupies half of slices S0 (X0Y0) and S2 (X1Y0), or half of slices S1 (X0Y1) and S3 (X1Y1).

If a dual-port 16 x 2-bit module is built, the two RAM16X1_D primitives occupy two slices, as long as they share the same clock and write enable, as illustrated in Figure 2-53.

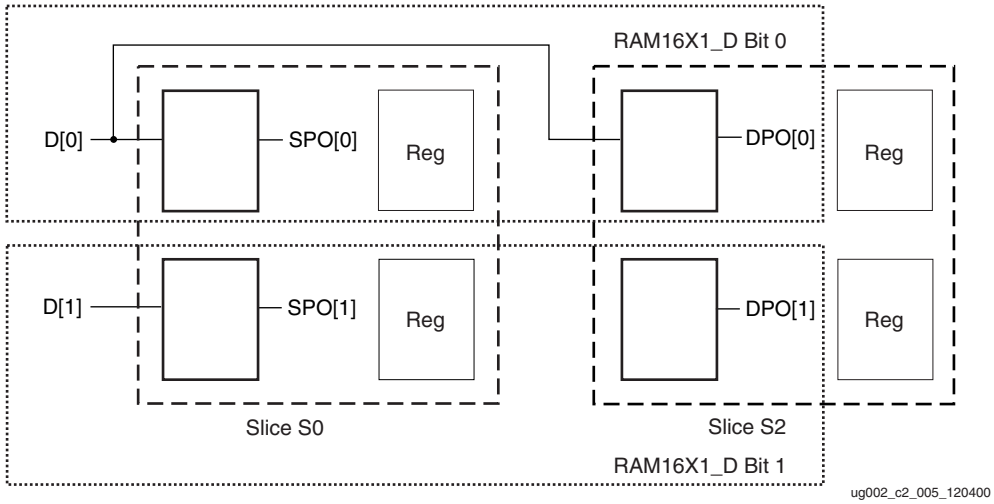


Figure 2-53: **Two RAM16X1_D Placement**

A RAM32X1S primitive fits in one slice, as shown in Figure 2-54.

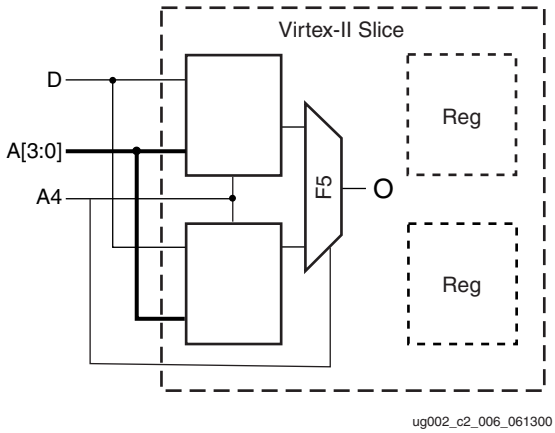


Figure 2-54: **RAM32X1_S Placement**

Following the same rules, a RAM32X1_D primitive fits in two slices, with one slice implementing the read/write port and the second slice implementing the second read port.

The RAM64X1_S primitive occupies two slices and the RAM64X1_D primitive occupies four slices (one CLB element), with two slices implementing the read/write port and two other slices implementing the second read port. The RAM64X1_S read path is built on the MUXF5 and MUXF6 multiplexers.

The RAM128X1_S primitive occupies four slices, equivalent to one CLB element.

Distributed SelectRAM placement locations use the slice location naming convention, allowing LOC properties to transfer easily from array to array.

Applications

Creating Larger RAM Structures

The memory compiler program generates wider and/or deeper memory structures using distributed SelectRAM instances. Along with an EDIF file for inclusion in a design, this program produces VHDL and Verilog instantiation templates and simulation models.

Table 2-18 shows the generic VHDL and Verilog distributed SelectRAM examples provided to implement *n*-bit-wide memories.

Table 2-18: **VHDL and Verilog Submodules**

Submodules	Primitive	Size	Type
XC2V_RAM16XN_S_SUBM	RAM16X1S	16 words x *n*-bit	single-port
XC2V_RAM32XN_S_SUBM	RAM32X1S	32 words x *n*-bit	single-port
XC2V_RAM64XN_S_SUBM	RAM64X1S	64 words x *n*-bit	single-port
XC2V_RAM128XN_S_SUBM	RAM128X1S	128 words x *n*-bit	single-port
XC2V_RAM16XN_D_SUBM	RAM16X1D	16 words x *n*-bit	dual-port
XC2V_RAM32XN_D_SUBM	RAM32X1D	32 words x *n*-bit	dual-port
XC2V_RAM64XN_D_SUBM	RAM64X1D	64 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, SelectRAM_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 following are single-port templates:

- SelectRAM_16S
- SelectRAM_32S
- SelectRAM_64S
- SelectRAM_128S

The following are dual-port templates:

- SelectRAM_16D
- SelectRAM_32D
- SelectRAM_64D

Templates for the SelectRAM_16S module are provided in VHDL and Verilog code as examples.

VHDL Template

```
--
-- Module: SelectRAM_16S
--
-- Description: VHDL instantiation template
--              Distributed SelectRAM
--              Single Port 16 x 1
--              can be used also for RAM16X1S_1
--
-- Device: Virtex-II Family
--
-------------------------------------------------------------------
--
-- Components Declarations:
--
component RAM16X1S
-- pragma translate_off
   generic (
-- RAM initialization ("0" by default) for functional simulation:
         INIT : bit_vector := X"0000"
         );
-- pragma translate_on
  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 SelectRAM 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: SelectRAM_16S
//
// Description: Verilog instantiation template
//              Distributed SelectRAM
//              Single Port 16 x 1
//              can be used also for RAM16X1S_1
//
// Device: Virtex-II Family
//
//-----------------------------------------------------------------------
//
//
// Syntax for Synopsys FPGA Express
// synopsys translate_off

  defparam

       //RAM initialization ("0" by default) for functional simulation:
  U_RAM16X1S.INIT = 16'h0000;
// synopsys translate_on


//Distributed SelectRAM 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
                  );

// synthesis attribute declarations
  /* synopsys attribute

  INIT "0000"
  */
```

Using Look-Up Tables as Shift Registers (SRLUTs)

Introduction

Virtex-II can configure any look-up table (LUT) as a 16-bit shift register without using the flip-flops available in each slice. Shift-in operations are synchronous with the clock, and output length is dynamically selectable. A separate dedicated output allows the cascading of any number of 16-bit shift registers to create whatever size shift register is needed. Each CLB resource can be configured using the 8 LUTs as a 128-bit shift register.

This section provides generic VHDL and Verilog submodules and reference code examples for implementing from 16-bit up to 128-bit shift registers. These submodules are built from 16-bit shift-register primitives and from dedicated MUXF5, MUXF6, MUXF7, and MUXF8 multiplexers.

These shift registers enable the development of efficient designs for applications that require delay or latency compensation. Shift registers are also useful in synchronous FIFO and content-addressable memory (CAM) designs. To quickly generate a Virtex-II shift register without using flip-flops (i.e., using the SRL16 element(s)), use the CORE Generator RAM-based Shift Register module.

Shift Register Operations

Data Flow

Each shift register (SRL16 primitive) supports:

- Synchronous shift-in
- Asynchronous 1-bit output when the address is changed dynamically
- Synchronous shift-out when the address is fixed

In addition, cascadable shift registers (SRLC16) support synchronous shift-out output of the last (16th) bit. This output has a dedicated connection to the input of the next SRLC16 inside the CLB resource. Two primitives are illustrated in Figure 2-55.

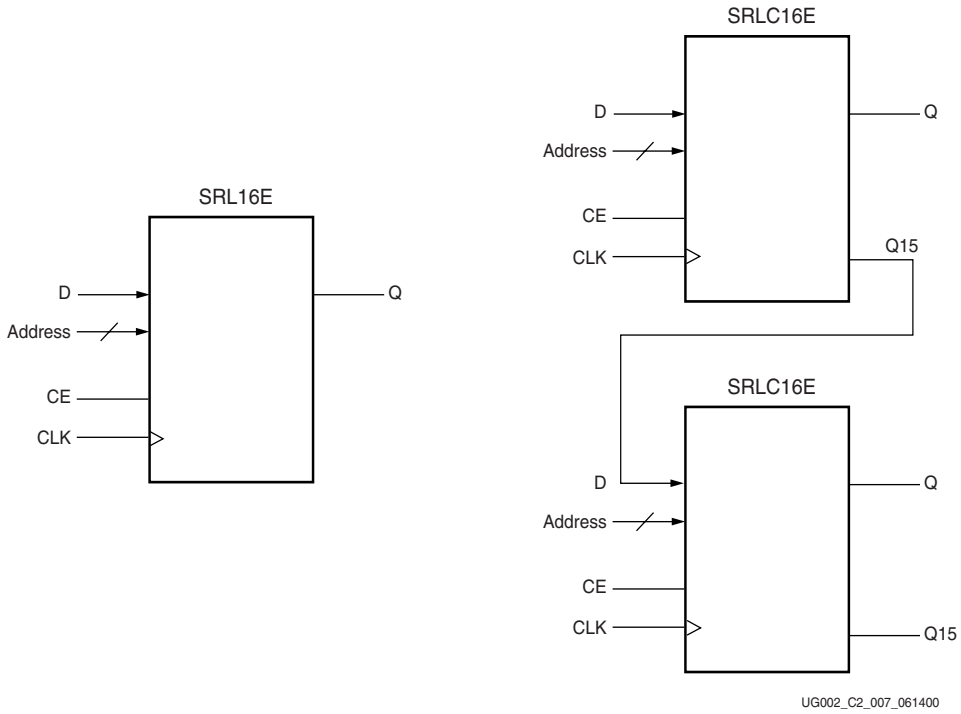


Figure 2-55: **Shift Register and Cascadable Shift Register**

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.

Figure 2-56 illustrates the shift and dynamic read operations.

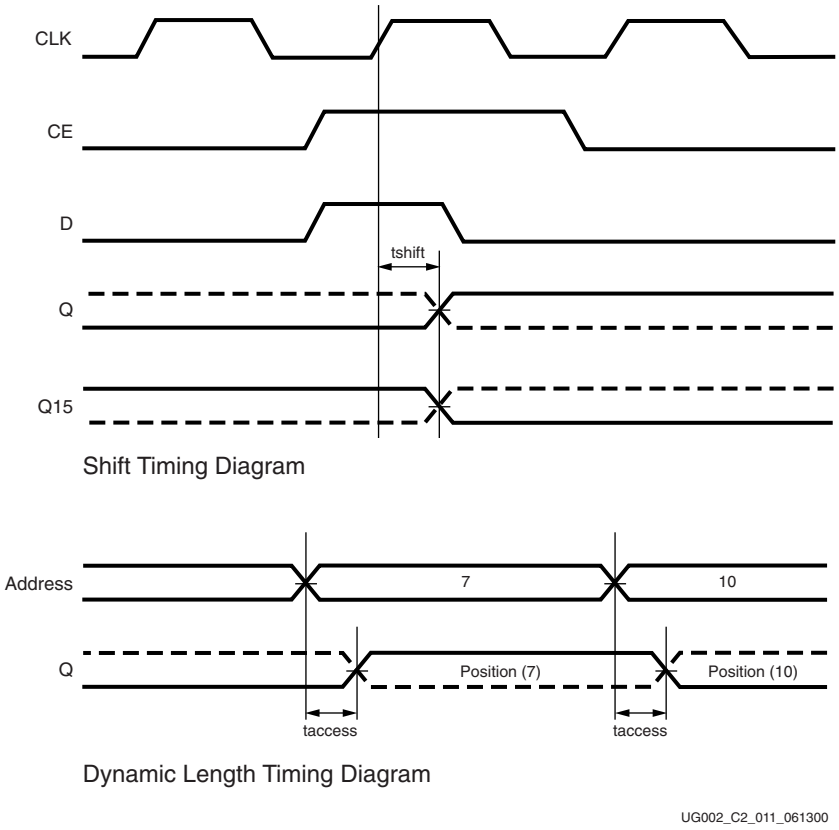


Figure 2-56: **Shift- and Dynamic-Length Timing Diagrams**

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 up 1 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.

Characteristics

- 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.

Library Primitives and Submodules

Eight library primitives are available that offer optional clock enable (CE), inverted clock ($\overline{CLK}$) and cascadable output (Q15) combinations.

Table 2-19 lists all of the available primitives for synthesis and simulation.

Table 2-19: **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, three submodules that implement 32-bit, 64-bit, and 128-bit cascadable shift registers are provided in VHDL and Verilog code. Table 2-20 lists available submodules.

Table 2-20: **Shift Register Submodules**

Submodule	Length	Control	Address Inputs	Output
SRLC32E_SUBM	32 bits	CLK, CE	A4,A3,A2,A1,A0	Q, Q31
SRLC64E_SUBM	64 bits	CLK, CE	A5, A4, A3,A2,A1,A0	Q, Q63
SRLC128E_SUBM	128 bits	CLK, CE	A6, A5, A4, A3,A2,A1,A0	Q, Q127

The submodules are based on SRLC16E primitives, which 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 2-57 represents the cascadable shift registers (32-bit and 64-bit) implemented by the submodules in Table 2-20.

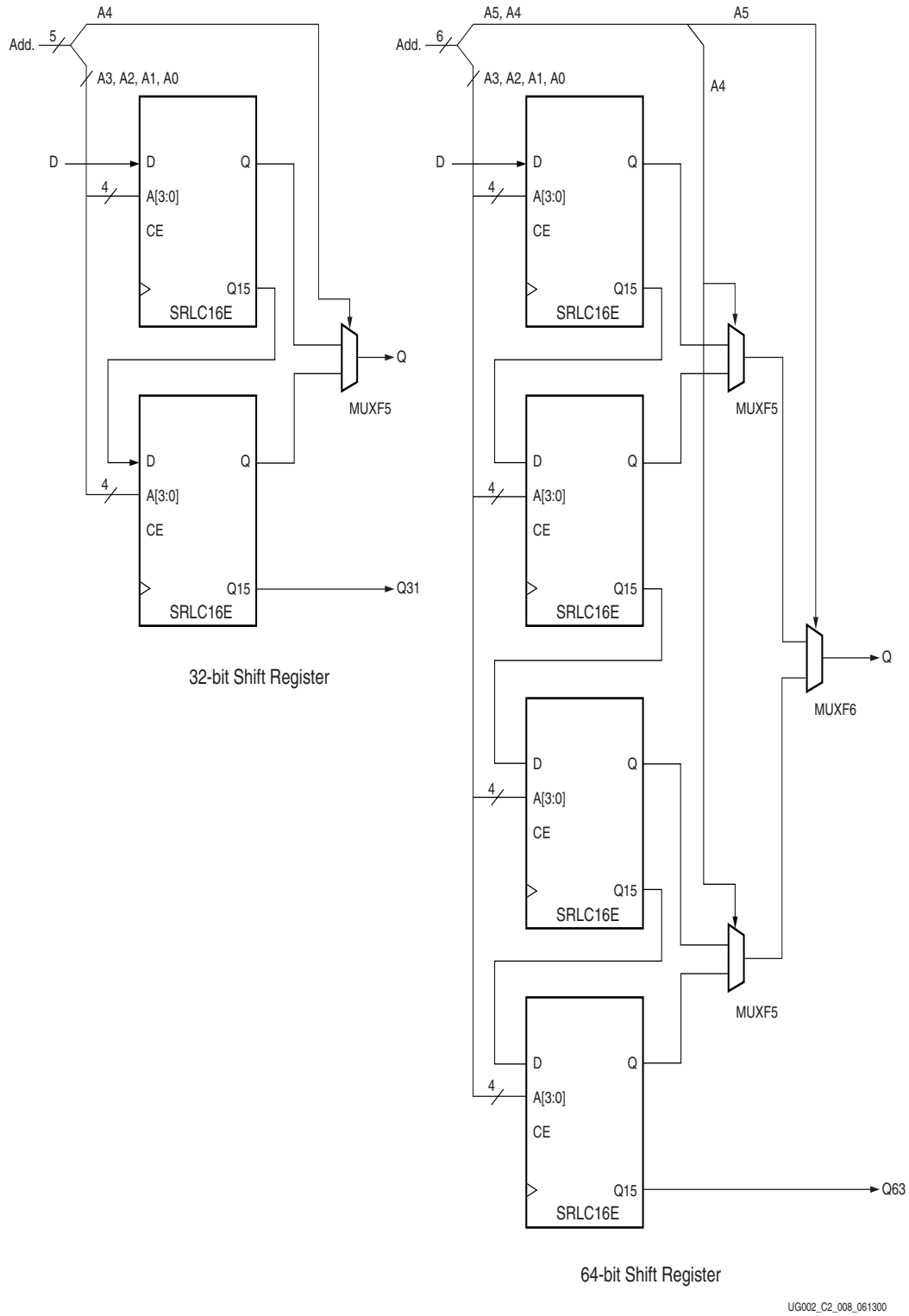


Figure 2-57: **Shift-Register Submodules (32-bit, 64-bit)**

A 128-bit shift register is built on the same scheme and uses MUXF7 (address input A6).

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 and 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 V2_SRL16E shift register instantiation code examples (in VHDL and Verilog) illustrate these techniques (see "VHDL and Verilog Templates" on page 232). V2_SRL16E.vhd and .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-in. The data and clock enable input pins have set-up 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 n^{th} bit is available on the output pin (Q). Address inputs have no effect on the cascadable output pin (Q15), which 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.

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 2-21.

Table 2-21: **Slice Coordinates in the Bottom-Left CLB Resource**

Slice S3	Slice S2	Slice S1	Slice S0
X1Y1	X1Y0	X0Y1	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 any slice.

A 64-bit shift register in static or dynamic address mode fits in two slices. These slices are either S0 and S1, or S2 and S3. Figure 2-58 illustrates the position of the four slices 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 S1 or in S3. The address selected as the output pin (Q) is the MUXF6 output.

A 128-bit shift register in static or dynamic address mode fits in a four-slice CLB resource. The data input pin has to be in slice S3. The address selected as the output pin (Q) is the MUXF7 output.

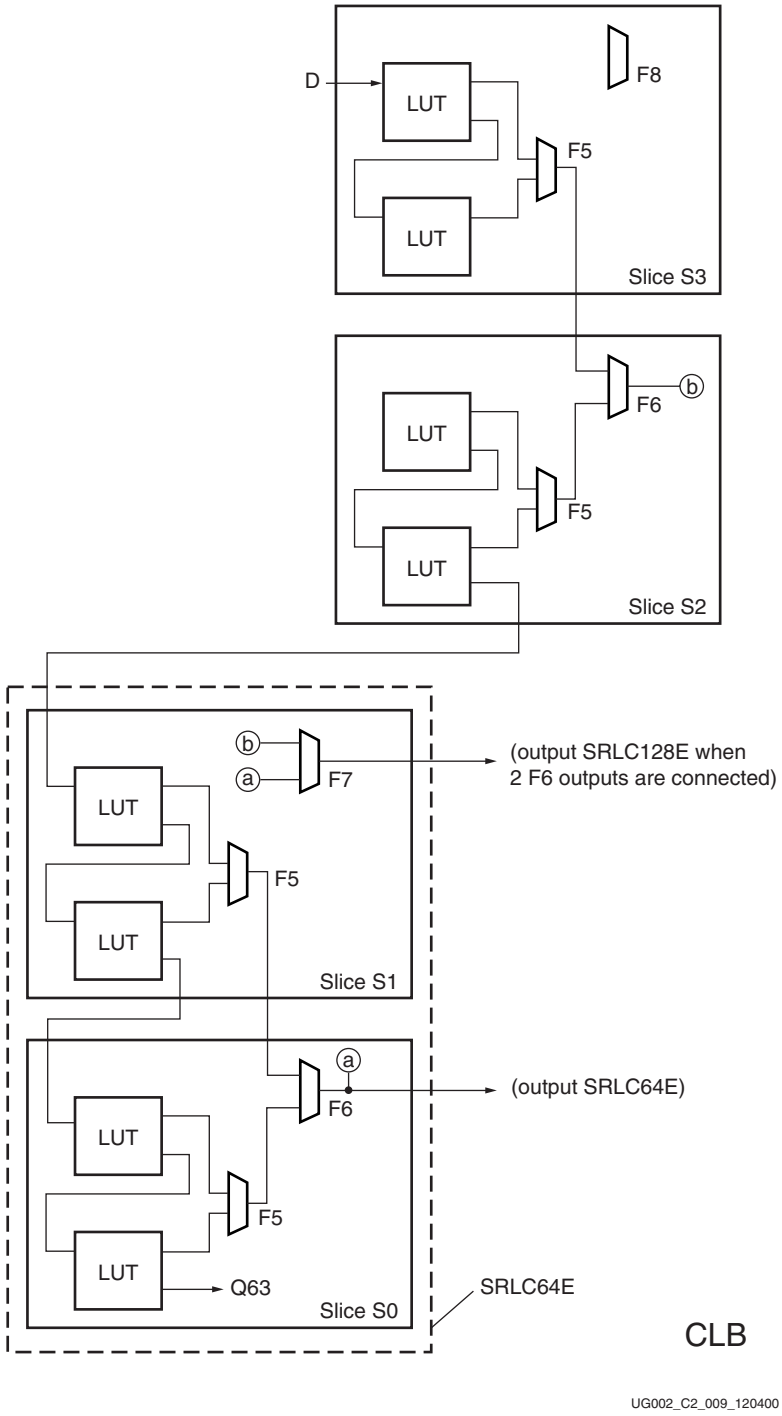


Figure 2-58: **Shift Register Placement**

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 2-59.

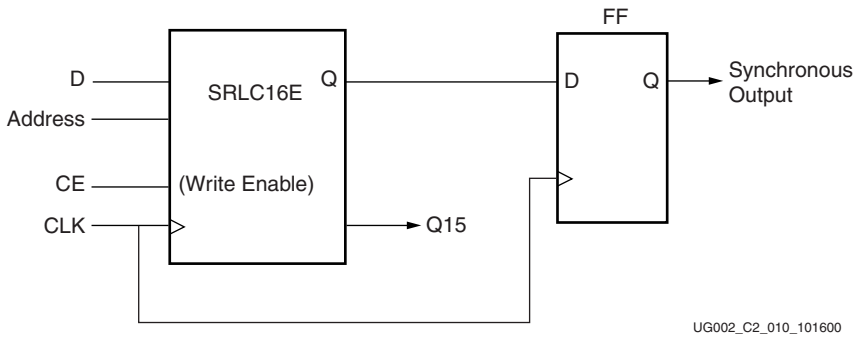


Figure 2-59: **Fully Synchronous Shift Register**

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 2-60 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.)

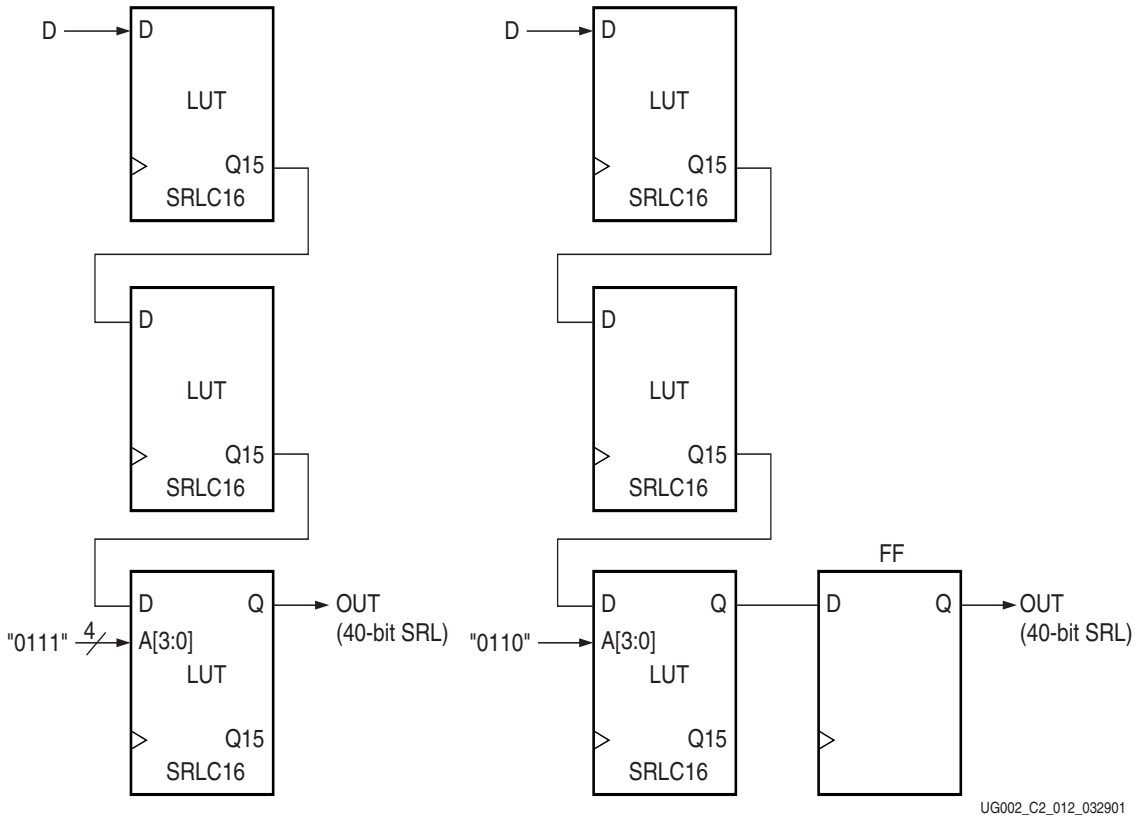


Figure 2-60: **40-bit Static-Length Shift Register**

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, 64, 128, or 256) templates are cascadable modules and instantiate the corresponding SRLCxE primitive (16) or submodule (32, 64, 128, or 256).

The ShiftRegister_16 template can be used to instantiate an SRL16 primitive.

VHDL and Verilog Templates

In template names, the number indicates the number of bits (for example, SHIFT_SELECT_16 is the template for the 16-bit shift register) and the "C" extension means the template is cascadable.

The following are templates for primitives:

- SHIFT_REGISTER_16
- SHIFT_REGISTER_16_C

The following are templates for submodules:

- SHIFT_REGISTER_32_C (submodule: SRLC32E_SUBM)
- SHIFT_REGISTER_64_C (submodule: SRLC64E_SUBM)
- SHIFT_REGISTER_128_C (submodule: SRLC128E_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-II Family
----------------------------------------------------------------------
-- Components Declarations:
--
component SRLC16E
-- pragma translate_off
  generic (
-- Shift Register initialization ("0" by default) for functional
simulation:
         INIT : bit_vector := X"0000"
  );
-- pragma translate_on
  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-II Family
//-----------------------------------------------------------------
// Syntax for Synopsys FPGA Express
// synopsys translate_off

  defparam

//Shift Register initialization ("0" by default) for functional
simulation:
  U_SRLC16E.INIT = 16'h0000;
// synopsys translate_on

//SelectShiftRegister-II Instantiation
   SRLC16E U_SRLC16E   ( .D(),
                         .A0(),
                         .A1(),
                         .A2(),
                        .A3(),
                         .CLK(),
                         .CE(),
                         .Q(),
                        .Q15()
          );

// synthesis attribute declarations
  /* synopsys attribute
  INIT "0000"
  */
```

Designing Large Multiplexers

Introduction

Virtex-II slices contain dedicated two-input multiplexers (one MUXF5 and one MUXFX per slice). These multiplexers combine the 4-input LUT outputs or the outputs of other multiplexers. Using the multiplexers MUXF5, MUXF6, MUXF7 and MUXF8 allows to combine 2, 4, 8 and 16 LUTs. Specific routing resources are associated with these 2-input multiplexers to guarantee a fast implementation of any combinatorial function built upon LUTs and MUXFX.

The combination of the LUTs and the MUXFX offers an unique solution to the design of wide-input functions. This section illustrates the implementation of large multiplexers up to 32:1. Any Virtex-II slice can implement a 4:1 multiplexer, any CLB can implement a 16:1 multiplexer, and 2 CLBs can implement a 32:1 multiplexer. Such multiplexers are just one example of wide-input combinatorial function taking advantage of the MUXFX feature. Many other logic functions can be mapped in the LUT and MUXFX features.

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 Bit Multiplexer and Bus Multiplexer modules.

For applications like comparators, encoder-decoders or “case” statement in VHDL or Verilog, these resources offer an optimal solution.

Virtex-II CLB Resources

Slice Multiplexers

Each Virtex-II slice has a MUXF5 to combine the outputs of the 2 LUTs and an extra MUXFX. Figure 2-61 illustrates a combinatorial function with up to 9 inputs in one slice.

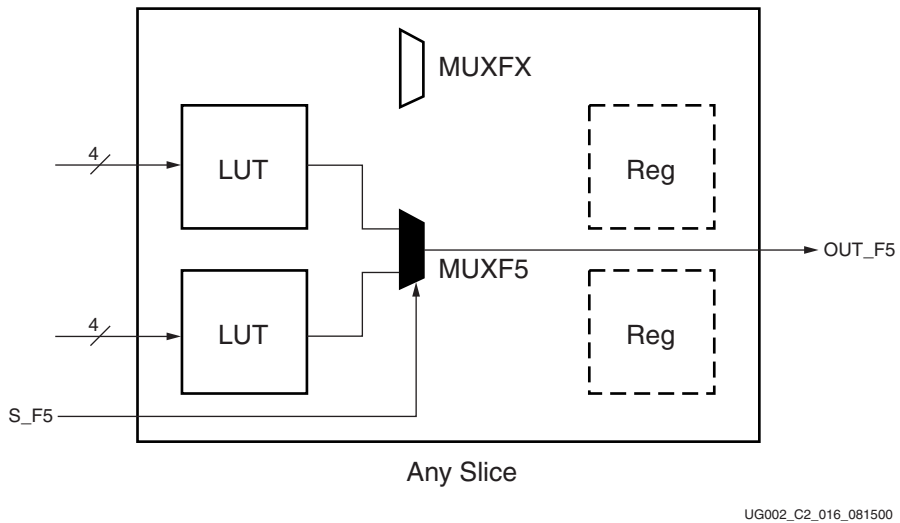


Figure 2-61: **LUTs and MUXF5 in a Slice**

Each Virtex-II CLB contains 4 slices. The second MUXFX implements a MUXF6, MUXF7 or MUXF8 according to the position of the slice in the CLB. These MUXFX are designed to allow LUTs combination up to 16 LUTs in two adjacent CLBs.

Figure 2-62 shows the relative position of the slices in the CLB.

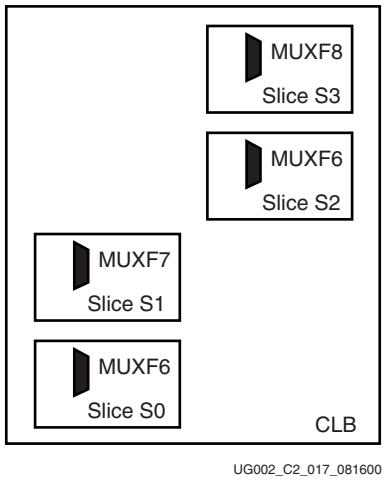


Figure 2-62: **Slice Positions in a CLB**

Slices S0 and S2 have a MUXF6, designed to combine the outputs of two MUXF5 resources. Figure 2-63 illustrates a combinatorial function up to 18 inputs in the slices S0 and S1, or in the slices S2 and S3.

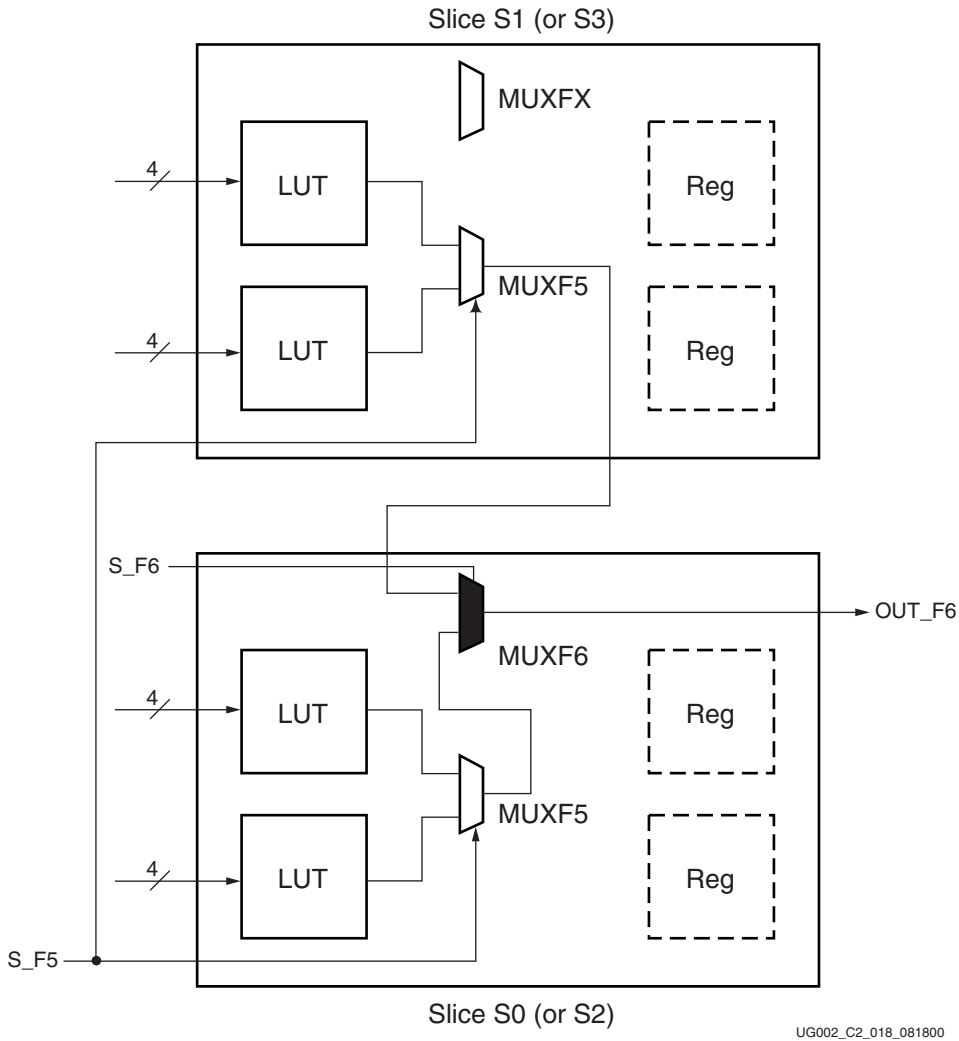


Figure 2-63: **LUTs and (MUXF5 and MUXF6) in Two Slices**

The slice S1 has a MUXF7, designed to combine the outputs of two MUXF6. Figure 2-64 illustrates a combinatorial function up to 35 inputs in a Virtex-II CLB.

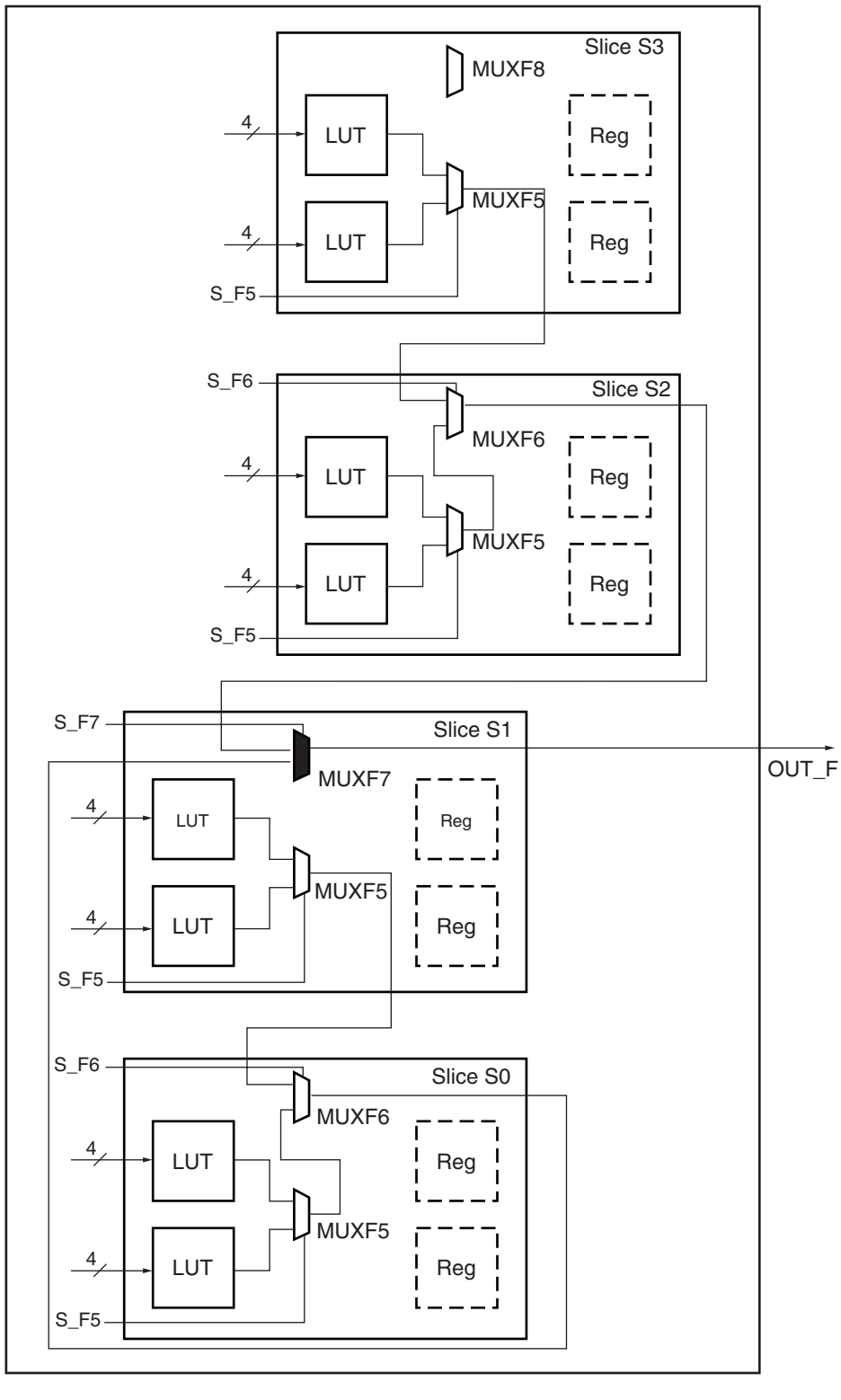


Figure 2-64: **LUTs and (MUXF5, MUXF6, and MUXF7) in One CLB**

The slice S3 of each CLB has a MUXF8. combinatorial functions of up to 68 inputs fit in two CLBs as shown in Figure 2-65. The outputs of two MUXF7 are combined through dedicated routing resources between two adjacent CLBs in a column.

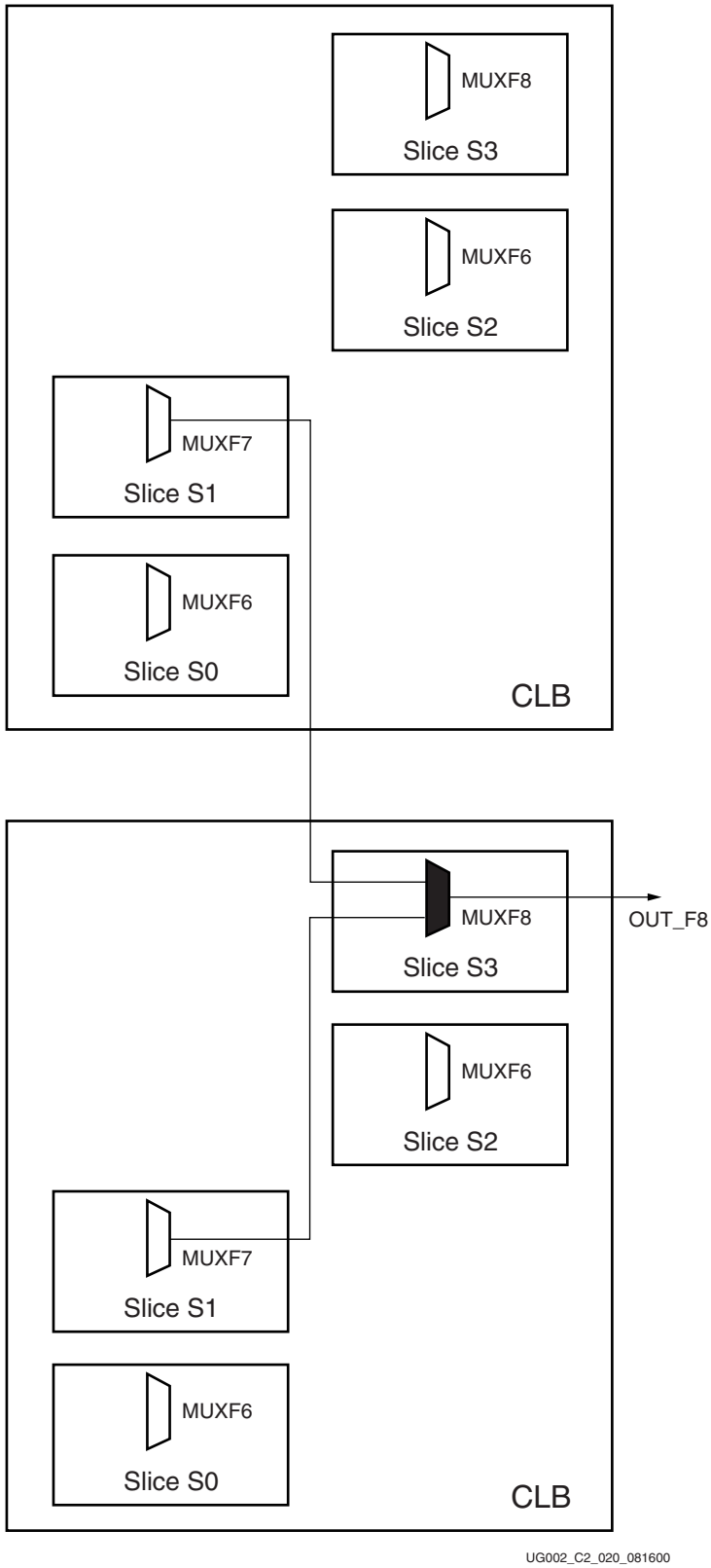


Figure 2-65: **MUXF8 Combining Two Adjacent CLBs**

Wide-Input Multiplexers

Each LUT can implement a 2:1 multiplexer. In each slice, the MUXF5 and two LUTs can implement a 4:1 multiplexer. As shown in Figure 2-66, the MUXF6 and two slices can implement a 8:1 multiplexer. The MUFXF7 and the four slices of any CLB can implement a 16:1 and the MUXF8 and two CLBs can implement a 32:1 multiplexer.

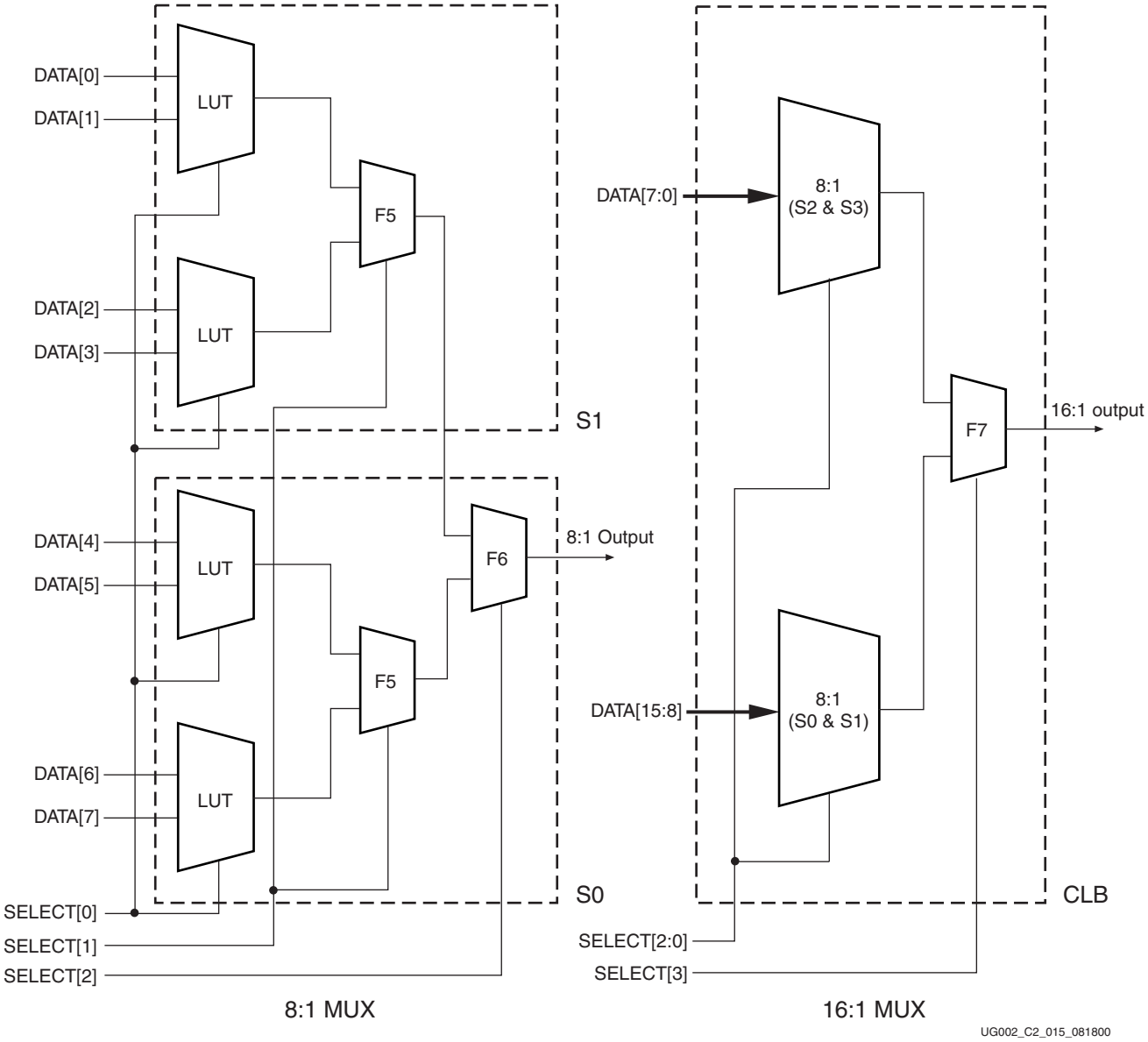


Figure 2-66: **8:1 and 16:1 Multiplexers**

Characteristics

- Implementation in one level of logic (LUT) and dedicated MUXFX
- Full combinatorial path

Library Primitives and Submodules

Four library primitives are available that offer access to the dedicated MUXFX in each slice. In the example shown in Table 2-22, MUXF7 is available only in slice S1.

Table 2-22: **MUXFX Resources**

Primitive	Slice	Control	Input	Output
MUXF5	S0, S1, S2, S3	S	I0, I1	O
MUXF6	S0, S2	S	I0, I1	O
MUXF7	S1	S	I0, I1	O
MUXF8	S3	S	I0, I1	O

In addition to the primitives, five submodules that implement multiplexers from 2:1 to 32:1 are provided in VHDL and Verilog code. Synthesis tools can automatically infer the above primitives (MUXF5, MUXF6, MUXF7, and MUXF8); however, the submodules described in this section used instantiation of the new MUXFX to guarantee an optimized result. Table 2-23 lists available submodules:

Table 2-23: **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 2-24 shows the DATA_I selected for each SELECT_I value.

Table 2-24: **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.

Applications

Multiplexers are used in various applications. These are often inferred by synthesis tools when a "case" statement is used (see the example below). 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-II 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 "cut and pasted" 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 following examples can be used as guidelines for designing other wide-input functions.

The following submodules are available:

- 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 in VHDL and Verilog are provided as example.

VHDL Template

```
-- Module: MUX_16_1_SUBM
-- Description: Multiplexer 16:1
--
-- Device: Virtex-II Family
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

-- Syntax for Synopsys FPGA Express
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on

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 Declarations:
signal DATA_MSB : std_logic;
signal DATA_LSB : std_logic;
--
begin
--
-- If synthesis tools support MUXF7 :
--SELECT_PROCESS: process (SELECT_I, DATA_I)
--begin
--case SELECT_I is
--  when "0000" => DATA_O <= DATA_I (0);
--  when "0001" => DATA_O <= DATA_I (1);
--  when "0010" => DATA_O <= DATA_I (2);
--  when "0011" => DATA_O <= DATA_I (3);
--  when "0100" => DATA_O <= DATA_I (4);
--  when "0101" => DATA_O <= DATA_I (5);
--  when "0110" => DATA_O <= DATA_I (6);
--  when "0111" => DATA_O <= DATA_I (7);
--  when "1000" => DATA_O <= DATA_I (8);
--  when "1001" => DATA_O <= DATA_I (9);
--  when "1010" => DATA_O <= DATA_I (10);
--  when "1011" => DATA_O <= DATA_I (11);
--  when "1100" => DATA_O <= DATA_I (12);
--  when "1101" => DATA_O <= DATA_I (13);
--  when "1110" => DATA_O <= DATA_I (14);
--  when "1111" => DATA_O <= DATA_I (15);
--  when others => DATA_O <= 'X';
--end case;
--end process SELECT_PROCESS;
--
-- If synthesis tools DO NOT support MUXF7 :
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-II 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];

/*
//If synthesis tools supports MUXF7 :
always @ (DATA_I or SELECT_I)

    case (SELECT_I)
       4'b0000 : DATA_O <= DATA_I[0];
  4'b0001 : DATA_O <= DATA_I[1];
  4'b0010 : DATA_O <= DATA_I[2];
  4'b0011 : DATA_O <= DATA_I[3];
        4'b0100 : DATA_O <= DATA_I[4];
  4'b0101 : DATA_O <= DATA_I[5];
  4'b0110 : DATA_O <= DATA_I[6];
  4'b0111 : DATA_O <= DATA_I[7];
        4'b1000 : DATA_O <= DATA_I[8];
  4'b1001 : DATA_O <= DATA_I[9];
  4'b1010 : DATA_O <= DATA_I[10];
  4'b1011 : DATA_O <= DATA_I[11];
```

```
        4'b1100 : DATA_O <= DATA_I[12];
  4'b1101 : DATA_O <= DATA_I[13];
  4'b1110 : DATA_O <= DATA_I[14];
  4'b1111 : DATA_O <= DATA_I[15];
  default : DATA_O <= 1'bx;
    endcase
*/

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

//
*/
```

Implementing Sum of Products (SOP) Logic

Introduction

Virtex-II slices contain a dedicated two-input multiplexer (MUXCY) and a two-input OR gate (ORCY) to perform operations involving wide AND and OR gates. These combine the four-input LUT outputs. These gates can be cascaded in a chain to provide the wide AND functionality across slices. The output from the cascaded AND gates can then be combined with the dedicated ORCY to produce the Sum of Products (SOP).

Virtex-II CLB Resources

Each Virtex-II slice has a MUXCY, which uses the output from the LUTs as a SELECT signal. Depending on the width of data desired, several slices can be used to provide the SOP output. Figure 2-67 illustrates the logic involved in designing a 16-input AND gate. It utilizes the 4-input LUT to provide the necessary SELECT signal for the MUXCY. Only when all of the input signals are High, can the V_{CC} at the bottom reach the output. This use of carry logic helps to perform AND functions at high speed and saves logic resources.

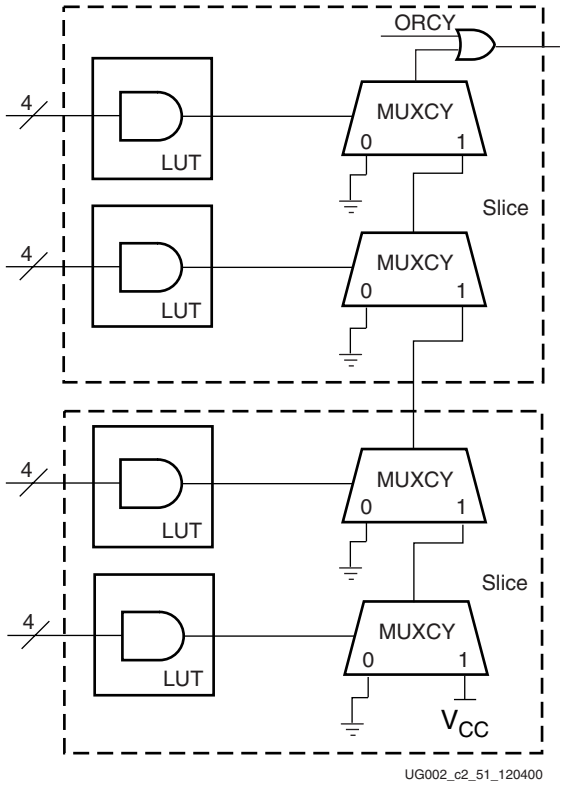


Figure 2-67: **Implementing a 16-bit Wide AND Gate Using MUXCY & ORCY**

The output from the chain of AND gates is passed as one of the inputs of the dedicated OR gate, ORCY. To calculate the SOP, these AND chains can be cascaded vertically across several CLBs, depending on the width of the input data. Figure 2-68 illustrates how the AND outputs are then passed in through the ORCY gates in a horizontal cascade, the sum of which is the Sum of Products.

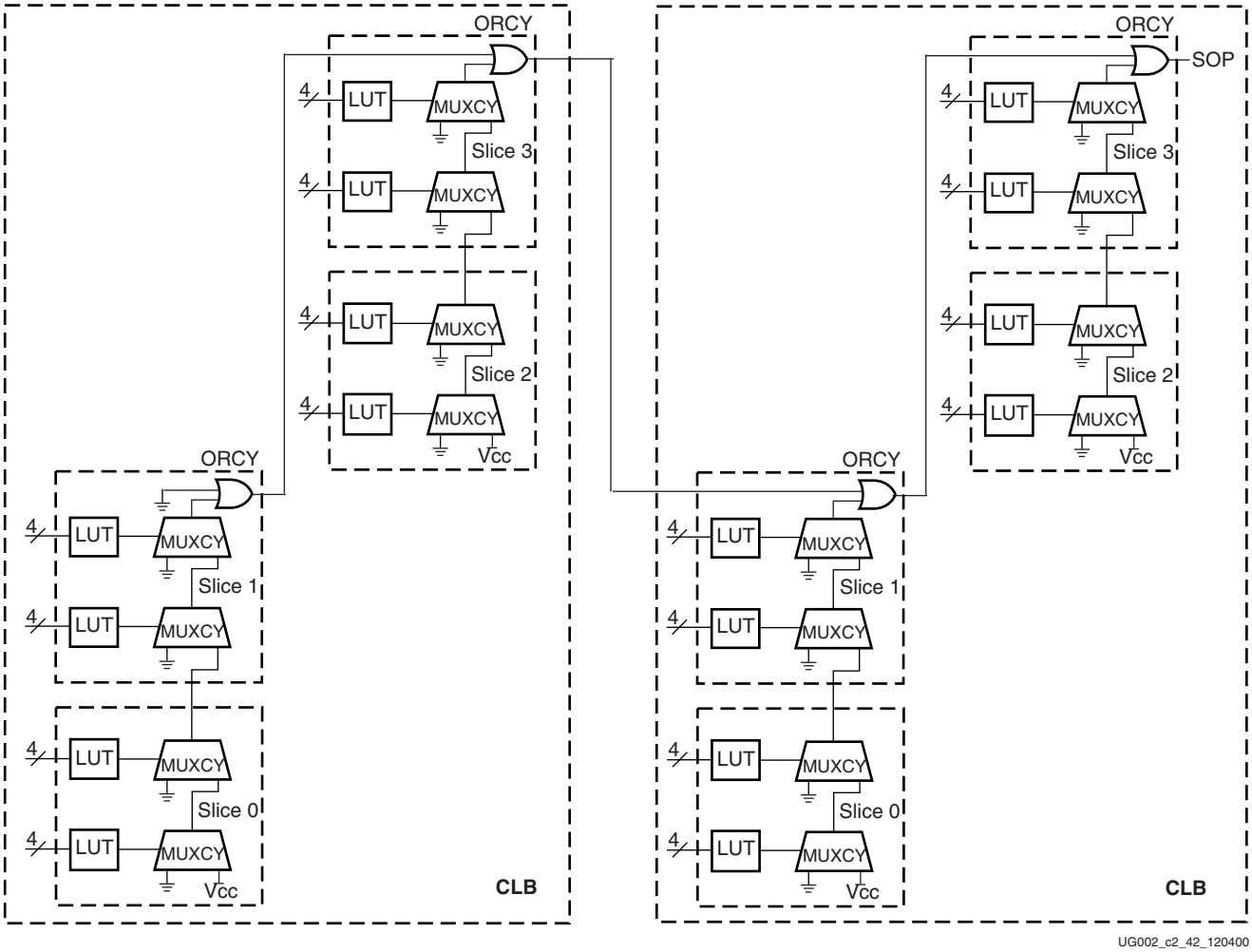


Figure 2-68: **64-bit Input SOP Design**

Port Signals

AND_WIDTH Parameter

The width of each AND gate used in the cascade.

PROD_TERM Parameter

The number of AND gates used along each vertical cascade.

AND_IN Parameter

Data input to the AND gates. The total width of data is calculated from the product of AND_WIDTH and PROD_TERM

SOP_OUT Parameter

The Sum of Products (SOP) output data from the cascade chain.

Applications

These logic gates can be used in various applications involving very wide AND gates and Sum of Products (SOP) functions.

VHDL and Verilog Instantiation

To implement wide-input AND functions, MUXCY and ORCY primitives can be instantiated in VHDL or Verilog code. The submodule code provided can be used to implement wide-input AND gates for any width of input data.

VHDL and Verilog Submodules

VHDL and Verilog submodules are available to implement the cascade chain of wide-input AND gates and OR gates to calculate the Sum of Products (SOP). The VHDL module provided uses a generic case, where the width of data and the product terms can be specified in the case. The Verilog module provides a 64-bit input example, using four wide AND chains, each of which handle 16 bits of data.

VHDL Templates

```
-- Module : AND_CHAIN
-- Description : 16 input AND gate
--
-- Device : Virtex-II Family
--------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--library UNISIM;
--use UNISIM.VCOMPONENTS.ALL;

entity AND_CHAIN is
 generic (
    input_width : integer); --must be a 4x value
 port (
    data_in : in std_logic_vector( input_width-1 downto 0);
    carry_in : in std_logic;
    out_andor_chain : out std_logic);
end AND_CHAIN;

architecture AND_CHAIN_arch of AND_CHAIN is

component ORCY
  port( i : std_logic;
        ci : in std_logic;
        o : out std_logic);
end component;

component AND_LOGIC
  port( sel_data : in std_logic_vector(3 downto 0);
        data_cin : in std_logic;
        data_out : out std_logic);
end component;

signal VCC, GND : std_logic;
signal cout : std_logic_vector(input_width/4 downto 0);
signal out_and_chain : std_logic;

begin

VCC <= '1';
GND <= '0';

--initialisation of first input for MUXCY
cout(0) <= VCC;

and_chain_x : for i in (input_width/4) - 1 downto 0 generate
```

```
  AND_LOGIC_inst : AND_LOGIC
        port map (
           sel_data => data_in((4 * i + 3) downto (4 * i)),
           data_cin => cout(i),
           data_out => cout(i + 1));
end generate;

out_and_chain <= cout(input_width/4);

orcy_inst : ORCY
  port map( i => out_and_chain,
        ci => carry_in,
        o => out_andor_chain);

end AND_CHAIN_arch;

----------------------------------------------------------------------
-- Module AND_LOGIC
-- Description : 4-input AND gate
--
-- Device : Virtex-II Family
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--library UNISIM;
--use UNISIM.VCOMPONENTS.ALL;

entity AND_LOGIC is
  port(
      sel_data : in std_logic_vector(3 downto 0); -- data for select
signal for MUXCY from LUT
      data_cin : in std_logic;  -- result from previous stage
      data_out : out std_logic);
end AND_LOGIC;

architecture AND_LOGIC_arch of AND_LOGIC is

component MUXCY
 port(
     DI : in std_logic;
     CI : in std_logic;
     s : in std_logic;
     o : out std_logic);
end component;

signal GND : std_logic;
signal sel:std_logic;

begin

GND <= '0';
sel <= sel_data(0) and sel_data(1) and sel_data(2) and sel_data(3);

--Wide AND gate using MUXCY
MUX : MUXCY
     port map (
        DI => GND,
        CI => data_cin,
        s => sel,
        o => data_out);

end AND_LOGIC_arch;
```

```
----------------------------------------------------------------------
-- Module : SOP_SUBM
-- Description : Implementing SOP using MUXCY and ORCY
--
-- Device : Virtex-II Family
----------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--library UNISIM;
--use UNISIM.VCOMPONENTS.ALL;

entity SOP_SUBM is
  generic(
     and_width : integer :=16 ;
     prod_term : integer := 4 );
  port(
     and_in : in std_logic_vector(and_width * prod_term - 1 downto 0);
     sop_out : out std_logic);
end SOP_SUBM;

architecture SOP_SUBM_arch of SOP_SUBM is

component AND_CHAIN
 generic (
    input_width : integer); --must be a 4x value
 port (
    data_in : in std_logic_vector( input_width-1 downto 0);
    carry_in : in std_logic;
    out_andor_chain : out std_logic);
end component;

signal VCC, GND : std_logic;
signal carry : std_logic_vector(prod_term downto 0);

begin

VCC <= '1';
GND <= '0';

carry(0) <= GND;
andor_inst : for i in 0 to (prod_term - 1) generate
      and_chainx : AND_CHAIN
             generic map(
                input_width => and_width)
             port map(
              data_in => and_in((and_width * i + (and_width -1)) downto
(and_width * i)),
                carry_in => carry(i),
                out_andor_chain => carry(i + 1));
end generate;
sop_out <= carry(prod_term);

end SOP_SUBM_arch;
```

Verilog Templates

```
// Module : AND_CHAIN
// Description : 16 input AND gate
//
// Device : Virtex-II Family
//--------------------------------------------------------------------
module AND_CHAIN(data_in, carry_in, out_andor_chain);
input [15:0] data_in;
input carry_in;
output out_andor_chain;
wire VCC = 1'b1;
wire out_and_chain;
wire dat_out1, data_out2, data_out3;
AND_LOGIC_OR u4(.sel_data(data_in[15:12]), .data_cin(data_out3),
.carry_in(carry_in), .data_out(out_andor_chain));

AND_LOGIC u3(.sel_data(data_in[11:8]), .data_cin(data_out2),
.data_out(data_out3));
AND_LOGIC u2(.sel_data(data_in[7:4]), .data_cin(data_out1),
.data_out(data_out2));
AND_LOGIC u1(.sel_data(data_in[3:0]), .data_cin(VCC),
.data_out(data_out1));
endmodule

//--------------------------------------------------------------------
// Module AND_LOGIC
// Description : 4-input AND gate
//
// Device : Virtex-II Family
//--------------------------------------------------------------------
// Module : init_and
//
module AND_LOGIC(sel_data, data_cin, data_out);
input[3:0] sel_data;
input data_cin;
output data_out;
wire GND = 1'b0;
wire VCC = 1'b1;
wire and_out;
assign and_out = sel_data[3] & sel_data[2] & sel_data[1] & sel_data[0];
MUXCY muxcy_inst (.DI(GND), .CI(data_cin), .S(and_out), .O(data_out));
endmodule

// Module AND_LOGIC + ORCY
module AND_LOGIC_OR(sel_data, data_cin, carry_in, data_out);
input[3:0] sel_data;
input data_cin;
input carry_in;
output data_out;
wire data_mux_out;
wire GND = 1'b0;
wire VCC = 1'b1;
wire and_out;
assign and_out = sel_data[3] & sel_data[2] & sel_data[1] & sel_data[0];
MUXCY muxcy_inst (.DI(GND), .CI(data_cin), .S(and_out),
.O(data_mux_out)) /* synthesis  RLOC="x0y0" */;
ORCY u5(.I(carry_in), .CI(data_mux_out), .O(data_out)) /* synthesis
RLOC="x0y0" */;
endmodule
```

```
//----------------------------------------------------------------------
// Module : SOP_SUBM
// Description : Implementing SOP using MUXCY and ORCY
//
// Device : Virtex-II Family
//----------------------------------------------------------------------
module SOP_SUBM(and_in, sop_out);
input [63:0] and_in;
output sop_out;
wire out_andor_chain1, out_andor_chain2, out_andor_chain3;
wire GND = 1'b0;
AND_CHAIN u4(.data_in(and_in[63:48]), .carry_in(out_andor_chain3),
.out_andor_chain(sop_out));
AND_CHAIN u3(.data_in(and_in[47:32]), .carry_in(out_andor_chain2),
.out_andor_chain(out_andor_chain3));
AND_CHAIN u2(.data_in(and_in[31:16]), .carry_in(out_andor_chain1),
.out_andor_chain(out_andor_chain2));
AND_CHAIN u1(.data_in(and_in[15:0]), .carry_in(GND),
.out_andor_chain(out_andor_chain1));
endmodule
```

Using Embedded Multipliers

Introduction

Virtex-II devices feature a large number of embedded 18-bit X 18-bit two's-complement embedded multipliers. The embedded multipliers offer fast, efficient means to create 18-bit signed by 18-bit signed multiplication products. The multiplier blocks share routing resources with the Block SelectRAM memory, allowing for increased efficiency for many applications. Cascading of multipliers can be implemented with additional logic resources in local Virtex-II slices.

Applications such as signed-signed, signed-unsigned, and unsigned-unsigned multiplication, logical, arithmetic, and barrel shifters, two's-complement and magnitude return are easily implemented.

Using the CORE Generator, the designer can quickly generate multipliers that make use of the embedded 18-bit x 18-bit two's-complement multipliers (V2.0 or later) of the Multiplier core for Virtex-II devices.

Two's-Complement Signed Multiplier

Data Flow

Each embedded multiplier block (MULT18X18 primitive) supports two independent dynamic data input ports: 18-bit signed or 17-bit unsigned. The MULT18X18 primitive is illustrated in Figure 2-69.

In addition, efficient cascading of multipliers up to 35-bit X 35-bit signed can be accomplished by using 4 embedded multipliers, one 36-bit adder, and one 53-bit adder. See Figure 2-70.

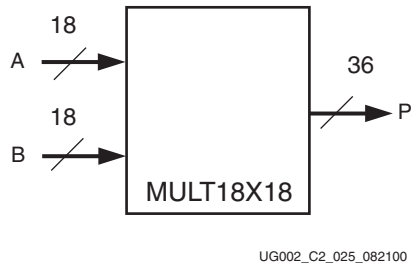


Figure 2-69: **Embedded Multiplier**

Library Primitives and Submodules

One library primitive (MULT18X18) is available. Table 2-25 lists the attributes of this primitive.

Table 2-25: **Embedded Multiplier Primitive**

Primitive	A width	B width	P width	Signed/Unsigned
MULT18X18	18	18	36	Signed (2's complement)

In addition to the primitive, 15 submodules that implement various widths of signed and unsigned multipliers and two's-complement return functions are provided in VHDL and Verilog code. Multipliers using cascaded MULT18X18 primitives are included with registers between stages causing three cycles of latency. Multipliers that make use of the embedded Virtex-II 18-bit by 18-bit two's complement multipliers can be easily generated using V2.0 of the CORE Generator Multiplier module. Table 2-26 lists cascaded multiplier submodules.

Table 2-26: **Embedded Multiplier Submodules - Cascaded MULT18X18**

Submodule	A Width	B Width	P Width	Signed/Unsigned
MULT35X35_S	35	35	70	Signed
MULT34X34_U	34	34	68	Unsigned

Figure 2-70 represents the cascaded scheme used to implement a 35-bit by 35-bit signed multiplier utilizing four embedded multipliers and two adders.

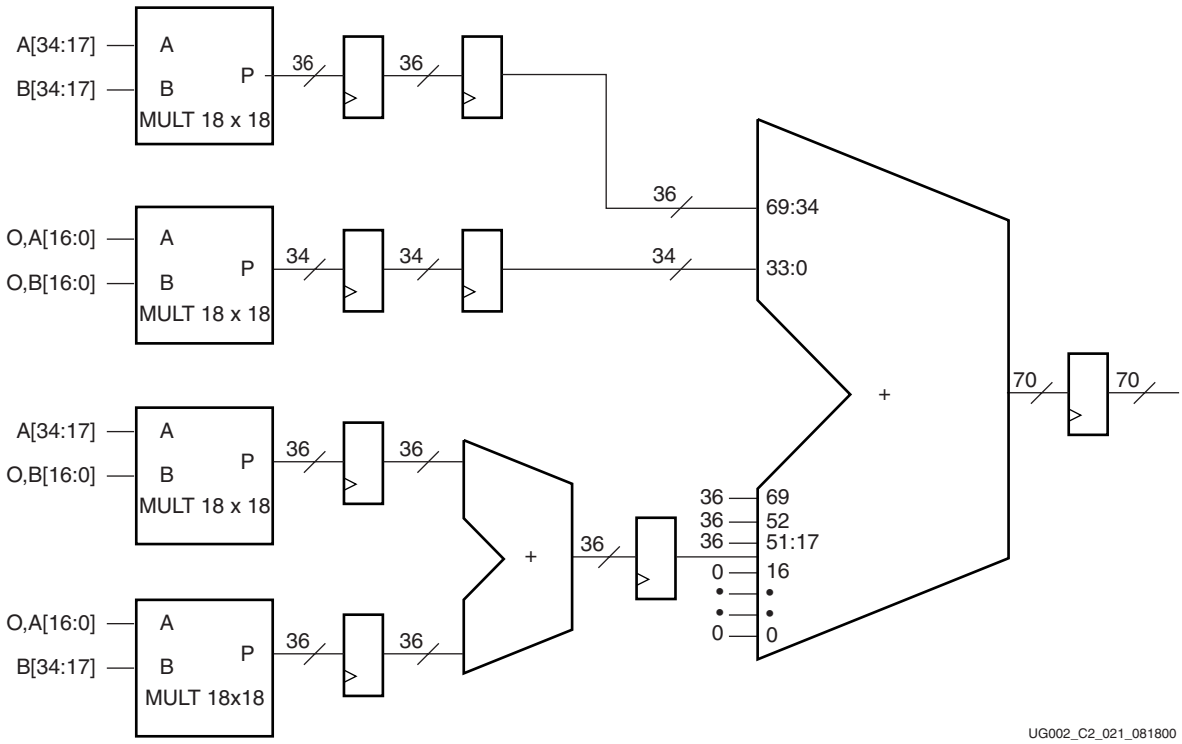


Figure 2-70: **MULT35X35_S Submodule**

The fixed adder is 53 bits wide (17 LSBs are always 0 on one input).

The 34-bit by 34-bit unsigned submodule is constructed in a similar manner with the most significant bit on each operand being tied to logic low.

Table 2-26 lists multipliers and two's-complement return functions that utilize one MULT18X18 primitive and are not registered.

Table 2-27: **Embedded Multiplier Submodules - Single MULT18X18**

Submodule	A width	B width	P width	Signed/Unsigned
MULT17X17_U	17	17	34	Unsigned
MULT8X8_S	8	8	16	Signed
MULT8X8_U	8	8	16	Unsigned
MULT4X4_S	4	4	8	Signed
MULT4X4_U	4	4	8	Unsigned
MULT_6X6S_5X5U	6 5	6 5	12 10	Signed Unsigned
MULT_5X5S_6X6U	5 6	5 6	10 12	Signed Unsigned
MULT_5X5U_5X5U	5 5	5 5	10 10	Unsigned Unsigned
MULT_4X4S_7X7U	4 7	4 7	8 14	Signed Unsigned
MULT_4X4S_3X3S	4 3	4 3	8 6	Signed Signed
TWOS_CMP18	18	-	18	-
TWOS_CMP9	9	-	9	-
MAGNTD_18	18	-	17	-

Multipliers of form MULT_aXaS_bXbU use one embedded multiplier to implement two multipliers with separate outputs. The submodules listed above use optimized pin assignments to achieve shortest possible through-delay.

Figure 2-71 and Figure 2-72 represent 4-bit by 4-bit signed multiplier and 4-bit by 4-bit unsigned multiplier implementations, respectively.

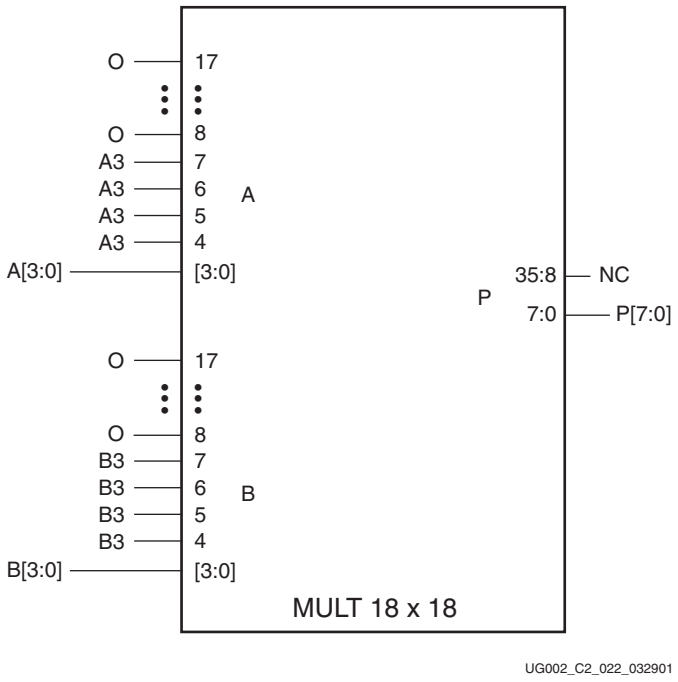


Figure 2-71: **MULT4X4_S Submodule**

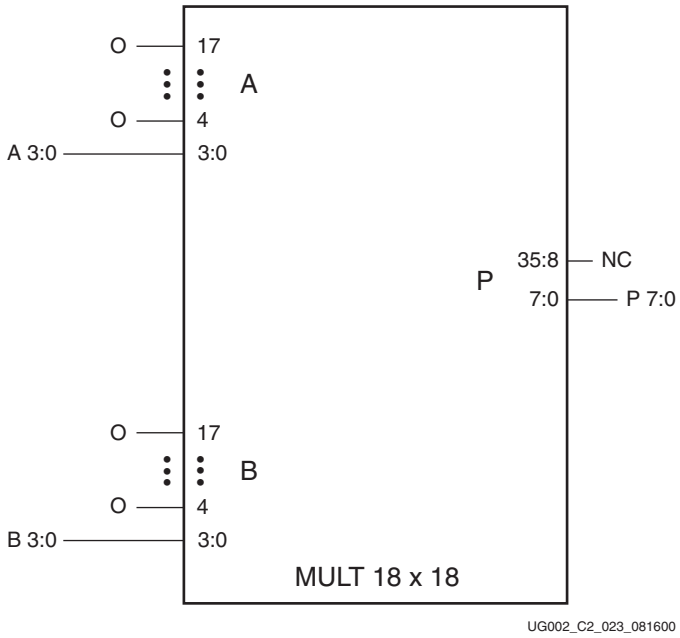


Figure 2-72: **MULT4X4_U Submodule**

Submodule MAGNTD_18 performs a magnitude return (i.e., absolute value) of a two's-complement number. An incoming negative number returns with a positive number, while an incoming positive number remains unchanged. Submodules TWOS_CMP18 and TWOS_CMP9 perform a two's-complement return function. The incoming number in two's-complement form (either signed or unsigned) is complemented when the DO_COMP pin is asserted High. Additional slice logic can be used with these submodules to efficiently convert sign-magnitude to two's-complement or vice-versa. Figure 2-73 shows the connections to a MULT18X18 to create the submodule TWOS_CMP9.

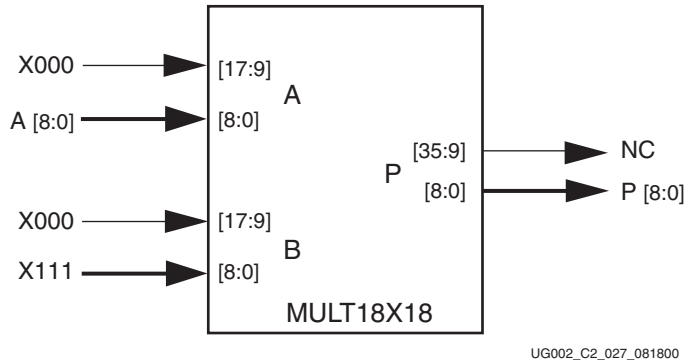


Figure 2-73: **TWOS_CMP9 Submodule**

Two Multipliers in a Single Primitive

Two multipliers can be implemented in a single primitive. For simplified illustration purposes, an assumption of two squares being implemented in the same MULT18X18 primitive is used. The following equation shows the form of the multiplication.

Two Multipliers per Primitive:

$$(X * 2^n + Y)(X * 2^n + Y) = (X^2 * 2^{2n}) + (Y^2) + (XY * 2^{n+1})$$

$(X * 2^n)$ is the input X appearing on the MSBs while Y appears on the LSBs to form the value $(X * 2^n + Y)$. Two multipliers can coexist in one MULT18X18 primitive, if the conditions in the following inequalities are met when neither X nor Y are 0.

Inequality Conditions for Two Multipliers per Primitive:

$(X^2 * 2^{2n})_{min} > (XY * 2^{n+1})_{max}$, $(XY * 2^{n+1})_{min} > (Y^2)_{max}$

For values 0 on X or Y, the equation becomes:

$X^2 * 2^{2n}$	{Y=0}
Y^2	{X=0}
0	{X=0, Y=0}

Figure 2-74 represents the MULT_6X6S_5X5U submodule.

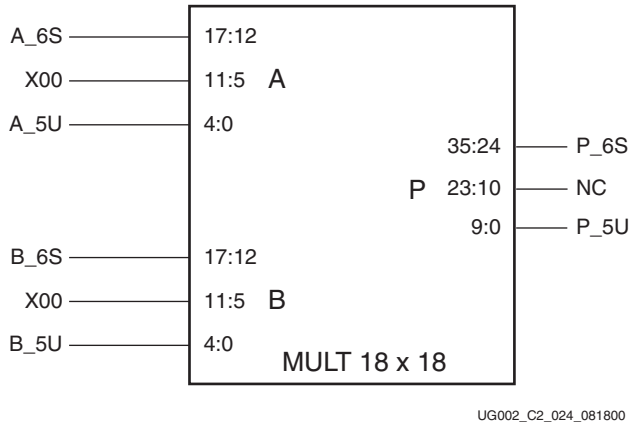


Figure 2-74: **MULT_6X6S_5X5U -- Connections to a MULT18X18 Primitive**

Table 2-28 shows values for X and Y where these conditions are met.

Table 2-28: **Two Multipliers per MULT18X18 Allowable Sizes**

X * X		Y * Y	
Signed Size	**Unsigned Size**	**Signed Size**	**Unsigned Size**
7 X 7	6 X 6	-	4 X 4
6 X 6	5 X 5	-	5 X 5
5 X 5	4 X 4	3 X 3	6 X 6
4 X 4	3 X 3	3 X 3	7 X 7
3 X 3	2 X 2	4 X 4	8 X 8

VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available as examples of primitives and submodules (see "VHDL and Verilog Templates" on page 256).

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.

Port Signals

Data In - A

The data input provides new data (up to 18 bits) to be used as one of the multiplication operands.

Data In - B

The data input provides new data (up to 18 bits) to be used as one of the multiplication operands.

Data Out - P

The data output bus P provides the data value (up to 36 bits) of two's-complement multiplication for operands A and B.

Location Constraints

Each embedded multiplier has location coordinates of the form XrowYcolumn. To constrain placement, multiplier instances can have LOC properties attached to

MULT18X18 embedded multiplier instances can have LOC properties attached to them to constrain placement. MULT18X18 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 = MULT18X18_X#Y#
```

For example, MULT18X18_X0Y0 is the bottom-left MULT18X18 location on the device.

VHDL and Verilog Templates

VHDL and Verilog templates are available for the primitive and submodules.

The following is a template for the primitive:

- SIGNED_MULT_18X18 (primitive: MULT18X18)

The following are templates for submodules:

- SIGNED_MULT_35X35 (submodule: MULT35X35_S)
- UNSIGNED_MULT_34X34 (submodule: MULT34X34_U)
- UNSIGNED_MULT_17X17 (submodule: MULT17X17_U)
- SIGNED_MULT_8X8 (submodule: MULT8X8_S)
- UNSIGNED_MULT_8X8 (submodule: MULT8X8_U)
- SIGNED_MULT_4X4 (submodule: MULT4X4_S)
- UNSIGNED_MULT_4X4 (submodule: MULT4X4_U)
- DUAL_MULT_6X6S_5X5U (submodule: MULT_6X6S_5X5U)
- DUAL_MULT_5X5S_6X6U (submodule: MULT_5X5S_6X6U)
- DUAL_MULT_5X5U_5X5U (submodule: MULT_5X5U_5X5U)
- DUAL_MULT_4X4S_7X7U (submodule: MULT_4X4S_7X7U)
- DUAL_MULT_4X4S_3X3S (submodule: MULT_4X4S_3X3S)
- TWOS_COMPLEMENTER_18BIT (submodule: TWOS_CMP18)
- TWOS_COMPLEMENTER_9BIT (submodule: TWOS_CMP9)
- MAGNITUDE_18BIT (submodule: MAGNTD_18)

The corresponding submodules have to be synthesized with the design.

Templates for the SIGNED_MULT_18X18 module are provided in VHDL and Verilog code as an example.

VHDL Template:

```
-- Module: SIGNED_MULT_18X18
-- Description: VHDL instantiation template
-- 18-bit X 18-bit embedded signed multiplier (asynchronous)
--
-- Device: Virtex-II Family
------------------------------------------------------------------------
-- Components Declarations
component MULT18X18
  port(
       A :  in std_logic_vector (17 downto 0);
       B :  in std_logic_vector (17 downto 0);
       P : out std_logic_vector (35 downto 0)
  );
end component;
--
-- Architecture Section
--
U_MULT18X18 : MULT18X18
  port map (
    A => , -- insert input signal #1
    B => , -- insert input signal #2
    P =>   -- insert output signal
  );
```

Verilog Template:

```
// Module: SIGNED_MULT_18X18
// Description: Verilog instantiation template
// 18-bit X 18-bit embedded signed multiplier (asynchronous)
//
// Device: Virtex-II Family
//------------------------------------------------------------------------
// Instantiation Section
//
MULT18X18 U_MULT18X18
  (
    .A () , // insert input signal #1
    .B () , // insert input signal #2
    .P ()   // insert output signal
  );
```

Using Single-Ended SelectI/O Resources

Summary

The Virtex-II FPGA series includes a highly configurable, high-performance single-ended SelectI/O resource that supports a wide variety of I/O standards. The SelectI/O resource includes a robust set of features, including programmable control of output drive strength, slew rate, and input delay and hold time. Taking advantage of the flexibility of SelectI/O features and the design considerations described in this document can improve and simplify system-level design.

Introduction

As FPGAs continue to grow in size and capacity, the larger and more complex systems designed for them demand an increased variety of I/O standards. Furthermore, as system clock speeds continue to increase, the need for high-performance I/O becomes more important. Chip-to-chip delays have an increasingly substantial impact on overall system speed. The task of achieving the desired system performance is becoming more difficult with the proliferation of low-voltage I/O standards. SelectI/O resolves this potential problem by providing a highly configurable, high-performance alternative to I/O resources used in more conventional programmable devices.

Virtex-II SelectI/O blocks can support up to 19 single-ended I/O standards. Supporting such a variety of I/O standards allows support for a wide variety of applications.

Each Input/Output Block (IOB) includes six registers, two each from the input, output, and 3-state signals within the IOB. These registers are optionally configured as either a D-type flip-flop or as a level-sensitive latch. The purpose of having six registers is to allow designers to design double-data-rate (DDR) logic in the I/O blocks. Each pair of the flip-flop (FF) has different clocks so that the flip-flops can be driven by two clocks with a 180-degree phase shift to achieve DDR. All I/O flip-flops still share the same reset/preset line.

The input buffer has an optional delay element used to guarantee a zero hold time requirement for input signals registered within the IOB.

Virtex-II SelectI/O features also provide dedicated resources for input reference voltage (V_{REF}) and input output source voltage (V_{CCO}), along with a convenient banking system that simplifies board design. Virtex-II inputs and outputs are powered from V_{CCO}. Differential amplifier inputs, such as GTL and SSTL, are powered from V_{REF}.

Fundamentals

Modern bus applications, pioneered by the largest and most influential components in the digital electronics industry, are commonly introduced with a new I/O standard tailored specifically to the needs of that application. The bus I/O standards provide specifications to other vendors who create products designed to interface with these applications. Each standard often has its own specifications for current, voltage, I/O buffering, and termination techniques.

The ability to provide the flexibility and time-to-market advantages of programmable logic is increasingly dependent on the capability of the programmable logic device to support an ever increasing variety of I/O standards.

SelectI/O resources feature highly configurable input and output buffers that provide support for a wide variety of I/O standards. An input buffer can be configured as either a simple buffer or as a differential amplifier input. An output buffer can be configured as either a Push-Pull output or as an Open Drain output. Table 2-29 illustrates all of the

supported single-ended I/O standards in Virtex-II devices. Each buffer type can support a variety of current and voltage requirements.

Table 2-29: **Supported Single-Ended I/O Standards**

I/O Standard	Input Reference Voltage (V_{REF})	Input Source Voltage (V_{CCO})	Output Source Voltage (V_{CCO})	Board Termination Voltage (V_{TT})
LVTTL	N/A	3.3	3.3	N/A
LVCMOS15	N/A	1.5	1.5	N/A
LVCMOS18	N/A	1.8	1.8	N/A
LVCMOS25	N/A	2.5	2.5	N/A
LVCMOS33	N/A	3.3	3.3	N/A
PCI33_3	N/A	3.3	3.3	N/A
PCI66_3	N/A	3.3	3.3	N/A
PCIX	N/A	3.3	3.3	N/A
GTL	0.80	N/A	N/A	1.2
GTL+	1.0	N/A	N/A	1.5
HSTL_I	0.75	N/A	1.5	0.75
HSTL_II	0.75	N/A	1.5	0.75
HSTL_III	0.9	N/A	1.5	1.5
HSTL_IV	0.9	N/A	1.5	1.5
SSTL3_I	1.5	N/A	3.3	1.5
SSTL3_II	1.5	N/A	3.3	1.5
SSTL2_I	1.25	N/A	2.5	1.25
SSTL2_II	1.25	N/A	2.5	1.25
AGP-2X	1.32	N/A	3.3	N/A

Overview of Supported I/O Standards

This section provides a brief overview of I/O standards supported by all Virtex-II devices.

While most I/O standards specify a range of allowed voltages, this document 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 TTL

The low-voltage TTL, or LVTTL, standard is a general purpose EIA/JESDSA standard for 3.3 V applications that use an LVTTL input buffer and a Push-Pull output buffer. This standard requires a 3.3 V input and output source voltage (V_{CCO}), but does not require the use of a reference voltage (V_{REF}) or a termination voltage (V_{TT}).

LVCMOS33 - 3.3 Volt Low-Voltage CMOS

This standard is an extension of the LVCMOS standard (JESD 8.-5). It is used in general purpose 3.3 V applications.The standard requires a 3.3 V input/output source voltage (V_{CCO}), but does not require the use of a reference voltage (V_{REF}) or a termination voltage (V_{TT}).

LVCMOS25 - 2.5 Volt Low-Voltage CMOS

This standard is an extension of the LVCMOS standard (JESD 8.-5). It is used in general purpose 2.5 volts or lower applications. This standard requires a 2.5 V input /output

source voltage (V_{CCO}), but does not require the use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}).

LVCMOS18 - 1.8 Volt Low-Voltage CMOS

This standard is an extension of the LVCMOS standard. It is used in general purpose 1.8 V applications. The use of a reference voltage (V_{REF}) or board termination voltage (V_{TT}) is not required.

LVCMOS15 - 1.5 Volt Low-Voltage CMOS

This standard is an extension of the LVCMOS standard. It is used in general purpose 1.5 V applications. The use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}) is not required.

PCI - Peripheral Component Interface

The PCI standard specifies support for 33 MHz, 66 MHz and 133 MHz PCI bus applications. It uses a 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 3.3 V input output source voltage (V_{CCO}).

GTL -Gunning Transceiver Logic Terminated

The 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 a open Drain output buffer.

GTL+ - 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.

HSTL - High-speed Transceiver Logic

The high-speed Transceiver Logic, or HSTL standard is a general purpose high-speed, 1.5V bus standard sponsored by IBM (EIA/JESD8-6). This standard has four variations or classes. Virtex-II SelectI/O supports all four Classes. This standard requires a Differential Amplifier input buffer and a Push-pull output buffer.

SSTL3 - Stub Series Terminated Logic for 3.3V

The Stub Series Terminated Logic for 3.3V, or SSTL3 standard is a general purpose 3.3V memory bus standard also sponsored by Hitachi and IBM (JESD8-8). This standard has two classes, I and II. Virtex-II SelectI/O supports both classes for the SSTL3 standard. This standard requires a Differential Amplifier input buffer and a Push-Pull output buffer.

SSTL2 - Stub Series Terminated Logic for 2.5V

The Stub Series Terminated Logic for 2.5V, or SSTL2 standard is a general purpose 2.5V memory bus standard also sponsored by Hitachi and IBM (JESD8-8). This standard has two classes, I and II. Virtex-II SelectI/O supports both classes for the SSTL2 standard. This standard requires a Differential Amplifier input buffer and a Push-Pull output buffer

AGP-2X - Advanced Graphics Port

The Intel AGP standard is a 3.3V Advanced Graphics Port-2X bus standard used with the Pentium II processor for graphic applications. This standard requires a Push-Pull output buffer and a Differential Amplifier input buffer.

Library Symbols

The Xilinx library includes an extensive list of symbols designed to provide support for the variety of SelectI/O features. Most of these symbols represent variations of the five generic SelectI/O symbols.

- IBUF (input buffer)
- IBUFG (clock input buffer)
- OBUF (output buffer)
- OBUFT (3-state output buffer)
- IOBUF (input/output buffer)

IBUF

Signals used as inputs to a Virtex-II device must source an input buffer (IBUF) via an external input port. The generic Virtex-II IBUF symbol is shown in Figure 2-75. The extension to the base name defines which I/O standard the IBUF uses. The assumed standard is LVTTL when the generic IBUF has no specified extension.

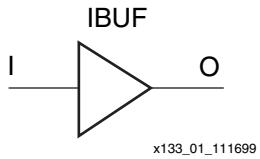


Figure 2-75: **Input Buffer (IBUF) Symbols**

Table 2-30 details variations of the IBUF symbol for single-ended Virtex-II I/O standards:

Table 2-30: **Variations of the IBUF Symbol**

IBUF	IBUF_HSTL_III
IBUF_LVCMOS15	IBUF_HSTL_IV
IBUF_LVCMOS18	IBUF_SSTL2_I
IBUF_LVCMOS25	IBUF_SSTL2_II
IBUF_LVCMOS33	IBUF_SSTL3_I
IBUF_APG	IBUF_SSTL3_II
IBUF_GTL	IBUF_PCI33_3
IBUF_GTLP	IBUF_PCI66_3
IBUF_HSTL_I	IBUF_PCIX
IBUF_HSTL_II	IBUF_AGP

When the IBUF symbol supports an I/O standard that requires a differential amplifier input, the IBUF is automatically configured as a differential amplifier input buffer. The low-voltage I/O standards with a differential amplifier input require an external reference voltage input V_{REF}.

The voltage reference signal is "banked" within the Virtex-II device on a half-edge basis, such that for all packages there are eight independent V_{REF} banks internally. For a representation of the Virtex-II I/O banks, see Figure 2-77. Within each bank approximately one of every six I/O pins is automatically configured as a V_{REF} input. After placing a differential amplifier input signal within a given V_{REF} bank, the same external source must drive all I/O pins configured as a V_{REF} input.

IBUF placement restrictions require that any differential amplifier input signals within a bank be of the same standard. How to specify a specific location for the IBUF via the LOC property is described below. Table 2-31 summarizes compatibility requirements of Virtex-II input standards.

An optional delay element in the input data path is associated with each IBUF. When the IBUF drives a flip-flop within the IOB, the delay element is activated by default to ensure a zero hold-time requirement at the device input pin. The IOBDELAY = NONE property overrides this default, thus reducing the input set-up time, but risking a hold-time requirement.

When the IBUF does not drive a flip-flop within the IOB, the delay element is deactivated by default to provide a shorter input set-up time. To delay the input signal, activate the delay element with the IOBDELAY = BOTH property.

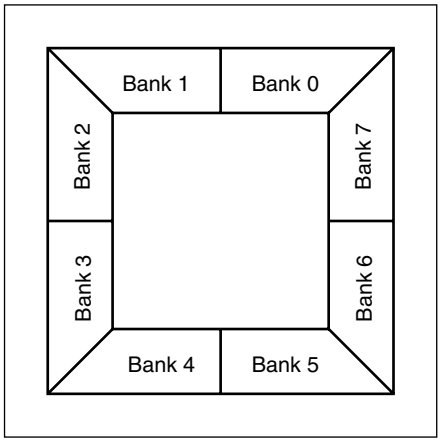


Figure 2-76: **Virtex-II I/O Banks: Top View for Flip-Chip Packages (FF & BF)**

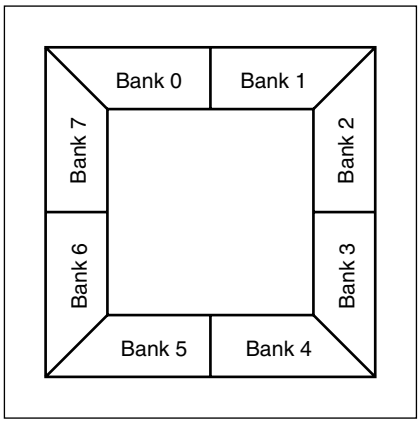


Figure 2-77: **Virtex-II I/O Banks: Top View for Wire-Bond Packages (CS, FG, & BG)**

Table 2-31: **Xilinx Input Standard Compatibility Requirements**

Rule 1	Standards with the same V_{CCO}, and V_{REF} can be placed within the same bank.
Rule 2	Standards that don't require a V_{REF} can be placed within the same bank with the standards that have the same V_{CCO} values

Each bank has its own V_{CCO} and V_{REF} voltage. Details on compatible input standards for each V_{CCO} / V_{REF} voltage combination are available in the **Virtex-II Data Sheet** .

OBUF

An OBUF must drive outputs through an external output port. Figure 2-78 shows the generic output buffer (OBUF) symbol.

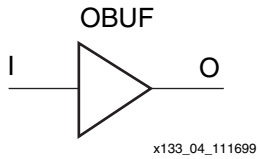


Figure 2-78: **Virtex-II Output Buffer (OBUF) Symbol**

The extension to the base name defines which I/O standard the OBUF uses. With no extension specified for the generic OBUF symbol, the assumed standard is slew rate limited LVTTL with 12mA drive strength.

The LVTTL and LVCMOS OBUFs can additionally support one of two slew rate modes to minimize bus transients. By default, the slew rate for each output buffer is reduced to minimize power bus transients, when switching non-critical signals.

LVTTL and LVCMOS output buffers have selectable drive strengths. The format for these OBUF symbol names is as follows:

OBUF_<slew_rate>_<drive_strength>

<slew_rate> is either F (fast) or S (slow) and <drive_strength> is specified in milliamperes. For LVTTL, LVCMOS25, and LVCMOS33, the supported drive strengths are 2, 4, 6, 8, 12, 16, and 24. For LVCMOS15, and LVCMOS18, the supported drive strengths are 2, 4, 6, 8, 12, and 16.

Table 2-32 details variations of the OBUF symbol.

Table 2-32: **Variations of the OBUF Symbol**

OBUF	OBUF_LVCMOS18_S_2	OBUF_LVCMOS33_S_4
OBUF_S_2	OBUF_LVCMOS18_S_4	OBUF_LVCMOS33_S_6
OBUF_S_4	OBUF_LVCMOS18_S_6	OBUF_LVCMOS33_S_8
OBUF_S_6	OBUF_LVCMOS18_S_8	OBUF_LVCMOS33_S_12
OBUF_S_8	OBUF_LVCMOS18_S_12	OBUF_LVCMOS33_S_16
OBUF_S_12	OBUF_LVCMOS18_S_16	OBUF_LVCMOS33_S_24
OBUF_S_16	OBUF_LVCMOS18_F_2	OBUF_LVCMOS33_F_2
OBUF_S_24	OBUF_LVCMOS18_F_4	OBUF_LVCMOS33_F_4
OBUF_F_2	OBUF_LVCMOS18_F_6	OBUF_LVCMOS33_F_6
OBUF_F_4	OBUF_LVCMOS18_F_8	OBUF_LVCMOS33_F_8
OBUF_F_6	OBUF_LVCMOS18_F_12	OBUF_LVCMOS33_F_12
OBUF_F_8	OBUF_LVCMOS18_F_16	OBUF_LVCMOS33_F_16
OBUF_F_12	OBUF_LVCMOS25	OBUF_LVCMOS33_F_24
OBUF_F_16	OBUF_LVCMOS25_S_2	OBUF_PCI33_3
OBUF_F_24	OBUF_LVCMOS25_S_4	OBUF_PCI66-3
OBUF_LVCMOS15	OBUF_LVCMOS25_S_6	OBUF_PCIX
OBUF_LVCMOS15_S_2	OBUF_LVCMOS25_S_8	OBUF_GTL
OBUF_LVCMOS15_S_4	OBUF_LVCMOS25_S_12	OBUF_GTLP
OBUF_LVCMOS15_S_6	OBUF_LVCMOS25_S_16	OBUF_HSTL_I

Table 2-32: **Variations of the OBUF Symbol *(Continued)***

OBUF_LVCMOS15_S_8	OBUF_LVCMOS25_S_24	OBUF_HSTL_II
OBUF_LVCMOS15_S_12	OBUF_LVCMOS25_F_2	OBUF_HSTL_III
OBUF_LVCMOS15_S_16	OBUF_LVCMOS25_F_4	OBUF_HSTL_IV
OBUF_LVCMOS15_F_2	OBUF_LVCMOS25_F_6	OBUF_SSTL3_I
OBUF_LVCMOS15_F_4	OBUF_LVCMOS25_F_8	OBUF_SSTL3_II
OBUF_LVCMOS15_F_6	OBUF_LVCMOS25_F_12	OBUF_SSTL2_I
OBUF_LVCMOS15_F_8	OBUF_LVCMOS25_F_16	OBUF_SSTL2_II
OBUF_LVCMOS15_F_12	OBUF_LVCMOS25_F_24	OBUF_AGP
OBUF_LVCMOS15_F_16	OBUF_LVCMOS33	
OBUF_LVCMOS18	OBUF_LVCMOS33_S_2	

OBUF placement restrictions require that within a given V_{CCO} bank each OBUF share the same output source drive voltage. Input buffers with the same V_{CCO} and output buffers that do not require V_{CCO} can be placed within any V_{CCO} bank. Table 2-33 summarizes Virtex-II output compatibility requirements. The LOC property can specify a location for the OBUF.

Table 2-33: **Output Standards Compatibility Requirements**

Rule 1	Only outputs with standards which share compatible V_{CCO} can be used within the same bank.
Rule 2	There are no placement restrictions for outputs with standards that do not require a V_{CCO}

Each bank has its own V_{CCO} voltage. Details on compatible output standards for each V_{CCO} voltage combination are available in the **Virtex-II Data Sheet** .

OBUFT

The generic 3-state output buffer OBUFT, shown in Figure 2-79, typically implements 3-state outputs or bidirectional I/O.

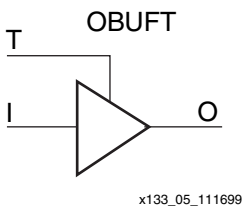


Figure 2-79: **3-State Output Buffer Symbol (OBUFT)**

The extension to the base name defines which I/O standard OBUFT uses. With no extension specified for the generic OBUFT symbol, the assumed standard is slew rate limited LVTTL with 12mA drive strength.

The LVTTL and LVCMOS OBUFTs additionally can support one of two slew rate modes to minimize bus transients. By default, the slew rate for each output buffer is reduced to minimize power bus transients, when switching non-critical signals.

LVTTL and LVCMOS 3-state buffers have selectable drive strengths. The format for these OBUFT symbol names is as follows:

OBUFT_<slew_rate>_<drive_strength>

<slew_rate> is either F(fast) or S(slow) and <drive_strength> is specified in milliamperes. For LVTTL, LVCMOS25, and LVCMOS33, the supported drive strengths are 2, 4, 6, 8, 12, 16, and 24. For LVCMOS15 and LVCMOS18, the supported drive strengths are 2, 4, 6, 8, 12, and 16.

Table 2-34 details variations of the OBUFT symbol.

Table 2-34: **Variations of the OBUFT Symbol**

OBUFT	OBUFT_LVCMOS18_S_2	OBUFT_LVCMOS33_S_4
OBUFT_S_2	OBUFT_LVCMOS18_S_4	OBUFT_LVCMOS33_S_6
OBUFT_S_4	OBUFT_LVCMOS18_S_6	OBUFT_LVCMOS33_S_8
OBUFT_S_6	OBUFT_LVCMOS18_S_8	OBUFT_LVCMOS33_S_12
OBUFT_S_8	OBUFT_LVCMOS18_S_12	OBUFT_LVCMOS33_S_16
OBUFT_S_12	OBUFT_LVCMOS18_S_16	OBUFT_LVCMOS33_S_24
OBUFT_S_16	OBUFT_LVCMOS18_F_2	OBUFT_LVCMOS33_F_2
OBUFT_S_24	OBUFT_LVCMOS18_F_4	OBUFT_LVCMOS33_F_4
OBUFT_F_2	OBUFT_LVCMOS18_F_6	OBUFT_LVCMOS33_F_6
OBUFT_F_4	OBUFT_LVCMOS18_F_8	OBUFT_LVCMOS33_F_8
OBUFT_F_6	OBUFT_LVCMOS18F_12	OBUFT_LVCMOS33_F_12
OBUFT_F_8	OBUFT_LVCMOS18_F_16	OBUFT_LVCMOS33_F_16
OBUFT_F_12	OBUFT_LVCMOS25	OBUFT_LVCMOS33_F_24
OBUFT_F_16	OBUFT_LVCMOS25_S_2	OBUFT_PCI33_3
OBUFT_F_24	OBUFT_LVCMOS25_S_4	OBUFT_PCI66-3
OBUFT_LVCMOS15	OBUFT_LVCMOS25_S_6	OBUFT_PCIX
OBUFT_LVCMOS15_S_2	OBUFT_LVCMOS25_S_8	OBUFT_GTL
OBUFT_LVCMOS15_S_4	OBUFT_LVCMOS25_S_12	OBUFT_GTLP
OBUFT_LVCMOS15_S_6	OBUFT_LVCMOS25_S_16	OBUFT_HSTL_I
OBUFT_LVCMOS15_S_8	OBUFT_LVCMOS25_S_24	OBUFT_HSTL_II
OBUFT_LVCMOS15_S_12	OBUFT_LVCMOS25_F_2	OBUFT_HSTL_III
OBUFT_LVCMOS15_S_16	OBUFT_LVCMOS25_F_4	OBUFT_HSTL_IV
OBUFT_LVCMOS15_F_2	OBUFT_LVCMOS25_F_6	OBUFT_SSTL3_I
OBUFT_LVCMOS15_F_4	OBUFT_LVCMOS25_F_8	OBUFT_SSTL3_II
OBUFT_LVCMOS15_F_6	OBUFT_LVCMOS25_F_12	OBUFT_SSTL2_I
OBUFT_LVCMOS15_F_8	OBUFT_LVCMOS25_F_16	OBUFT_SSTL2_II
OBUFT_LVCMOS15_F_12	OBUFT_LVCMOS25_F_24	OBUFT_AGP
OBUFT_LVCMOS15_F_16	OBUFT_LVCMOS33	
OBUFT_LVCMOS18	OBUFT_LVCMOS33_S_2	

OBUFT placement restrictions require that within a given V_{CCO} bank each OBUFT share the same output source drive voltage. Input buffers with the same V_{CCO} and output buffers that do not require V_{CCO} can be placed within any V_{CCO} bank. The LOC property can specify a location for the OBUFT.

3-state output buffers and bidirectional buffers can have either a weak pull-up resistor, a weak pull-down resistor, or a weak "keeper" circuit. Control this feature by adding the appropriate symbol to the output net of the OBUFT (PULLUP, PULLDOWN, or KEEPER).

The weak "keeper" circuit requires the input buffer within the IOB to sample the I/O signal. Thus, OBUFTs programmed for an I/O standard that requires a V_{REF} have

automatic placement of a V_{REF} in the bank with an OBUFT configured with a weak "keeper" typically implement a bidirectional I/O. In this case, the IBUF (and the corresponding V_{REF}) are placed explicitly.

IOBUF

Use the IOBUF symbol for bidirectional signals that require both an input buffer and a 3-state output buffer with an active High 3-state pin. Figure 2-80 shows the generic input/output IOBUF buffer.

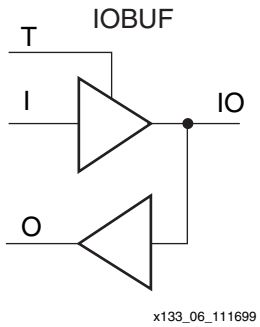


Figure 2-80: **Input/Output Buffer Symbol (IOBUF)**

The extension to the base name defines which I/O standard the IOBUF uses. With no extension specified for the generic IOBUF symbol, the assumed standard is LVTTL input buffer and slew rate limited LVTTL with 12mA drive strength for the output buffer.

The LVTTL and LVCMOS IOBUFs can additionally support one of two slew rate modes to minimize bus transients. By default, the slew rate for each output buffer is reduced to minimize power bus transients, when switching non-critical signals.

LVTTL and LVCMOS output buffers have selectable drive strengths. The format for these OBUF symbol names is as follows:

OBUF_<slew_rate>_<drive_strength>

<slew_rate> is either F (fast) or S (slow) and <drive_strength> is specified in milliamperes. For LVTTL, LVCMOS25 and LVCMOS33, the supported drive strengths are 2, 4, 6, 8, 12, 16, and 24. For LVCMOS15, and LVCMOS18, the supported drive strengths are 2, 4, 6, 8, 12, and 16. Table 2-35 details variations of the IOBUF symbol.

Table 2-35: **Variations of the IOBUF Symbol**

IOBUF	IOBUF_LVCMOS18_S_2	IOBUF_LVCMOS33_S_4
IOBUF_S_2	IOBUF_LVCMOS18_S_4	IOBUF_LVCMOS33_S_6
IOBUF_S_4	IOBUF_LVCMOS18_S_6	IOBUF_LVCMOS33_S_8
IOBUF_S_6	IOBUF_LVCMOS18_S_8	IOBUF_LVCMOS33_S_12
IOBUF_S_8	IOBUF_LVCMOS18_S_12	IOBUF_LVCMOS33_S_16
IOBUF_S_12	IOBUF_LVCMOS18_S_16	IOBUF_LVCMOS33_S_24
IOBUF_S_16	IOBUF_LVCMOS18_F_2	IOBUF_LVCMOS33_F_2
IOBUF_S_24	IOBUF_LVCMOS18_F_4	IOBUF_LVCMOS33_F_4
IOBUF_F_2	IOBUF_LVCMOS18_F_6	IOBUF_LVCMOS33_F_6
IOBUF_F_4	IOBUF_LVCMOS18_F_8	IOBUF_LVCMOS33_F_8
IOBUF_F_6	IOBUF_LVCMOS18F_12	IOBUF_LVCMOS33_F_12
IOBUF_F_8	IOBUF_LVCMOS18_F_16	IOBUF_LVCMOS33_F_16

Table 2-35: **Variations of the IOBUF Symbol *(Continued)***

IOBUF_F_12	IOBUF_LVCMOS25	IOBUF_LVCMOS33_F_24
IOBUF_F_16	IOBUF_LVCMOS25_S_2	IOBUF_PCI33_3
IOBUF_F_24	IOBUF_LVCMOS25_S_4	IOBUF_PCI66-3
IOBUF_LVCMOS15	IOBUF_LVCMOS25_S_6	IOBUF_PCIX
IOBUF_LVCMOS15_S_2	IOBUF_LVCMOS25_S_8	IOBUF_GTL
IOBUF_LVCMOS15_S_4	IOBUF_LVCMOS25_S_12	IOBUF_GTLP
IOBUF_LVCMOS15_S_6	IOBUF_LVCMOS25_S_16	IOBUF_HSTL_I
IOBUF_LVCMOS15_S_8	IOBUF_LVCMOS25_S_24	IOBUF_HSTL_II
IOBUF_LVCMOS15_S_12	IOBUF_LVCMOS25_F_2	IOBUF_HSTL_III
IOBUF_LVCMOS15_S_16	IOBUF_LVCMOS25_F_4	IOBUF_HSTL_IV
IOBUF_LVCMOS15_F_2	IOBUF_LVCMOS25_F_6	IOBUF_SSTL3_I
IOBUF_LVCMOS15_F_4	IOBUF_LVCMOS25_F_8	IOBUF_SSTL3_II
IOBUF_LVCMOS15_F_6	IOBUF_LVCMOS25_F_12	IOBUF_SSTL2_I
IOBUF_LVCMOS15_F_8	IOBUF_LVCMOS25_F_16	IOBUF_SSTL2_II
IOBUF_LVCMOS15_F_12	IOBUF_LVCMOS25_F_24	IOBUF_AGP
IOBUF_LVCMOS15_F_16	IOBUF_LVCMOS33	
IOBUF_LVCMOS18	IOBUF_LVCMOS33_S_2	

When the IOBUF symbol supports an I/O standard that requires a differential amplifier input, IOBUF is automatically configured as a differential amplifier input buffer. Low-voltage I/O standards with a differential amplifier input require an external reference voltage input V_{REF}.

The voltage reference signal is "banked" within the Virtex-II device on a half-edge basis, such that for all packages there are eight independent V_{REF} banks internally. For a representation of the Virtex-II I/O banks, see Figure 2-77. Within each bank approximately one of every twelve I/O pins is automatically configured as a V_{REF} input. After placing a differential amplifier input signal within a given V_{REF} bank, the same external source must drive all I/O pins configured as a V_{REF} input.

IOBUF placement restrictions require any differential amplifier input signals within a bank be of the same standard.

Additional restrictions on Virtex-II SelectI/O IOBUF placement require that within a given V_{CCO} bank each IOBUF share the same output source drive voltage. Input buffers with the same V_{CCO} and output buffers that do not require V_{CCO} can be placed within any V_{CCO} bank. The LOC property can specify a location for the OBUF.

An optional delay element is associated with the input path in each IOBUF. When the IOBUF drives an input flip-flop within the IOB, the delay element is activated by default to ensure the zero hold-time requirement. Override this default with the IOBDELAY = NONE property.

In the case when the IOBUF does not drive an input flip-flop within the IOB, the delay element is deactivated by default to provide higher performance. To delay the input signal, deactivate the delay element with the IOBDELAY = BOTH property.

3-state output buffers and bidirectional buffers can have a weak pull-up resistor, a weak pull-down resistor, or a weak "keeper" circuit. Control this feature by adding the appropriate symbol to the output net of the IOBUF (PULLUP, PULLDOWN, or KEEPER).

SelectI/O Properties

Access to some SelectI/O features (for example, location constraints, input delay, output drive strength, and slew rate) is available through properties associated with these features.

Input Delay Properties

An optional delay element is associated with the input path in each IBUF. When the IBUF drives an input flip-flop within the IOB, the delay element activates by default to ensure the zero hold-time requirement. Override this default with the IOBDELAY = NONE property.

In the case when the IBUF does not drive an input flip-flop within the IOB, the delay element is deactivated by default to provide higher performance. To delay the input signal, activate the delay element with the IOBDELAY = BOTH property.

IOB Flip-Flop/Latch Properties

The Virtex-II series I/O block (IOB) includes two optional registers on the input path, two optional registers on the output path, and two optional registers on the 3-state control pin. The design implementation software automatically takes advantage of these registers when the following option for the MAP program is specified.

```
Map -pr b <filename>
```

Alternatively, the IOB = TRUE property can be placed on a register to force the mapper to place the register in an IOB.

The two registers for each path makes designing double-data-rate (DDR) logic much simpler. Each pair of the registers has separate clock inputs, which can be driven by either the positive edge or the negative edge of the clock. Users can use both edges of the clocks to clock data in and out from the IOB. For details on DDR, see "Using Double-Data-Rate (DDR) I/O" on page 303.

Location Constraints

Specify the location of each SelectI/O symbol with the location constraint LOC attached to the SelectI/O symbol. The external port identifier indicates the value of the location constrain. The format of the port identifier depends on the package chosen for the specified design.

The LOC properties use the following form:

- LOC=A42;
- LOC=P37;

Output Slew Rate Property

As mentioned above, a variety of symbol names provide the option of choosing the desired slew rate for the output buffers. In the case of the LVTTL or LVCMOS output buffers (OBUF, OBUFT, and IOBUF), slew rate control can be alternatively programmed with the SLEW = property. By the default, the slew rate for each output buffer is reduced to minimize power bus transients when switching non-critical signals. The SLEW = property has one of the two following values:

- SLEW = SLOW
- SLEW = FAST

Output Drive Strength Property

The desired output drive strength can be additionally specified by choosing the appropriate library symbol. The Xilinx library also provides an alternative method for specifying this feature. For the LVTTL, and LVCMOS output buffers (OBUF, OBUFT, and

IOBUF), the desired drive strength can be specified with the DRIVE = property. This property could have one of the following values:

- DRIVE = 2
- DRIVE = 4
- DRIVE = 6
- DRIVE = 8
- DRIVE = 12
- DRIVE = 16
- DRIVE = 24

Design Considerations

Reference Voltage (V_{REF}) Pins

Low-voltage I/O standards with a differential amplifier input buffer require an input reference voltage (V_{REF}). Provide the V_{REF} as an external signal to the device.

The voltage reference signal is "banked" within the Virtex-II device on a half-edge basis such that for all packages there are eight independent V_{REF} banks internally. See Figure 2-77 for a representation of the Virtex-II I/O banks. Within each bank approximately one of every twelve I/O pins is automatically configured as a V_{REF} input. After placing a differential amplifier input signal within a given V_{REF} bank, the same external source must drive all I/O pins configured as a V_{REF} input.

Within each V_{REF} bank, any input buffers that require a V_{REF} signal must be of the same type. Output buffers that have the same V_{CCO} values as the input buffers can be placed within the same V_{REF} bank.

Output Drive Source Voltage (V_{CCO}) Pins

Many of the low-voltage I/O standards supported by SelectI/O devices require a different output drive source voltage (V_{CCO}). As a result each device can often have to support multiple output drive source voltages.

Output buffers within a given V_{CCO} bank must share the same output drive source voltage. Input buffers for LVTTL, LVCMOS15, LVCMOS18, LVCMOS25, LVCMOS33, PCI33_3, PCI66_3, PCIX use the V_{CCO} voltage for input V_{CCO} voltage.

Transmission Line Effects

The delay of an electrical signal along a wire is dominated by the rise and fall times when the signal travels a short distance. Transmission line delays vary with inductance and capacitance. But a well-designed board can experience delays of approximately 180ps per inch. Transmission line effects, or reflections, typically start at 1.5" for fast (1.5ns) rise and fall times. Poor (or non-existent) termination or changes in the transmission line impedance cause these reflections and can cause additional delay in longer traces. As a system speeds continue to increase, the effect of I/O delays can become a limiting factor and therefore transmission line termination becomes increasingly more important.

Termination Techniques

A variety of termination techniques reduce the impact of transmission line effects.

The following are output termination techniques:

- None
- Series
- Parallel (Shunt)
- Series and Parallel (Series-Shunt)

The following are input termination techniques:

- None
- Parallel (Shunt)

These termination techniques can be applied in any combination. A generic example of each combination of termination methods appears in Figure 2-81.

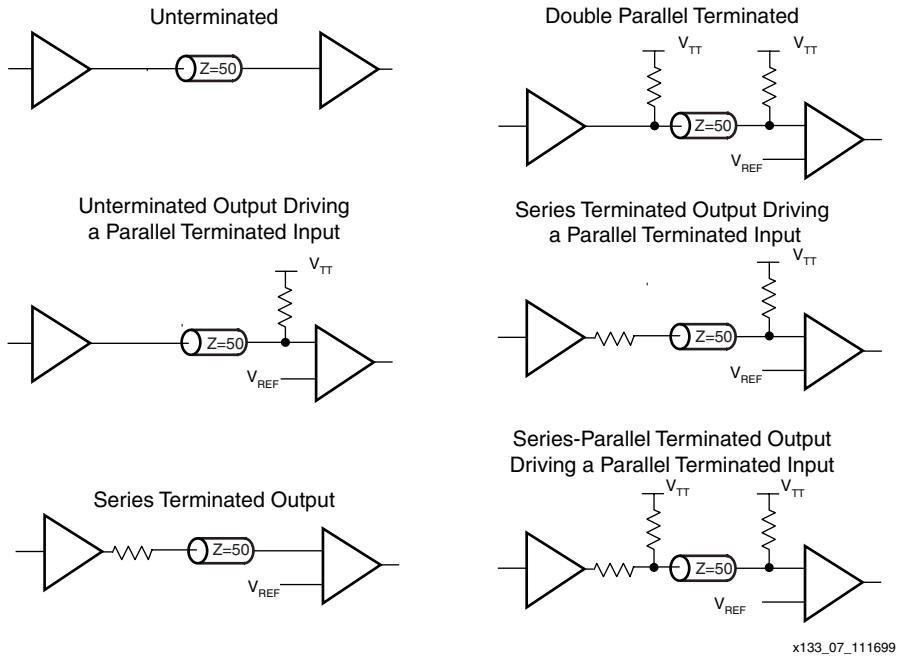


Figure 2-81: **Overview of Standard Input and Output Termination Methods**

Simultaneous Switching Guidelines

Ground bounce can occur with high-speed digital I_{CS} when multiple outputs change states simultaneously, causing undesired transient behavior on an output or in the internal logic. This problem is also referred to as the Simultaneous Switching Output (SSO) problem.

Ground bounce is primarily due to current changes in the combined inductance of ground pins, bond wires, and group metallization. The IC internal ground level deviates from the external system ground level for a short duration (a few nanoseconds) after multiple outputs change state simultaneously.

Ground bounce affects stable low outputs and all inputs because they interpret the incoming signal by comparing it to the internal ground. If the ground bounce amplitude exceeds the actual instantaneous noise margin, then a non-changing input can be interpreted as a short pulse with a polarity opposite to the ground bounce. Table 2-36 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 2-37 for the number of effective output power/ground pairs for each Virtex-II device and package combination.

Table 2-36: **Guidelines for Max Number of Simultaneously Switching Outputs per Power/Ground Pair**

Standard	Package			
	FG,BG,FF,BF	CS	XC2V40-FG	XC2V40-CS
LVTTL2_slow	68	51	51	34
LVTTL4_slow	41	31	31	21
LVTTL6_slow	29	22	22	15
LVTTL8_slow	22	17	17	11
LVTTL12_slow	15	11	11	8
LVTTL16_slow	11	8	8	6
LVTTL24_slow	7	5	5	4
LVTTL2_fast	40	30	30	20
LVTTL4_fast	24	18	18	12
LVTTL6_fast	17	13	13	9
LVTTL8_fast	13	10	10	7
LVTTL12_fast	10	8	8	5
LVTTL16_fast	8	6	6	4
LVTTL24_fast	5	4	4	3
LVDCI_15 50 Ω impedance	10	8	8	5
LVDCI_DV2_15 25 Ω impedance	5	4	4	3
LVCMOS15_2_slow	51	38	38	26
LVCMOS15_4_slow	31	23	23	16
LVCMOS15_6_slow	22	17	17	11
LVCMOS15_8_slow	17	13	13	9
LVCMOS15_12_slow	11	8	8	6
LVCMOS15_16_slow	8	6	6	4
LVCMOS15_2_fast	30	23	23	15
LVCMOS15_4_fast	18	14	14	9
LVCMOS15_6_fast	13	10	10	7
LVCMOS15_8_fast	10	8	8	5
LVCMOS15_12_fast	8	6	6	4
LVCMOS15_16_fast	6	5	5	3
LVDCI_18 50 Ω impedance	11	8	8	6
LVDCI_DV2_18 25 Ω impedance	6	4	4	3
LVCMOS18_2_slow	58	44	44	29
LVCMOS18_4_slow	35	26	26	18

Table 2-36: **Guidelines for Max Number of Simultaneously Switching Outputs per Power/Ground Pair *(Continued)***

Standard	Package			
	FG,BG,FF,BF	CS	XC2V40-FG	XC2V40-CS
LVCMOS18_6_slow	25	19	19	13
LVCMOS18_8_slow	19	14	14	10
LVCMOS18_12_slow	13	10	10	7
LVCMOS18_16_slow	10	8	8	5
LVCMOS18_2_fast	34	26	26	17
LVCMOS18_4_fast	20	15	15	10
LVCMOS18_6_fast	15	11	11	8
LVCMOS18_8_fast	11	8	8	6
LVCMOS18_12_fast	9	7	7	5
LVCMOS18_16_fast	7	5	5	4
LVDCI_25 50 Ω impedance	13	10	10	7
LVDCI_DV2_25 25 Ω impedance	7	5	5	3
LVCMOS25_2_slow	68	51	51	34
LVCMOS25_4_slow	41	31	31	21
LVCMOS25_6_slow	29	22	22	15
LVCMOS25_8_slow	22	17	17	11
LVCMOS25_12_slow	15	11	11	8
LVCMOS25_16_slow	11	8	8	6
LVCMOS25_24_slow	7	5	5	4
LVCMOS25_2_fast	40	30	30	20
LVCMOS25_4_fast	24	18	18	12
LVCMOS25_6_fast	17	13	13	9
LVCMOS25_8_fast	13	10	10	7
LVCMOS25_12_fast	10	8	8	5
LVCMOS25_16_fast	8	6	6	4
LVCMOS25_24_fast	5	4	4	2
LVDCI_33 50 Ω impedance	13	10	10	7
LVDCI_DV2_33 25 Ω impedance	7	5	5	3
LVCMOS33_2_slow	68	51	51	34
LVCMOS33_4_slow	41	31	31	21
LVCMOS33_6_slow	29	22	22	15
LVCMOS33_8_slow	22	17	17	11

Table 2-36: **Guidelines for Max Number of Simultaneously Switching Outputs per Power/Ground Pair *(Continued)***

Standard	Package			
	FG,BG,FF,BF	CS	XC2V40-FG	XC2V40-CS
LVCMOS33_12_slow	15	11	11	8
LVCMOS33_16_slow	11	8	8	6
LVCMOS33_24_slow	7	5	5	4
LVCMOS33_2_fast	40	30	30	20
LVCMOS33_4_fast	24	18	18	12
LVCMOS33_6_fast	17	13	13	9
LVCMOS33_8_fast	13	10	10	7
LVCMOS33_12_fast	10	8	8	5
LVCMOS33_16_fast	8	6	6	4
LVCMOS33_24_fast	5	4	4	2
PCI33/66/X	8	6	6	4
GTL	4	3	3	2
GTL_DCI	3	2	2	1
GTL+	4	3	3	2
GTL+_DCI	3	2	2	1
HSTLI	20	15	15	10
HSTLI_DCI	20	15	15	10
HSTLII	10	8	8	5
HSTLII_DCI	7	5	5	4
HSTLIII	8	6	6	4
HSTLIII_DCI	8	6	6	4
HSTLIV	4	3	3	2
HSTLIV_DCI	4	3	3	2
SSTL2I	15	11	11	8
SSTL2I_DCI	15	11	11	8
SSTL2II	10	8	8	5
SSTL2II_DCI	5	4	4	3
SSTL3I	12	9	9	6
SSTL3I_DCI	12	9	9	6
SSTL3II	8	6	6	4
SSTL3II_DCI	4	3	3	2
AGP	9	7	7	5

Table 2-37: **Virtex-II Equivalent Power/Ground Pairs per Bank**

Package	XC2V Device											
	40	80	250	500	1000	1500	2000	3000	4000	6000	8000	10000
CS144[1]	1	1	1									
FG256[1]	1	2	3	3	3							
FG456[1]			3	4	5							
FG676[1]						6	7	7				
BG575[1]					5	6	6					
BG728[1]							7	8				
FF896[2]					7	8	10					
FF1152[2]								11	13	13	13	13
FF1517[2]									14	17	17	17
BF957[2]							10	10	10	11	11	11

Notes:
1. Wire-bond only.
2. Flip-chip only.

Application Example

Creating a design with the SelectI/O feature requires either assignment of the `IOSTANDARD` attribute in the constraint file or instantiation of the desired library symbol within the design code.

To enter the `IOSTANDARD` attribute in the constraint file (UCF file), the following syntax can be used:

```
NET <pad net name> IOSTANDARD=<the name of the standard>
```

For example, to enter PCIX standard, use

```
NET <pad net name> IOSTANDARD=PCIX;
```

To instantiate a library symbol in the HDL code, use the proper input or output buffer name, and follow the standard syntax of instantiation.

For example, to instantiate a GTL input buffer in VHDL, the following syntax can be used:

```
GTL_buffer : IBUF_GTL port map (I=>data_in, O=>data_gtl_in);
```

At the board level, designers need to know the termination techniques required for each I/O standard.

This section describes some common application examples illustrating the termination techniques recommended by each of the single-ended standard supported by the SelectI/O features.

Termination Example

Circuit examples involving typical termination techniques for each of the SelectI/O standards follow. For a full range of accepted values for the DC voltage specifications for each standard, refer to the table associated with each figure.

The resistors used in each termination technique example and the transmission lines depicted represent board level components and are not meant to represent components on the device.

GTL

A sample circuit illustrating a valid termination technique for GTL is shown in Figure 2-82.

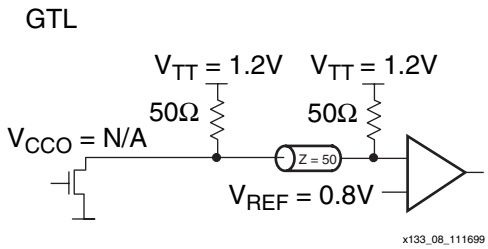


Figure 2-82: **GTL Terminated**

Table 2-38 lists DC voltage specifications.

Table 2-38: **GTL 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
$V_{IH} \geq V_{REF} + 0.05$	0.79	0.85	-
$V_{IL} \leq V_{REF} - 0.05$	-	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 0.4 V	32	-	-
I_{OL}at V_{OL} (mA) at 0.2 V	-	-	40

Notes:

1. N must be greater than or equal to 0.653 and less than or equal to 0.68.

GTL +

Figure 2-83 shows a sample circuit illustrating a valid termination technique for GTL+.

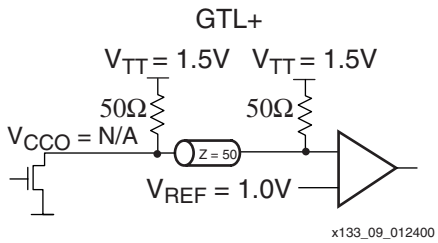


Figure 2-83: **GTL+ Terminated**

Table 2-39 lists DC voltage specifications.

Table 2-39: **GTL+ Voltage Specifications**

Parameter	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
$V_{IH} \geq V_{REF} + 0.1$	0.98	1.1	-
$V_{IL} \leq V_{REF} - 0.1$	-	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.

HSTL Class I

Figure 2-88 shows a sample circuit illustrating a valid termination technique for HSTL_I.

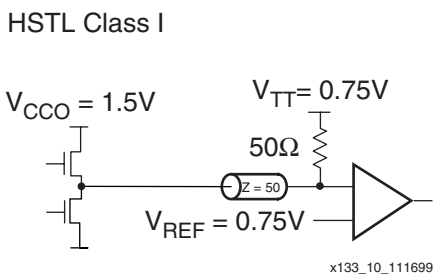


Figure 2-84: **Terminated HSTL Class I**

Table 2-44 lists DC voltage specifications.

Table 2-40: **HSTL Class I Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.40	1.50	1.60
V_{REF}	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)	–8	-	-
I_{OL}at V_{OL} (mA)	8	-	-

HSTL Class II

Figure 2-89 shows a sample circuit illustrating a valid termination technique for HSTL_II.

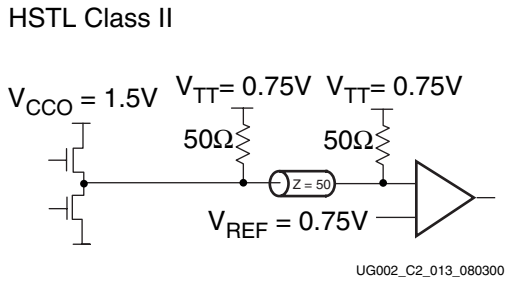


Figure 2-85: **Terminated HSTL Class II**

Table 2-45 lists DC voltage specifications.

Table 2-41: **HSTL Class II Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.40	1.50	1.60
V_{REF} (1)	-	0.75	-
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)	–16	-	-
I_{OL} at V_{OL} (mA)	16	-	-

Notes:

1. 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 III

Figure 2-90 shows a sample circuit illustrating a valid termination technique for HSTL_III.

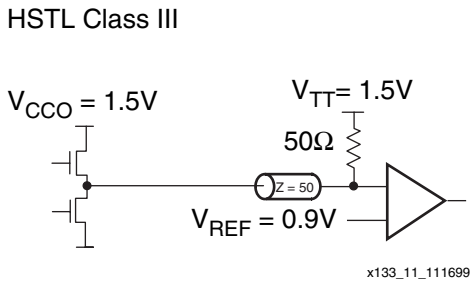


Figure 2-86: **Terminated HSTL Class III**

Table 2-46 lists DC voltage specifications.

Table 2-42: **HSTL Class III Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.40	1.50	1.60
V_{REF} (1)	-	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)	–8	-	-
I_{OL} at V_{OL} (mA)	24	-	-

Notes:

1. 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 2-91 shows a sample circuit illustrating a valid termination technique for HSTL_IV.

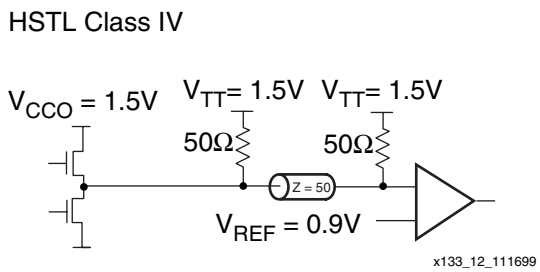


Figure 2-87: **Terminated HSTL Class IV**

Table 2-47 lists DC voltage specifications.

Table 2-43: **HSTL Class IV Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.40	1.50	1.60
V_{REF}	-	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)	–8	-	-
I_{OL} at V_{OL} (mA)	48	-	-

Notes:

1. 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 2-88 shows a sample circuit illustrating a valid termination technique for HSTL_I.

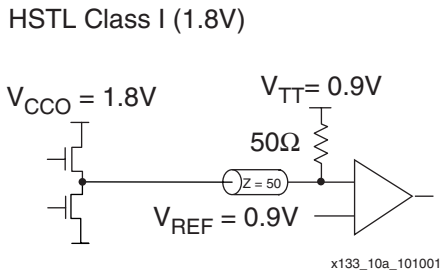


Figure 2-88: **Terminated HSTL Class I (1.8V)**

Table 2-44 lists DC voltage specifications.

Table 2-44: **HSTL Class I (1.8V) Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.7	1.8	1.9
V_{REF}	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)	–8	-	-
I_{OL}at V_{OL} (mA)	8	-	-

HSTL Class II (1.8V)

Figure 2-89 shows a sample circuit illustrating a valid termination technique for HSTL_II.

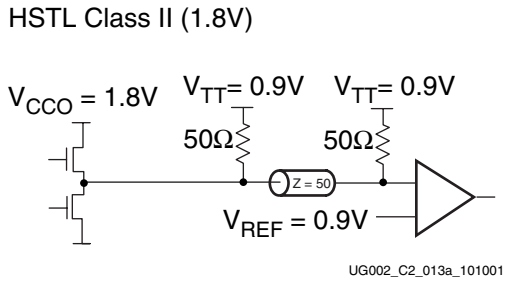


Figure 2-89: **Terminated HSTL Class II (1.8V)**

Table 2-45 lists DC voltage specifications.

Table 2-45: **HSTL Class II (1.8V) Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.7	1.8	1.9
V_{REF} (1)	-	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)	–16	-	-
I_{OL} at V_{OL} (mA)	16	-	-

Notes:

1. 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 III (1.8V)

Figure 2-90 shows a sample circuit illustrating a valid termination technique for HSTL_III.

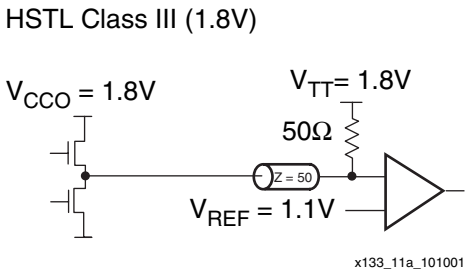


Figure 2-90: **Terminated HSTL Class III (1.8V)**

Table 2-46 lists DC voltage specifications.

Table 2-46: **HSTL Class III (1.8V) Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.7	1.8	1.9
V_{REF} (1)	-	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)	–8	-	-
I_{OL} at V_{OL} (mA)	24	-	-

Notes:

1. 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 2-91 shows a sample circuit illustrating a valid termination technique for HSTL_IV.

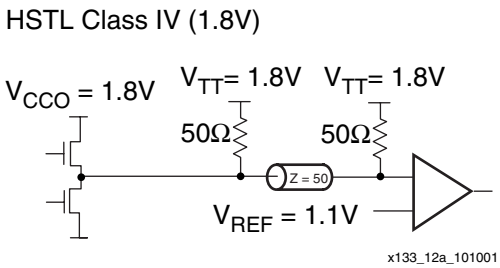


Figure 2-91: **Terminated HSTL Class IV (1.8V)**

Table 2-47 lists DC voltage specifications.

Table 2-47: **HSTL Class IV (1.8V) Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.7	1.8	1.9
V_{REF}	-	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)	–8	-	-
I_{OL} at V_{OL} (mA)	48	-	-

Notes:

1. 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.

SSTL3_I

Figure 2-92 shows a sample circuit illustrating a valid termination technique for SSTL3_I.

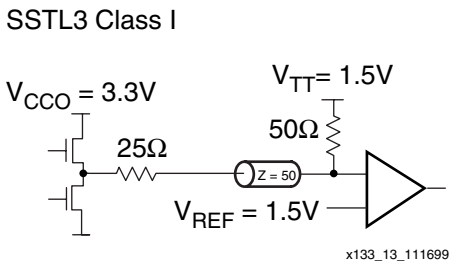


Figure 2-92: **Terminated SSTL3_I**

Table 2-48 lists DC voltage specifications.

Table 2-48: **SSTL3_I Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
$V_{REF} = 0.45 \times V_{CCO}$	1.3	1.5	1.7
$V_{TT} = V_{REF}$	1.3	1.5	1.7
$V_{IH} \geq V_{REF} + 0.2$	1.5	1.7	3.9(1)
$V_{IL} \leq V_{REF} - 0.2$	–0.3(2)	1.3	1.5
$V_{OH} \geq V_{REF} + 0.6$	1.9	2.1	-
$V_{OL} \leq V_{REF} - 0.6$	-	0.9	1.1
I_{OH} at V_{OH} (mA)	–8	-	-
I_{OL} at V_{OL} (mA)	8	-	-

Notes:
1. V_{IH} maximum is V_{CCO} + 0.3
2. V_{IL} minimum does not conform to the formula

SSTL3_II

Figure 2-93 shows a sample circuit illustrating a valid termination technique for SSTL3_II.

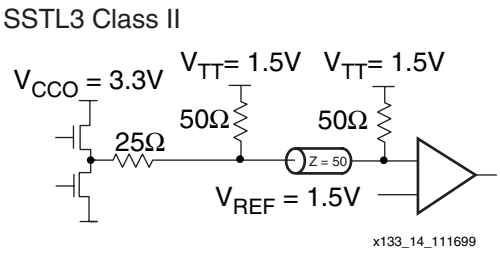


Figure 2-93: **Terminated SSTL3_II**

Table 2-49 lists DC voltage specifications.

Table 2-49: **SSTL3_II Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
$V_{REF} = 0.45 \times V_{CCO}$	1.3	1.5	1.7
$V_{TT} = V_{REF}$	1.3	1.5	1.7
$V_{IH} \geq V_{REF} + 0.2$	1.5	1.7	3.9(1)
$V_{IL} \leq V_{REF} - 0.2$	–0.3(2)	1.3	1.5
$V_{OH} \geq V_{REF} + 0.8$	2.1	2.3	-
$V_{OL} \leq V_{REF} - 0.8$	-	0.7	0.9
I_{OH} at V_{OH} (mA)	–16	-	-
I_{OL} at V_{OL} (mA)	16	-	-

Notes:
1. V_{IH} maximum is V_{CCO} + 0.3
2. V_{IL} minimum does not conform to the formula

SSTL2_I

Figure 2-94 shows a sample circuit illustrating a valid termination technique for SSTL2_I.

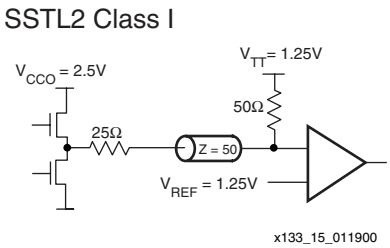


Figure 2-94: **Terminated SSTL2_I**

Table 2-50 lists DC voltage specifications.

Table 2-50: **SSTL2_I Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
$V_{REF} = 0.5 \times V_{CCO}$	1.15	1.25	1.35
$V_{TT} = V_{REF} + N$(1)	1.11	1.25	1.39
$V_{IH} \geq V_{REF} + 0.18$	1.33	1.43	3.0(2)
$V_{IL} \leq V_{REF} - 0.18$	–0.3(3)	1.07	1.17
$V_{OH} \geq V_{REF} + 0.61$	1.76	1.82	1.96
$V_{OL} \leq V_{REF} - 0.61$	0.54	0.64	0.74
I_{OH} at V_{OH} (mA)	–7.6	-	-
I_{OL} at V_{OL} (mA)	7.6	-	-

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.

SSTL2_II

Figure 2-95 shows a sample circuit illustrating a valid termination technique for SSTL2_II.

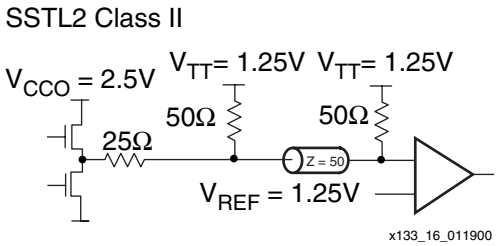


Figure 2-95: **Terminated SSTL2_II**

Table 2-51 lists DC voltage specifications.

Table 2-51: **SSTL2_II Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
$V_{REF} = 0.5 \times V_{CCO}$	1.15	1.25	1.35
$V_{TT} = V_{REF} + N^{(1)}$	1.11	1.25	1.39
$V_{IH} \geq V_{REF} + 0.18$	1.33	1.43	$3.0^{(2)}$
$V_{IL} \leq V_{REF} - 0.18$	$-0.3^{(3)}$	1.07	1.17
$V_{OH} \geq V_{REF} + 0.8$	1.95	2.05	-
$V_{OL} \leq V_{REF} - 0.8$	-	0.45	0.55
I_{OH} at V_{OH} (mA)	–15.2	-	-
I_{OL} at V_{OL} (mA)	15.2	-	-

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.

PCI33_3, PCI66_3, and PCIX

Table 2-52 lists DC voltage specifications.

Table 2-52: **PCI33_3, PCI66_3, and PCIX 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.

LVTTL

Table 2-53 lists DC voltage specifications.

Table 2-53: **LVTTL Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
V_{REF}	-	-	-
V_{TT}	-	-	-
V_{IH}	2.0	-	3.6
V_{IL}	–0.5	-	0.8
V_{OH}	2.4	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–24	-	-
I_{OL}at V_{OL} (mA)	24	-	-

Notes:
1. $V_{OL}$and V_{OH} for lower drive currents are sample tested.

LVCMOS15

Table 2-54 lists DC voltage specifications.

Table 2-54: **LVCMOS15 Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	-	1.5	-
V_{REF}	-	-	-
V_{TT}	-	-	-
$V_{IH} = 0.7 \times V_{CCO}$	1.05	-	1.65
$V_{IL} = 0.2 \times V_{CCO}$	–0.5	-	0.3
$V_{OH} = V_{CCO} - 0.45$	-	1.05	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–16	-	-
I_{OL}at V_{OL} (mA)	16	-	-

LVCMOS18

Table 2-55 lists DC voltage specifications.

Table 2-55: **LVCMOS18 Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	1.7	1.8	1.9
V_{REF}	-	-	-
V_{TT}	-	-	-
$V_{IH} = 0.7 \times V_{CCO}$	1.19	-	1.95
$V_{IL} = 0.2 \times V_{CCO}$	–0.5	-	0.4
$V_{OH} = V_{CCO} - 0.4$	1.3	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–16	-	-
I_{OL}at V_{OL} (mA)	16	-	-

LVCMOS25

Table 2-56 lists DC voltage specifications.

Table 2-56: **LVCMOS25 Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
V_{REF}	-	-	-
V_{TT}	-	-	-
V_{IH}	1.7	-	2.7
V_{IL}	–0.5	-	0.7
V_{OH}	1.9	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–24	-	-
I_{OL}at V_{OL} (mA)	24	-	-

LVCMOS33

Table 2-57 lists DC voltage specifications.

Table 2-57: **LVCMOS33 Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
V_{REF}	-	-	-
V_{TT}	-	-	-
V_{IH}	2.0	-	3.6
V_{IL}	–0.5	-	0.8
V_{OH}	2.6	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–24	-	-
I_{OL}at V_{OL} (mA)	24	-	-

AGP-2X

Table 2-58 lists DC voltage specifications.

Table 2-58: **AGP-2X Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
$V_{REF} = N \times V_{CCO}$(1)	1.17	1.32	1.48
V_{TT}	-	-	-
$V_{IH} \geq V_{REF} + 0.2$	1.37	1.52	-
$V_{IL} \leq V_{REF} - 0.2$	-	1.12	1.28
$V_{OH} = 0.9 \times V_{CCO}$	2.7	3.0	-
$V_{OL} = 0.1 \times V_{CCO}$	-	0.33	0.36
I_{OH} at V_{OH} (mA)	Note 2	-	-
I_{OL}at V_{OL} (mA)	Note 2	-	-

Notes:

1. N must be greater than or equal to 0.39 and less than or equal to 0.41.
2. Tested according to the relevant specification.

Using Digitally Controlled Impedance (DCI)

Introduction

As FPGAs get bigger and system clock speeds get faster, PCB board design and manufacturing has become more difficult. With ever faster edge rates, maintaining signal integrity becomes a critical issue. Designers must make sure that most PC board traces are terminated properly 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 the increase in the device I/O counts, adding resistors close to the device pins increases the board area and component count and might even be physically impossible. To address these issues and to achieve better signal integrity, Xilinx developed a new I/O technology for the Virtex-II device family, Digitally Controlled Impedance (DCI).

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 termination resistors.

DCI provides the 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 completely.

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 (configuration option).

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. Check the Virtex-II pinout for 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, which compensates for process variations, is done during the device startup sequence. The second phase, which maintains the impedance in response to temperature and supply voltage changes, begins immediately after the first phase and continues indefinitely, even while the part is operating. By default, the DONE pin does not go High until the impedance adjustment process has completed.

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 he following types of on-chip terminations:

1. Termination to V_{CCO} (Single Termination)
2. Termination to $V_{CCO}/2$ (Split Termination, Thevenin equivalent)

For bidirectional operation, both ends of the line can be DCI-terminated permanently:

1. Termination to V_{CCO} (Single Termination)
2. 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 LVTTL, LVCMOS, etc., must have a drive impedance that matches the characteristic impedance of the driven line. DCI can provide a controlled impedance output drivers that eliminate reflections without an external source termination. The impedance is set by the external reference resistors, whose resistance should be equal to the trace impedance. Figure 2-96 illustrates a controlled impedance driver inside Virtex-II device. The DCI I/O standards that support Controlled Impedance Driver are: LVDCI_15, LVDCI_18, LVDCI_25, and LVDCI_33.

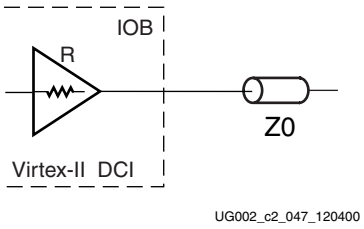


Figure 2-96: **Controlled Impedance Driver**

Controlled Impedance Driver With Half Impedance (Source Termination)

DCI can also provide drivers with one half of the impedance of the reference resistors. The DCI I/O standards that support controlled impedance driver with half impedance are: LVDCI_DV2_15, LVDCI_DV2_18, LVDCI_DV2_25, and LVDCI_DV2_33

Figure 2-97 illustrates a controlled driver with half impedance inside a Virtex-II device.

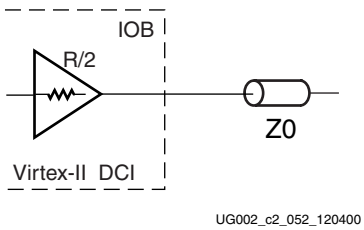


Figure 2-97: **Controlled Impedance Driver With Half Impedance**

Termination to V_{CCO} (Single Termination)

Some I/O standards, such as HSTL Class III, IV, etc., require an input termination to V_{CCO}. See Figure 2-98.

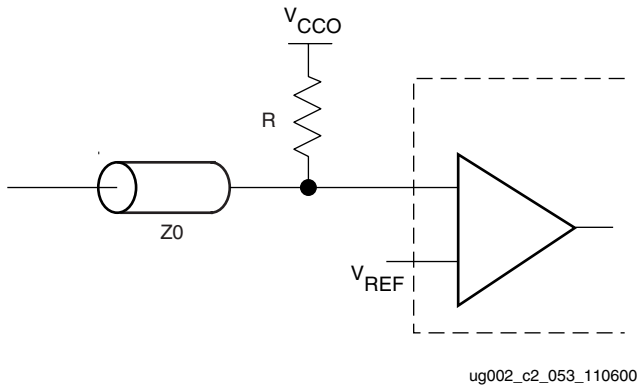


Figure 2-98: **Single Termination Without DCI**

DCI can provide this termination to V_{CCO} using single termination. The termination resistance is set by the reference resistors. For GTL and HSTL standards, they should be controlled by 50-ohm reference resistors. The DCI I/O standards that support single termination are: GTL_DCI, GTLP_DCI, HSTL_III_DCI, and HSTL_IV_DCI.

Figure 2-99 illustrates single termination inside a Virtex-II device.

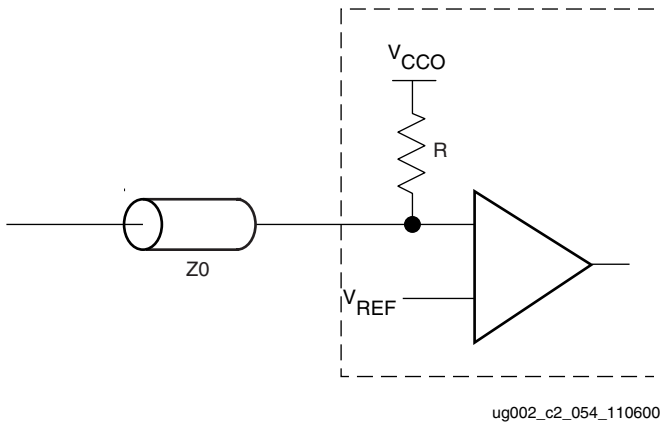


Figure 2-99: **Single Termination Using DCI**

Termination to $V_{CCO}/2$ (Split Termination)

Some I/O standards, such as HSTL Class I, II, SSTL3_I, etc., require an input termination voltage of $V_{CCO}/2$. See Figure 2-100.

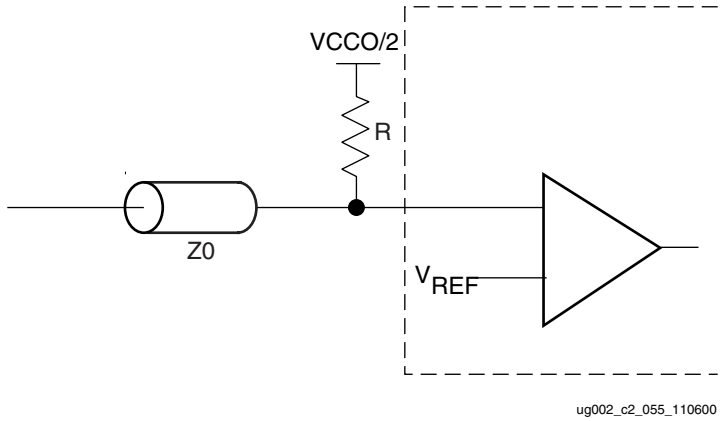


Figure 2-100: **Split Termination Without DCI**

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_{CC} and ground are each twice the reference resistor value. If users are planning to use HSTL or SSTL standards, the reference resistors should be 50-ohms. The DCI I/O standards that support split termination are: HSTL_I_DCI, HSTL_II_DCI, SSTL2_I_DCI, SSTL2_II_DCI, SSTL3_I_DCI, and SSTL3_II_DCI.

Figure 2-101 illustrates split termination inside a Virtex-II device.

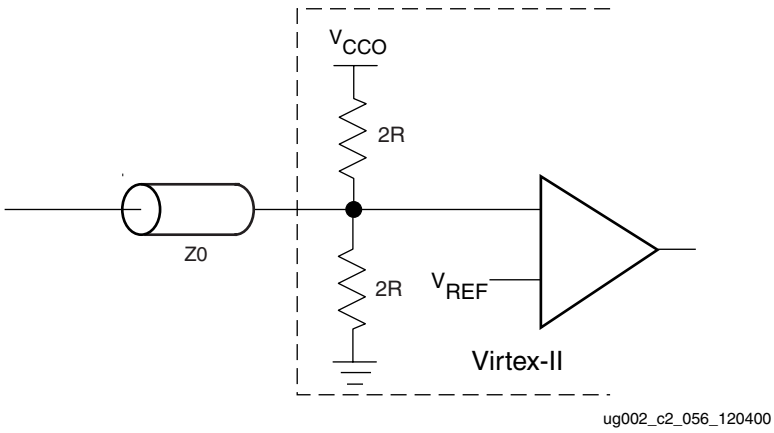


Figure 2-101: **Split Termination Using DCI**

Driver With Single Termination

Some I/O standards, such as HSTL Class IV, require an output termination to V_{CCO}. Figure 2-102 illustrates the output termination to V_{CCO}.

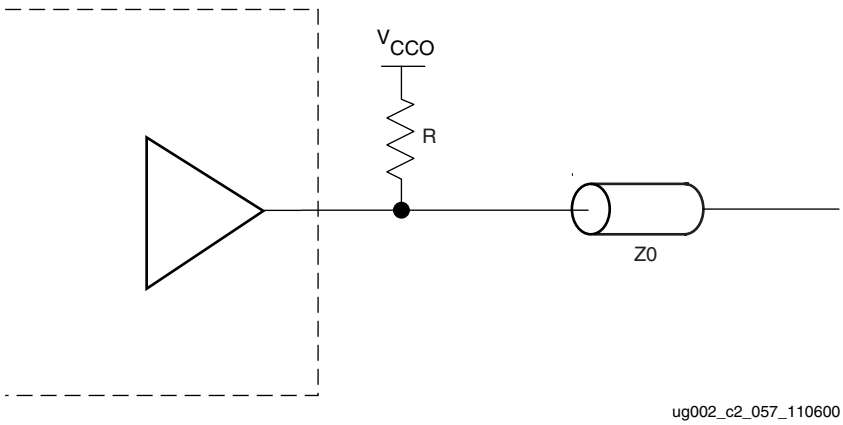


Figure 2-102: **Driver With Single Termination Without DCI**

DCI can provide this termination to V_{CCO} using single termination. In this case, DCI only controls the impedance of the termination, but not the driver. If users are planning to use GTL or HSTL standards, the external reference resistors should be 50-ohms. The DCI I/O standards that support a driver with single termination are: GTL_DCI, GTLP_DCI, and HSTL_IV_DCI.

Figure 2-103 illustrates a driver with single termination inside a Virtex-II device

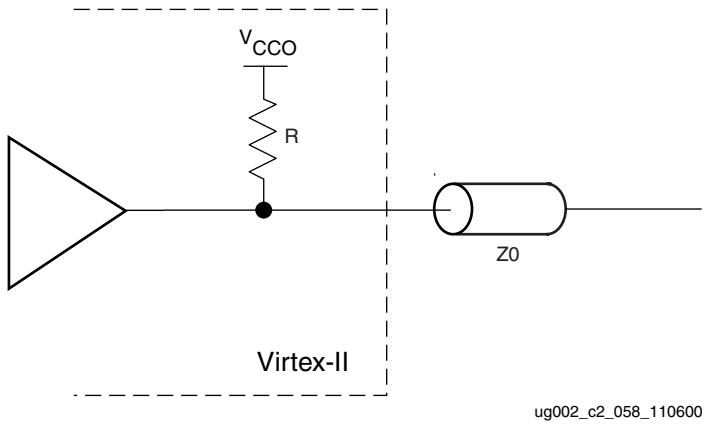


Figure 2-103: **Driver With Single Termination Using DCI**

Driver With Split Termination

Some I/O standards, such as HSTL Class II, require an output termination to $V_{CCO}/2$. See Figure 2-104.

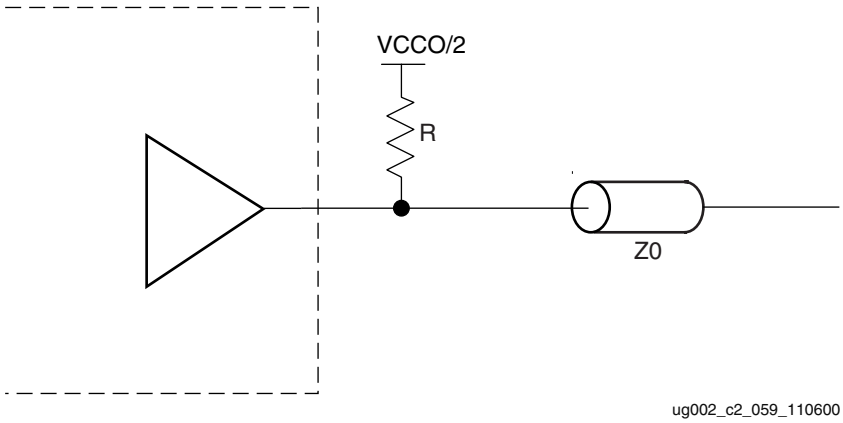


Figure 2-104: **Driver With Split Terminating**

DCI can provide this termination to $V_{CCO}/2$ using split termination. It only controls the impedance of the termination, but not the driver. For HSTL or SSTL standards, the external reference resistors should be 50-ohms. The DCI I/O standards that support a Driver with split termination are: HSTL_II_DCI, SSTL2_II_DCI, and SSTL3_II_DCI.

Figure 2-105 illustrates a driver with split termination inside a Virtex-II device.

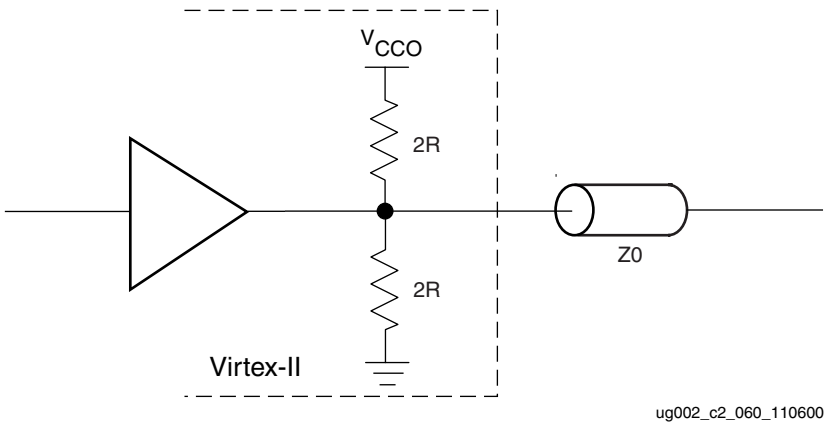


Figure 2-105: **Driver With Split Termination Using DCI**

Software Support

This section lists the valid DCI I/O buffer library components and describes how to use DCI in the Xilinx software.

DCI I/O Buffer Library Components

The DCI input buffer library components, including global clock buffer, are the following:

- IBUFG_GTLP_DCI
- IBUFG_GTL_DCI
- IBUFG_HSTL_I_DCI
- IBUFG_HSTL_II_DCI
- IBUFG_HSTL_III_DCI
- IBUFG_HSTL_IV_DCI
- IBUFG_LVDCI_15
- IBUFG_LVDCI_18
- IBUFG_LVDCI_25
- IBUFG_LVDCI_33
- IBUFG_LVDCI_DV2_15
- IBUFG_LVDCI_DV2_18
- IBUFG_LVDCI_DV2_25
- IBUFG_LVDCI_DV2_33
- IBUFG_SSTL2_I_DCI
- IBUFG_SSTL2_II_DCI
- IBUFG_SSTL3_I_DCI
- IBUFG_SSTL3_II_DCI
- IBUF_GTLP_DCI
- IBUF_GTL_DCI
- IBUF_HSTL_I_DCI
- IBUF_HSTL_II_DCI
- IBUF_HSTL_III_DCI
- IBUF_HSTL_IV_DCI
- IBUF_LVDCI_15
- IBUF_LVDCI_18
- IBUF_LVDCI_25
- IBUF_LVDCI_33
- IBUF_LVDCI_DV2_15
- IBUF_LVDCI_DV2_18
- IBUF_LVDCI_DV2_25
- IBUF_LVDCI_DV2_33
- IBUF_SSTL2_I_DCI
- IBUF_SSTL2_II_DCI
- IBUF_SSTL3_I_DCI
- IBUF_SSTL3_II_DCI

The following are DCI output buffer library components:

- OBUF_GTLP_DCI
- OBUF_GTL_DCI
- OBUF_HSTL_I_DCI
- OBUF_HSTL_II_DCI
- OBUF_HSTL_III_DCI
- OBUF_HSTL_IV_DCI
- OBUF_LVDCI_15
- OBUF_LVDCI_18
- OBUF_LVDCI_25
- OBUF_LVDCI_33
- OBUF_LVDCI_DV2_15
- OBUF_LVDCI_DV2_18
- OBUF_LVDCI_DV2_25
- OBUF_LVDCI_DV2_33
- OBUF_SSTL2_I_DCI
- OBUF_SSTL2_II_DCI
- OBUF_SSTL3_I_DCI
- OBUF_SSTL3_II_DCI

The following are DCI 3 state output buffer library components:

- OBUFT_GTLP_DCI
- OBUFT_GTL_DCI
- OBUFT_HSTL_I_DCI
- OBUFT_HSTL_II_DCI
- OBUFT_HSTL_III_DCI
- OBUFT_HSTL_IV_DCI
- OBUFT_LVDCI_15
- OBUFT_LVDCI_18
- OBUFT_LVDCI_25
- OBUFT_LVDCI_33
- OBUFT_LVDCI_DV2_15
- OBUFT_LVDCI_DV2_18
- OBUFT_LVDCI_DV2_25
- OBUFT_LVDCI_DV2_33
- OBUFT_SSTL2_I_DCI
- OBUFT_SSTL2_II_DCI
- OBUFT_SSTL3_I_DCI
- OBUFT_SSTL3_II_DCI

The following are DCI I/O buffer library components:

- IOBUF_GTLP_DCI
- IOBUF_GTL_DCI
- IOBUF_HSTL_II_DCI
- IOBUF_HSTL_IV_DCI
- IOBUF_SSTL2_II_DCI
- IOBUF_SSTL3_II_DCI
- IOBUF_LVDCI_15
- IOBUF_LVDCI_18
- IOBUF_LVDCI_25
- IOBUF_LVDCI_33
- IOBUF_LVDCI_DV2_15
- IOBUF_LVDCI_DV2_18
- IOBUF_LVDCI_DV2_25
- IOBUF_LVDCI_DV2_33

How to Use DCI in the Software

There are two ways for users to use DCI for Virtex-II devices:

1. Use the IOSTANDARD attribute in the constraint file.
2. Instantiate DCI input or output buffers in the HDL code.

IOSTANDARD Attribute

The IOSTANDARD attribute can be entered through the NCF or UCF file. The syntax is as follows:

```
NET <net name> IOSTANDARD = LVDCI_25;
```

Where <net name> is the name between the IPAD and IBUF or OPAD or OBUF. For HDL designs, this name is the same as the port name.

The following are valid DCI attributes for output drivers:

- LVDCI_15
- LVDCI_18
- LVDCI_25
- LVDCI_33
- LVDCI_DV2_15
- LVDCI_DV2_15
- LVDCI_DV2_25
- LVDCI_DV2_33

The following are valid DCI attributes for terminations:

- GTL_DCI
- GTLP_DCI
- HSTL_I_DCI
- HSTL_II_DCI

- HSTL_III_DCI
- HSTL_IV_DCI
- SSTL2_I_DCI
- SSTL2_II_DCI
- SSTL3_I_DCI
- SSTL3_II_DCI

VHDL Example

Instantiating DCI input and output buffers is the same as instantiating any other I/O buffers. Users must make sure that the correct I/O buffer names are used and follow the standard syntax of instantiation.

For example, to instantiate a HSTL Class I output DCI buffer, the following syntax can be used:

```
HSTL_DCI_buffer: OBUF_HSTL_I_DCI port map (I=>data_out, O=>data_out_DCI);
```

Below is an example VHDL code that instantiates four 2.5 V LVDCI drivers and four HSTL Class I outputs.

```
-- Module: DCI_TEST
--
-- Description: VHDL example for DCI SelectI/O
-- Device: Virtex-II Family
----------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity dci_test is
port (clk, reset, ce, control : in std_logic;
  A, B : in std_logic_vector (3 downto 0);
  Dout : out std_logic_vector (3 downto 0);
  muxout : out std_logic_vector (3 downto 0));
end dci_test;

architecture dci_arch of dci_test is

--DCI output buffer component declaration
component OBUF_LVDCI_25 port (I : in std_logic; O : out std_logic);
end component;
attribute syn_black_box of OBUF_LVDCI_25 : component is true;
attribute black_box_pad_pin of OBUF_LVDCI_25 : component is "O";

--HSTL Class I DCI output buffer component declaration
component OBUF_HSTL_I_DCI port (I : in std_logic; O: out std_logic);
end component;
attribute syn_black_box of OBUF_HSTL_I_DCI : component is true;
attribute black_box_pad_pin of OBUF_HSTL_I_DCI : component is "O";

signal muxout_int : std_logic_vector (3 downto 0);
signal dout_int : std_logic_vector (3 downto 0);

begin

process (clk, reset)
begin
  if (reset = '1') then
        dout_int<="0000";
    elsif (clk'event and clk='1') then
```

```
      dout_int<=dout_int+1;
end if;
end process;

process (controls, A, B, DOUT_INT)

begin
  if (control='1') then
    muxout_int<=A and B;
else
    muxout_int<=Dout_int;
  end if;
end process;

U0 : OBUF_LVDCI_25 port map(
  I=>dout_int(0),
  O=>dout(0));

U1 : OBUF_LVDCI_25 port map(
       I=>dout_int(1),
       O=>dout(1));
U2 : OBUF_LVDCI_25 port map(
       I=>dout_int(2),
       O=>dout(2));
U3 : OBUF_LVDCI_25 port map(
       I=>dout_int(3),
       O=>dout(3));

K0 : OBUF_HSTL_I_DCI port map(
  I=>muxout_int(0),
  O=>muxout(0));

K1 : OBUF_HSTL_I_DCI port map(
       I=>muxout_int(1),
       O=>muxout(1));
K2 : OBUF_HSTL_I_DCI port map(
       I=>muxout_int(2),
       O=>muxout(2));
K3 : OBUF_HSTL_I_DCI port map(
       I=>muxout_int(3),
       O=>muxout(3));

end dci_arch;
```

DCI in Virtex-II Hardware

DCI only works with certain single-ended I/O standards and does not work with any differential I/O standard. DCI supports the following Virtex-II standards:

LVDCI, LVDCI_DV2, GTL_DCI, GTLP_DCI, HSTL_I_DCI, HSTL_II_DCI, HSTL_III_DCI, HSTL_IV_DCI, SSTL2_I_DCI, SSTL2_II_DCI, SSTL3_I_DCI, and SSTL3_II_DCI.

To correctly use DCI in a Virtex-II 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. External reference resistors must be connected to multi-purpose pins (VRN and VRP) in the bank cannot be used as regular I/Os. Refer to the Virtex-II 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.

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 they should be 50 ohms.
5. The values of the reference resistors must be within the supported range. Availability of this range is planned for the next release of the **Virtex-II Data Sheet**. (~30 to 100 Ω)
6. Follow the DCI I/O banking rules.

The DCI I/O banking rules are the following:

1. V_{REF} must be compatible for all of the inputs in the same bank.
2. V_{CCO} must be compatible for all of the inputs and outputs in the same bank.
3. No more than one DCI I/O standard using Single Termination type is allowed per bank.
4. No more than one DCI I/O standard using Split Termination type is allowed per bank.
5. Single Termination and Split Termination, Controlled Impedance Driver, and Controlled Impedance Driver with Half Impedance can co-exist in the same bank.

The behavior of 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 any special care actions that must be taken for each DCI I/O standard.

LVDCI_15, LVDCI_18, LVDCI_25, LVDCI_33

Using these buffers configures the outputs as controlled impedance drivers. The number extension at the end indicates the V_{CCO} voltage that should be used. For example, 15 means V_{CCO}=1.5 V, etc. There is no slew rate control or drive strength settings for LVDCI drivers.

LVDCI_DV2_15, LVDCI_DV2_18, LVDCI_DV2_25, LVDCI_DV_33

Using these buffers configures the outputs as controlled drivers with half impedance. The number extension at the end indicates the V_{CCO} voltage that should be used. For example, 15 means V_{CCO}=1.5 V, etc. There is no slew rate control or drive strength settings for LVDCI_DV2 drivers.

GTL_DCI

GTLP does not require a V_{CCO} voltage. However, for GTL_DCI, V_{CCO} must be connected to 1.2 V. GTL_DCI provides single termination to V_{CCO} for inputs or outputs.

GTLP_DCI

GTL+ does not require a V_{CCO} voltage. However, for GTLP_DCI, V_{CCO} must be connected to 1.5 V. GTLP_DCI provides single termination to V_{CCO} for inputs or outputs.

HSTL_ I_DCI, HSTL_ III_DCI

HSTL_I_DCI provides split termination to $V_{CCO}/2$ for inputs. HSTL_III_DCI provides single termination to V_{CCO} for inputs.

HSTL_ II_DCI, HSTL_ IV_DCI

HSTL_II_DCI provides split termination to $V_{CCO}/2$ for inputs or outputs. HSTL_IV_ DCI provides single termination to V_{CCO} for inputs or outputs.

SSTL2_ I_DCI, SSTL3_I_DCI

SSTL2_I_DCI and SSTL3_I_DCI provide split termination to $V_{CCO}/2$ for inputs. Then I/O standards are SSTL compatible.

SSTL2_II_DCI, SSTL3_II_DCI

SSTL2_II_DCI and SSTL3_II_DCI provide split termination to $V_{CCO}/2$ for inputs. Then I/O standards are SSTL compatible.

Figure 2-106 provides examples illustrating the use of the HSTL_I_DCI, HSTL_II_DCI, HSTL_III_DCI, and HSTL_IV_DCI I/O standards.

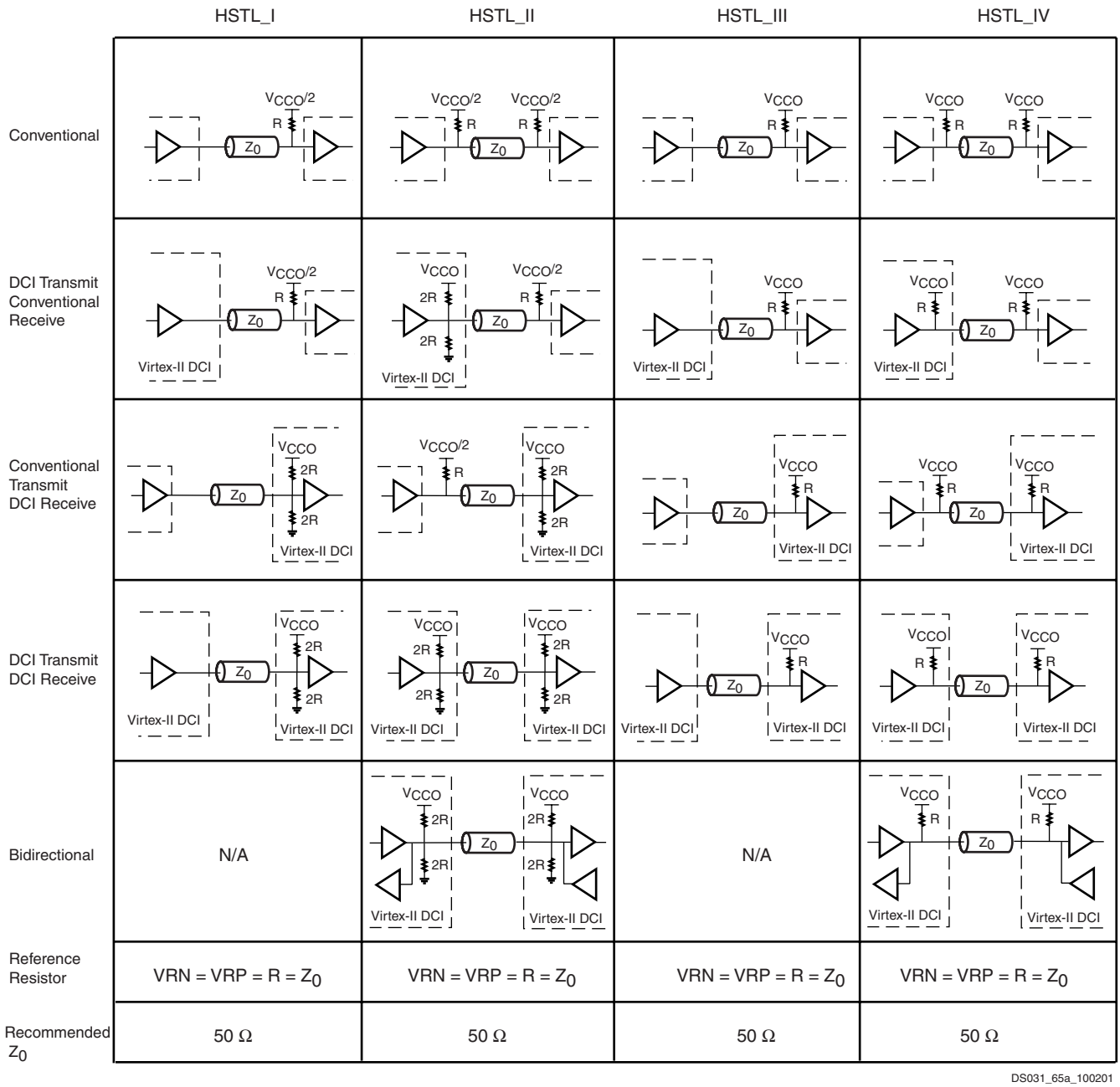


Figure 2-106: **HSTL DCI Usage Examples**

Figure 2-107 provides examples illustrating the use of the SSTL2_I_DCI, SSTL2_II_DCI, SSTL3_I_DCI, and SSTL3_II_DCI I/O standards.

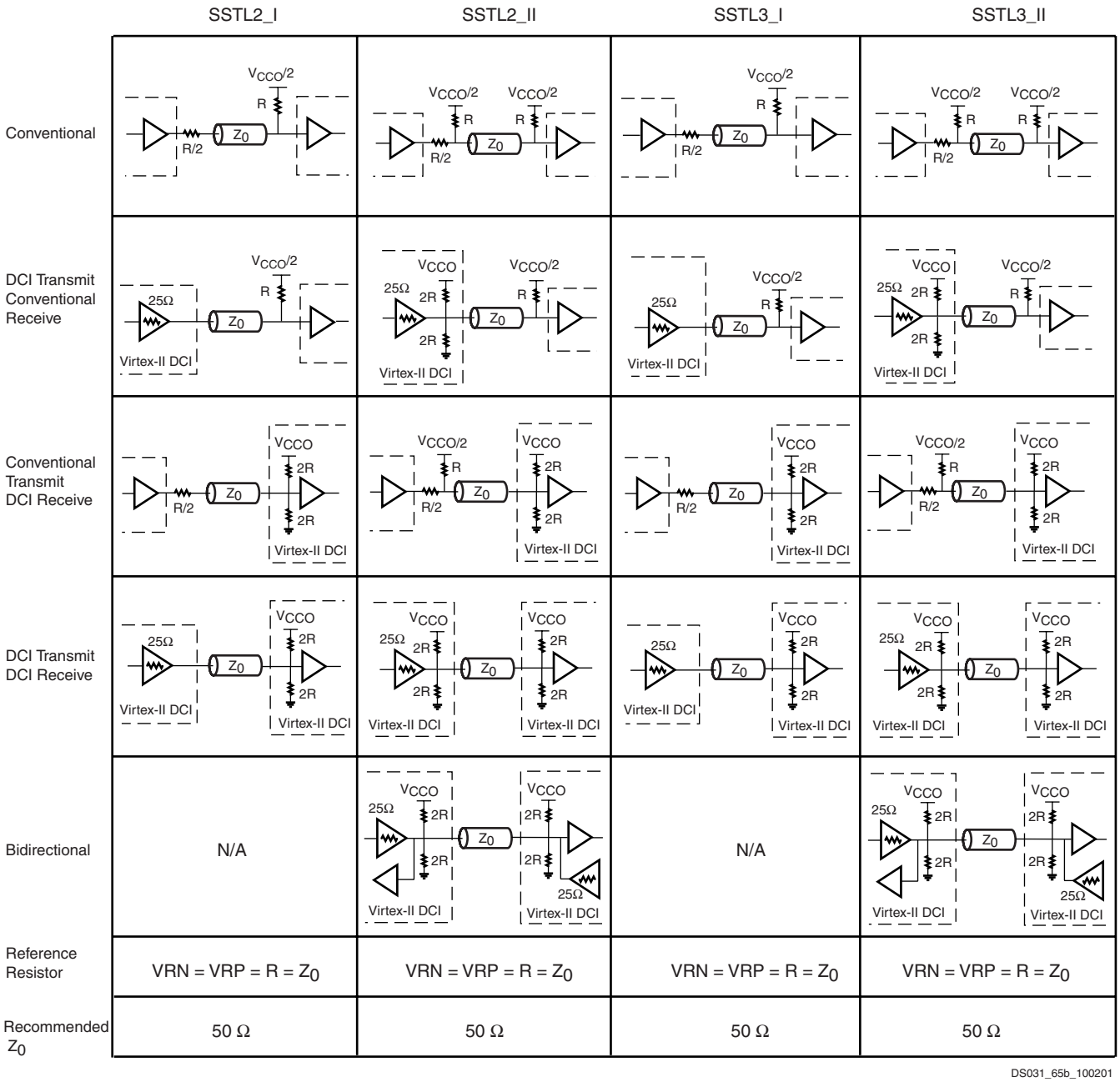


Figure 2-107: **SSTL DCI Usage Examples**

Using Double-Data-Rate (DDR) I/O

Introduction

Virtex-II devices have dedicated registers in a single IOB to implement input, output, and output with 3-state control Double-Data-Rate (DDR) registers. Input and output DDR is accomplished with the use of two registers in the IOB. A single clock triggers one register on a Low to High transition and a second register on a High to Low transition. Output DDR with 3-state requires the use of four registers in the IOB clocked in a similar fashion. Since the introduction of DLLs, Xilinx devices can generate low-skew clock signals that are 180 degrees out of phase, with a 50/50 duty cycle. These clocks reach the DDR registers in the IOB via dedicated routing resources.

Data Flow

Input DDR

Input DDR is accomplished via a single input signal driving two registers in the IOB. Both registers are clocked on the rising edge of their respective clocks. With proper clock forwarding, alternating bits from the input signal are clocked in on the rising edge of the two clocks, which are 180 degrees out of phase. Figure 2-108 depicts the input DDR registers and the signals involved.

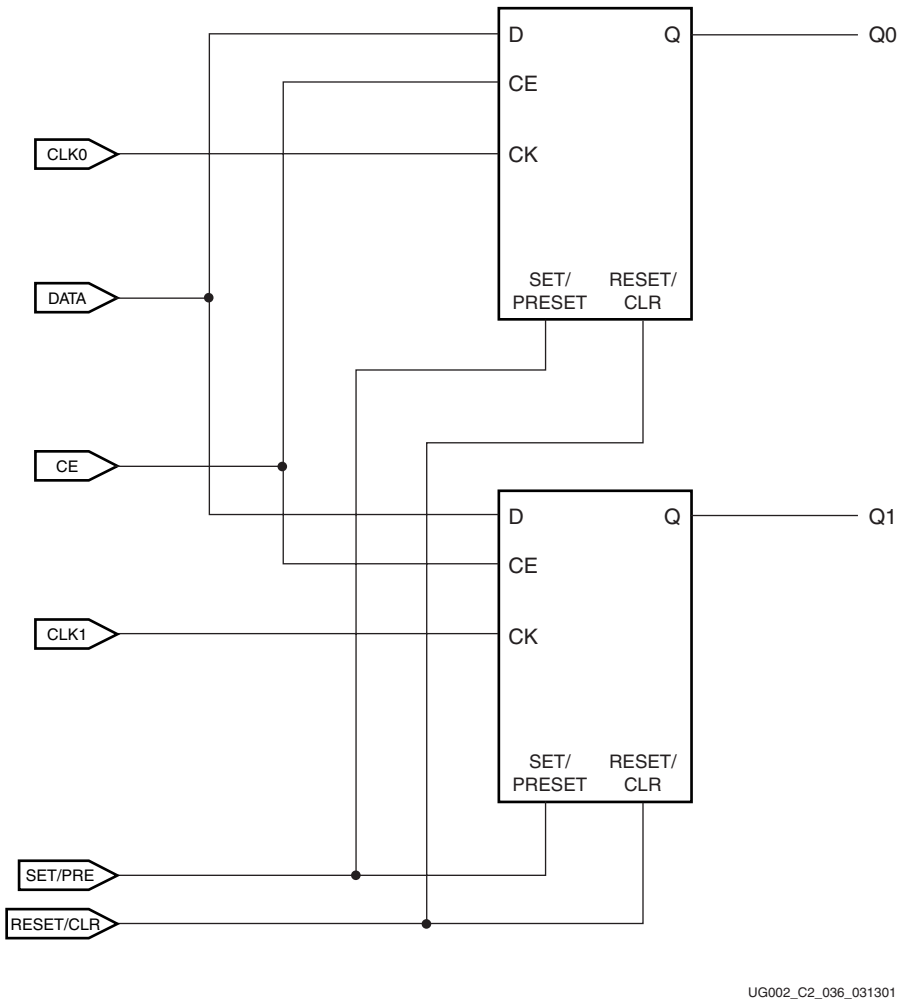


Figure 2-108: **Input DDR**

CLK0 and CLK1 are 180 degrees out of phase. Both registers share the SET/PRE and RESET/CLR lines. As shown in Figure 2-109, alternating bits on the DATA line are clocked in via Q0 and Q1 while CE is High. The clocks are shifted out of phase by the DCM (CLK0 and CLK180 outputs) or by the inverter available on the CLK1 clock input..

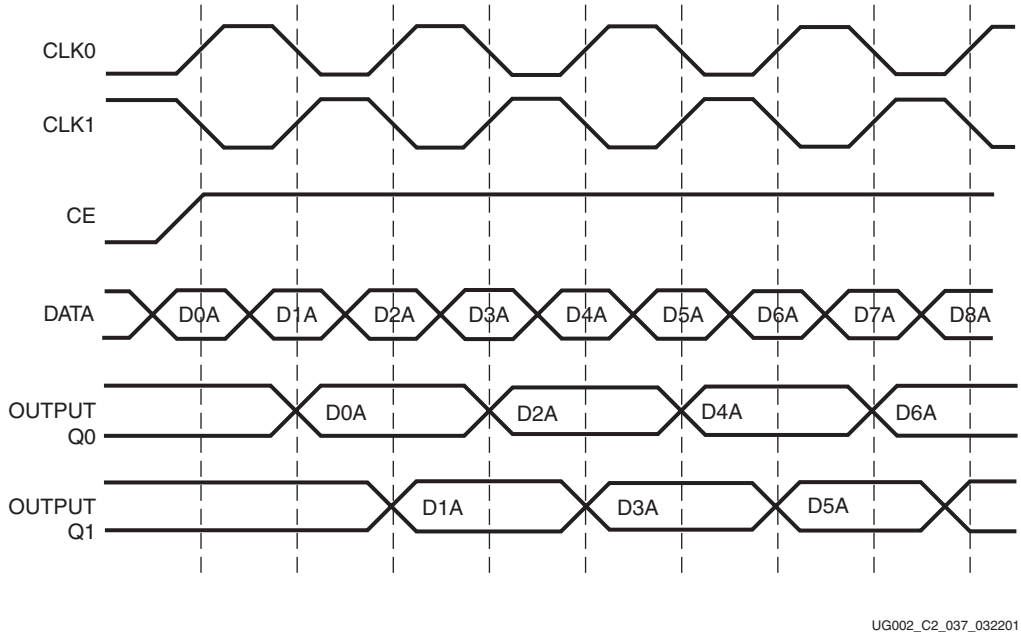


Figure 2-109: **Input DDR Timing Diagram**

Output DDR

Output DDR registers are used to clock output from the chip at twice the throughput of a single rising-edge clocking scheme. Clocking for output DDR is the same as input DDR. The clocks driving both registers are 180 degrees out of phase. The DDR MUX selects the register outputs. The output consists of alternating bits from DATA_1 and DATA_2. Figure 2-110 depicts the output DDR registers and the signals involved.

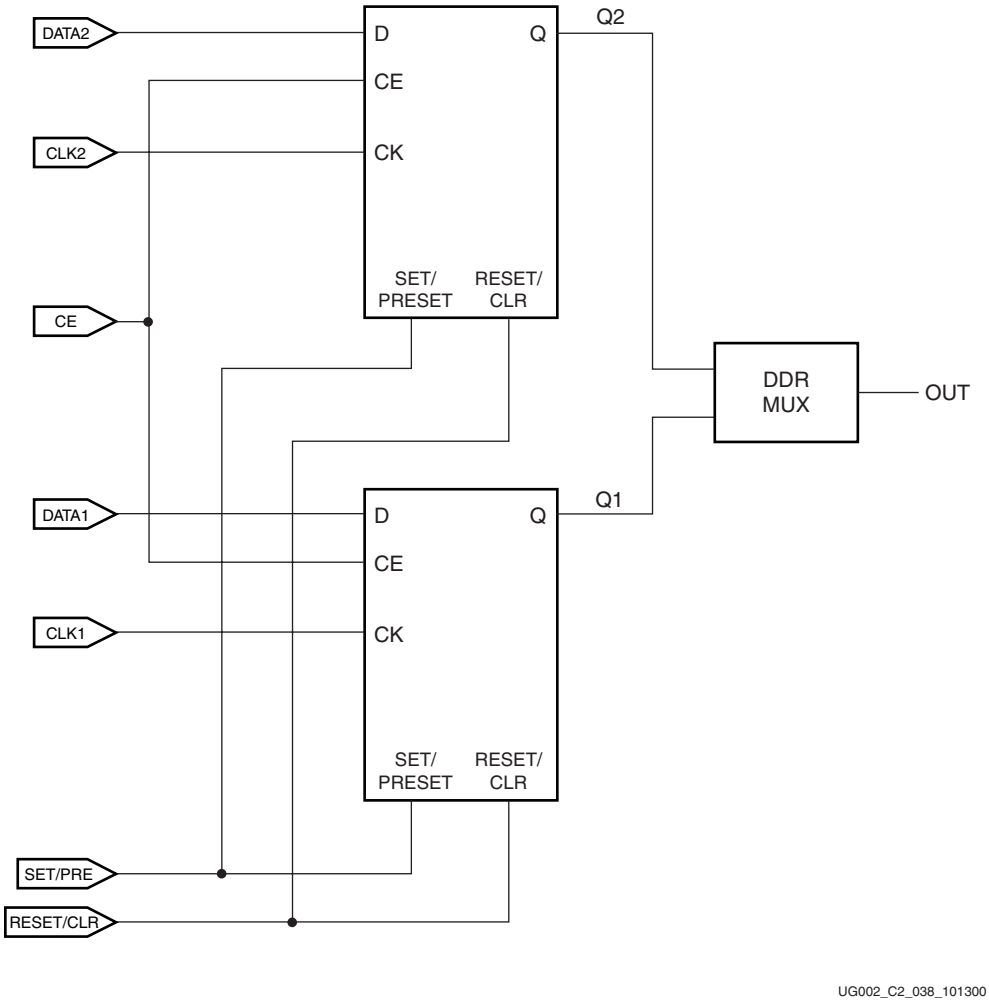


Figure 2-110: **Output DDR**

Both registers share the SET/PRE and RESET/CLR line. Both registers share the CE line which must be High for outputs to be seen on Q1 and Q2. Figure 2-111 shows the data flow for the output DDR registers.

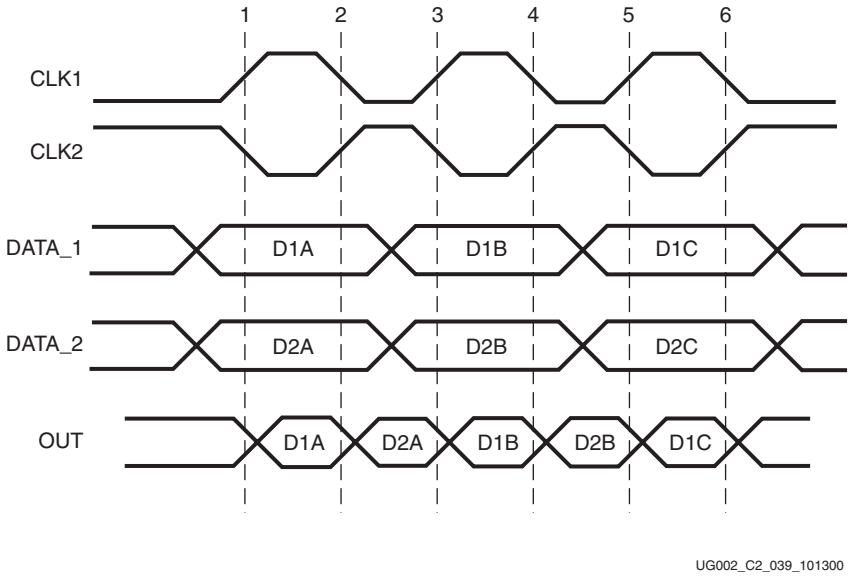


Figure 2-111: **Output DDR Timing Diagram**

Output DDR With 3-State Control

The 3-state control allows the output to have one of two values, either the output from the DDR MUX or high impedance.

The Enable signal is driven by a second DDR MUX (Figure 2-112). This application requires the instantiation of two output DDR primitives.

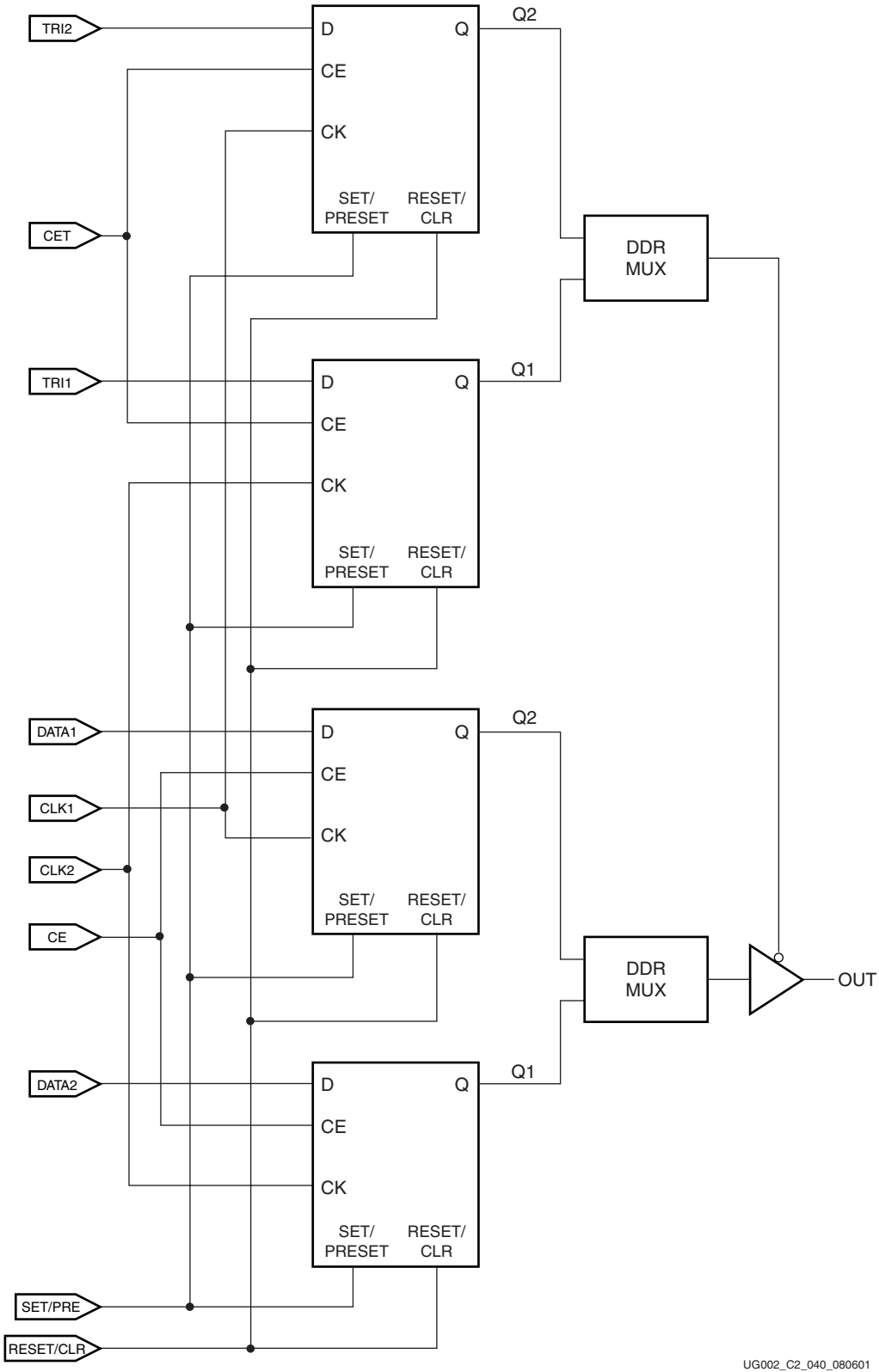


Figure 2-112: **Output DDR With 3-State Control**

All four registers share the SET/PRESET and RESET/CLEAR lines. Two registers are required to accomplish the DDR task and two registers are required for the 3-state control. There are two Clock Enable signals, one for output DDRs performing the DDR function and another for the output DDRs performing the 3-state control function. Two 180 degree out of phase clocks are used. CLK1 clocks one of the DDR registers and a 3-state register. CLK2 clocks the other DDR register and the other 3-state register.

The DDR registers and 3-state registers are associated by the clock that is driving them. Therefore, the DDR register that is clocked by CLK1 is associated to the 3-state register being clocked by CLK1. The remaining two registers are associated by CLK2. If both 3-state registers are driving a logic High, the output sees a high impedance. If both 3-state registers are driving a logic Low, the output sees the values from the DDR MUX see Figure 2-113).

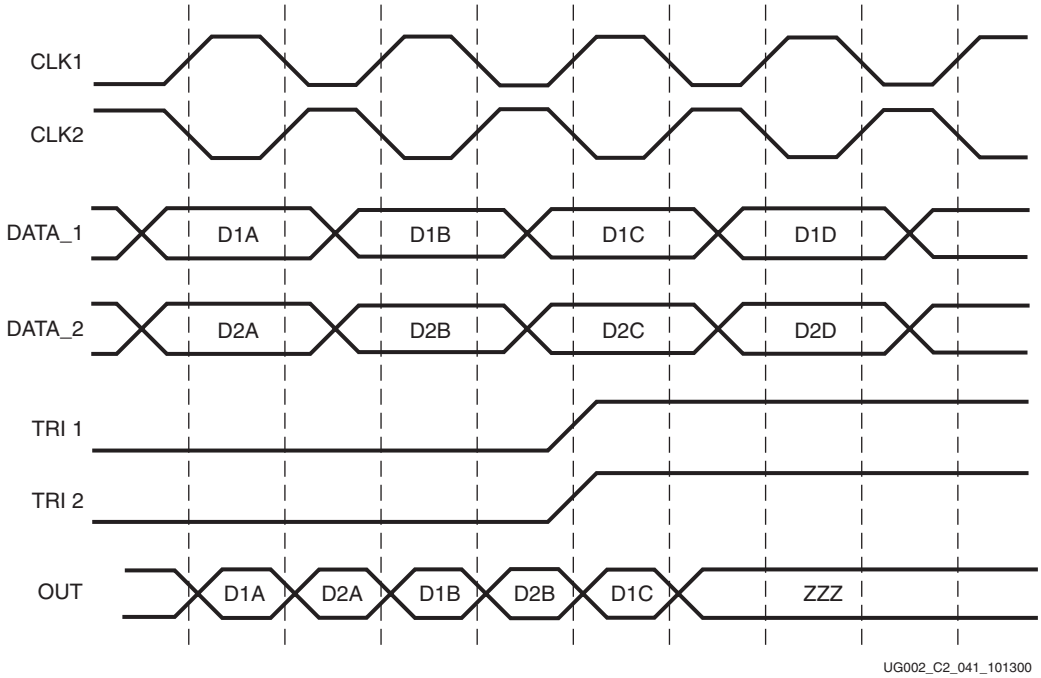


Figure 2-113: **Timing Diagram for Output DDR With 3-State Control**

When the 3-state registers are not driving the same logic value, the 3-state register being clocked by CLK1 is called TREG1. The other 3-state register TREG2 is clocked by CLK2. Similarly, the DDR register being clocked by CLK1 is called DREG1, and the other DDR register DREG2 is clocked by CLK2. If TREG1 is driving a logic High and TREG2 is driving a logic Low, the output sees a high impedance when CLK1 is High and the value out of DREG2 when CLK2 is High. If TREG2 is driving a logic High and TREG1 is driving a logic Low, the output sees a high impedance when CLK2 is High and the value out of DREG1 when CLK1 is High.

Characteristics

- All registers in an IOB share the same SET/PRE and RESET/CLR lines.
- The 3-State and Output DDR registers have common clocks (OTCLK1 & OTCLK2).
- All signals can be inverted (with no added delay) inside the IOB.
- DDR MUXing is handled automatically within the IOB. There is no manual control of the MUX-select. This control is generated from the clock.
- When several clocks are used, and when using DDR registers, the floorplan of a design should take into account that the input clock to an IOB is shared with a pair of IOBs.

Library Primitives

Input DDR registers are inferred, and dedicated output DDR registers have been provided as primitives for Virtex-II designs. Input DDR registers consist of two inferred registers that clock in a single data line on each edge. Generating 3-state output with DDR registers is as simple as instantiating a primitive.

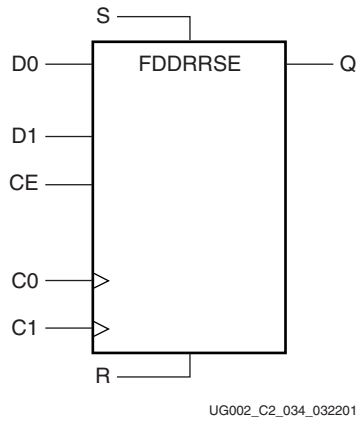


Figure 2-114: **FDDRRSE Symbol: DDR Flip-Flop With Clock Enable and Synchronous Reset and Set**

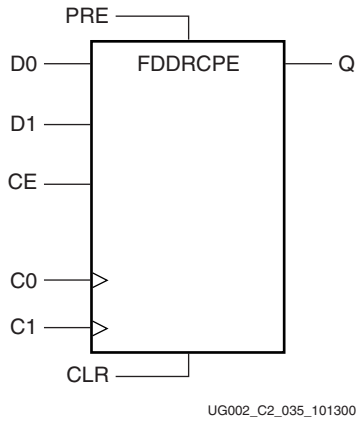


Figure 2-115: **FDDRCPE Symbol: DDR Flip-Flop With Clock Enable and Asynchronous PRESET and CLR**

VHDL and Verilog Instantiation

Examples are available in "VHDL and Verilog Templates" on page 311.

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.

Constraints file syntax is provided where input registers need to be used. These settings force the input DDR registers into the IOB. The output registers should be instantiated and do not require any constraints file syntax to be pushed into the IOB.

Port Signals

FDDRRSE

Data inputs - D0 and D1

D0 and D1 are the data inputs into the DDR flip-flop. Data on the D0 input is loaded into the flip-flop when R and S are Low and CE is High during a Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when R and S are Low and CE is High during a Low-to-High C1 clock transition.

Clock Enable - CE

The enable pin affects the loading of data into the DDR flip-flop. When Low, new data is not loaded into the flip-flop. CE must be High to load new data into the flip-flop.

Clocks - C0 and C1

These two clocks are phase shifted 180 degrees (via the DLL) and allow selection of two separate data inputs (D0 and D1).

Synchronous Set - S and Synchronous Reset - R

The Reset (R) input, when High, overrides all other inputs and resets the output Low during any Low-to-High clock transition (C0 or C1). Reset has precedence over Set. When the Set (S) input is High and R is Low, the flip-flop is set, output High, during a Low-to-High clock transition (C0 or C1).

Data Output - Q

When power is applied, the flip-flop is asynchronously cleared and the output is Low.

During normal operation, The value of Q is either D0 or D1. The Data Inputs description above states how the value of Q is chosen.

FDDRCPE

Data inputs - D0 and D1

D0 and D1 are the data inputs into the DDR flip-flop. Data on the D0 input is loaded into the flip-flop when PRE and CLR are Low and CE is High during a Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when PRE and CLR are Low and CE is High during a Low-to-High C1 clock transition.

Clock Enable - CE

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 flip-flop. CE must be High to load new data into the flip-flop.

Clocks - C0 and C1

These two clocks are phase shifted 180 degrees (via the DLL) and allow selection of two separate data inputs (D0 and D1).

Asynchronous Preset - PRE and Asynchronous Clear - CLR

The Preset (PRE) input, when High, sets the Q output High. When the Clear (CLR) input is High, the output is reset to Low.

Data Output - Q

When power is applied, the flip-flop is asynchronously cleared and the output is Low. During normal operation, The value of Q is either D0 or D1. The Data Inputs description above states how the value of Q is chosen.

Initialization in VHDL or Verilog

Output DDR primitives can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the output DDR instantiation and are copied in the EDIF output file to be compiled by Xilinx tools. The VHDL code simulation uses a `generic` parameter to pass the attributes. The Verilog code simulation uses the `defparam` parameter to pass the attributes.

The DDR code examples (in VHDL ad Verilog) illustrate the following techniques.

Location Constraints

DDR instances can have LOC properties attached to them to constrain pin placement.

The LOC constraint uses the following form.

```
NET <net_name> LOC=A8;
```

Where "A8" is a valid I/O pin location.

Applications

DDR SDRAM

The DDR SDRAM is an enhancement to the Synchronous DRAM by effectively doubling the data throughput of the memory device. Commands are registered at every positive clock edge. Input data is registered on both edges of the data strobe, and output data is referenced to both edges of the data strobe, as well as both edges of the clock.

Clock Forwarding

DDR can be used to forward a copy of the clock on the output. This can be useful for propagating a clock along with double-data-rate data that has an identical delay. It is also useful for multiple clock generation, where there is a unique clock driver for every clock load.

VHDL and Verilog Templates

VHDL and Verilog templates are available for output, output with 3-state enable, and input DDR registers.

Input DDR

To implement an Input DDR application, paste the following template in your code.

DDR_input.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DDR_Input is

  Port (
    clk : in std_logic;
    d : in std_logic;
    rst : in std_logic;
    q1 : out std_logic;
    q2 : out std_logic
       );

end DDR_Input;

--Describe input DDR registers (behaviorally) to be inferred

architecture behavioral of DDR_Input is
```

```
begin

q1reg : process (clk, d, rst)

begin
   if rst='1' then --asynchronous reset, active high
     q1 <= '0';
   elsif clk'event and clk='1' then  --Clock event - posedge
     q1 <= d;

   end if;
end process;

q2reg : process (clk, d, rst)

begin
   if rst='1' then --asynchronous reset, active high
      q2 <= '0';
   elsif clk'event and clk='0' then  --Clock event - negedge
      q2 <= d;
   end if;
end process;

end behavioral;


-- NOTE: You must include the following constraints in the .ucf
-- file when running back-end tools,
-- in order to ensure that IOB DDR registers are used:
--
-- INST "q2_reg" IOB=TRUE;
-- INST "q1_reg" IOB=TRUE;
--
-- Depending on the synthesis tools you use, it may be required to
-- check the edif file for modifications to
-- original net names...in this case, Synopsis changed the
-- names: q1 and q2 to q1_reg and q2_reg
```

DDR_input.v

```
module DDR_Input (data_in , q1, q2, clk, rst);

input data_in, clk, rst;
output q1, q2;
reg q1, q2;


//Describe input DDR registers (behaviorally) to be inferred

always @ (posedge clk or posedge rst) //rising-edge DDR reg. and
asynchronous reset

  begin
  if (rst)
     q1 = 1'b0;
  else
     q1 = data_in;
   end
```

```
always @ (negedge clk or posedge rst) //falling-edge DDR reg. and
asynchronous reset

begin
  if (rst)
     q2 = 1'b0;
  else
     q2 = data_in;
   end

assign data_out = q1 & q2;



endmodule

/* NOTE: You must include the following constraints in the .ucf file
when running back-end tools, \
  in order to ensure that IOB DDR registers are used:

INST "q2_reg" IOB=TRUE;
INST "q1_reg" IOB=TRUE;

Depending on the synthesis tools you use, it may be required to check
the edif file for modifications to
original net names...in this case, Synopsis changed the names: q1 and q2
to q1_reg and q2_reg

*/
```


Output DDR

To implement an Output DDR application, paste the following template in your code.

DDR_out.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- pragma translate_off
LIBRARY UNISIM;
use UNISIM.VCOMPONENTS.ALL;
--pragma translate_on

entity DDR_Output is
  Port(
   clk : in std_logic; --clk and clk180 can be outputs from the DCM or
clk180 can be the
   clk180 : in std_logic;  --logical inverse of clk (the inverter is
located in the IOB and will be inferred.
   d0 : in std_logic;  --data in to fddr
   d1 : in std_logic;  --data in to fddr
   ce : in std_logic;  --clock enable
   rst : in std_logic;  --reset
   set : in std_logic;  --set
   q : out std_logic  --DDR output
   );

end DDR_Output;

architecture behavioral of DDR_Output is

component FDDRRSE
  port(
```

```
    Q  : out std_logic;
    D0 : in std_logic;
    D1 : in std_logic;
    C0 : in std_logic;
    C1 : in std_logic;
    CE : in std_logic;
    R  : in std_logic;
    S : in std_logic
    );
end component;

begin

U0: FDDRRSE
  port map (
    Q => q,
    D0 => d0,
    D1 => d1,
    C0 => clk,
    C1 => clk180,
    CE => ce,
    R => rst,
    S => set
    );


end behavioral;
```

DDR_out.v

```
module DDR_Output (d0 , d1, q, clk, clk180, rst, set, ce);

input d0, d1, clk, clk180, rst, set, ce;
output q;


//Synchronous Output DDR primitive instantiation

FDDRRSE U1  ( .D0(d0),
              .D1(d1),
              .C0(clk),
              .C1(clk180),
              .CE(ce),
              .R(rst),
              .S(set),
              .Q(q)
            );
endmodule
```

Output DDR With 3-State Enable

To implement an Output DDR with 3-state Enable, paste the following template in your code:

DDR_3state.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- pragma translate_off
LIBRARY UNISIM;
use UNISIM.VCOMPONENTS.ALL;
--pragma translate_on
```

```
entity DDR_3state is
  Port(
    clk : in std_logic; --clk and clk180 can be outputs from the DCM or
clk180 can be the
    clk180 : in std_logic; --logical inverse of clk (the inverter is
located in the IOB and will be inferred.
    d0 : in std_logic; --data in to fddr
    d1 : in std_logic; --data in to fddr
    ce : in std_logic; --clock enable
    set : in std_logic; --set
    rst : in std_logic; --reset
    en0 : in std_logic; --enable signal
    en1 : in std_logic; --enable signal
    data_out : out std_logic --data seen at pad
  );

end DDR_3state;

architecture behavioral of DDR_3state is

signal ddr_out, tri : std_logic;

component FDDRRSE
   port  (
    Q   : out std_logic;
    D0  : in std_logic;
    D1  : in std_logic;
    C0  : in std_logic;
    C1  : in std_logic;
    CE  : in std_logic;
    R   : in std_logic;
    S   : in std_logic
         );
end component;


begin

--Instantiate Ouput DDR registers
U0: FDDRRSE port map(Q  => tri,
       D0 => en0,
       D1 => en1,
       C0 => clk,
       C1 => clk180,
       CE => ce,
       R  => rst,
       S  => set
      );

--Instantiate three-state DDR registers
U1: FDDRRSE port map(   Q  => ddr_out,
       D0 => d0,
       D1 => d1,
       C0 => clk,
       C1 => clk180,
       CE => ce,
       R  => rst,
       S  => set
      );

--inferr the 3-State buffer
process(tri, ddr_out)
```

```
begin
  if tri = '1' then
     data_out <= 'Z';
  elsif tri = '0' then
     data_out <= ddr_out;
  end if;
end process;

end behavioral;
```

DDR_3state.v

```
module DDR_3state (d0 , d1, data_out, en_0, en_1, clk, clk180, rst, set,
ce);

input d0, d1, clk, clk180, rst, set, ce, en_0, en_1;

output data_out;
reg data_out;

wire q, q_tri;

//Synchronous Output DDR primitive instantiation

FDDRRSE U1  ( .D0(d0),
              .D1(d1),
              .C0(clk),
              .C1(clk180),
              .CE(ce),
              .R(rst),
              .S(set),
              .Q(q)
            );

//Synchronous 3-State DDR primitive instantiation

FDDRRSE U2  ( .D0(en_0),
              .D1(en_1),
              .C0(clk),
              .C1(clk180),
              .CE(ce),
              .R(rst),
              .S(set),
              .Q(q_tri)
            );

//3-State buffer description

always @ (q_tri or q)
  begin
  if (q_tri)
    data_out = 1'bz;
  else
    data_out = q;
  end

endmodule
```

Using LVDS I/O

Introduction

Low Voltage Differential Signaling (LVDS) is a very popular and powerful high-speed interface in many system applications. Virtex-II I/Os are designed to comply with the IEEE electrical specifications for LVDS to make system and board design easier. With the addition of an LVDS current-mode driver in the IOBs, which eliminates the need for external source termination in point-to-point applications, and with the choice of two different voltage modes and an extended mode, Virtex-II devices provide the most flexible solution for doing an LVDS design in an FPGA.

Table 2-59 lists all LVDS primitives that are available for Virtex-II devices.

Table 2-59: **Available Virtex-II LVDS Primitives**

Input	Output	3-State	Clock	Bi-Directional
IBUF_LVDS	**OBUF_LVDS**	**OBUFT_LVDS**	**IBUFG_LVDS**	**IOBUF_LVDS**
IBUFDS_LVDS_25	OBUFDS_LVDS_25	OBUFTDS_LVDS_25	IBUFGDS_LVDS_25	
IBUFDS_LVDS_33	OBUFDS_LVDS_33	OBUFTDS_LVDS_33	IBUFGDS_LVDS_33	
IBUFDS_LVDSEXT_25	OBUFDS_LVDSEXT_25	OBUFTDS_LVDSEXT_25	IBUFGDS_LVDSEXT_25	
IBUFDS_LVDSEXT_33	OBUFDS_LVDSEXT_33	OBUFTDS_LVDSEXT_33	IBUFGDS_LVDSEXT_33	

The primitives in **bold** type are pre-existing LVDS primitives used in Virtex-E and earlier designs. They are not current-mode drivers and are still required for BLVDS (Bus LVDS) applications.

*DS_LVDS_25 = 2.5V V_{CCO} LVDS Buffer

*DS_LVDS_33 = 3.3V V_{CCO} LVDS Buffer

There is no difference in the AC characteristics of either voltage-mode LVDS I/O. These choices now provide more flexibility for mixed-I/O banking rules; that is, an LVTTL I/O can coexist with the 3.3V LVDS buffer in the same bank.

DS_LVDSEXT = Extended mode LVDS buffer

This buffer provides a higher drive capability and voltage swing (350 - 750 mV), which makes it ideal for long-distance or cable LVDS links.

The output AC characteristics of this LVDS driver are not within the EIA/TIA specifications. This LVDS driver is intended for situations that require higher drive capabilities in order to produce an LVDS signal that is within EIA/TIA specification at the receiver.

Creating an LVDS Input/Clock Buffer

Figure 2-116 illustrates the LVDS input and clock buffer primitives shown in Table 2-60. The pin names used are the same as those used in the HDL library primitives.

Table 2-60: **LVDS Input and Clock Buffer Primitives**

LVDS Inputs	LVDS Clocks
IBUFDS_LVDS_25	IBUFGDS_LVDS_25
IBUFDS_LVDS_33	IBUFGDS_LVDS_33
IBUFDS_LVDSEXT_25	IBUFGDS_LVDSEXT_25
IBUFDS_LVDSEXT_33	IBUFGDS_LVDSEXT_33

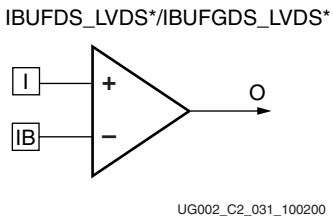


Figure 2-116: **LVDS Input and Clock Primitives**

To create an LVDS input, instantiate the desired mode (2.5 V, 3.3 V, or Extended) LVDS input buffer. Notice that the P and N channels are included in the primitive (I = P, IB = N). Software automatically uses the appropriate pin from an adjacent IOB for the N channel. The same applies to LVDS clocks: Use IBUFGDS_LVDS*

LVDS Input HDL Examples

VHDL Instantiation

```
U1: IBUFDS_LVDS_25
  port map  (
              I => data_in_P,
              IB => data_in_N
              O => data_in
            );
```

Verilog Instantiation

```
IBUFDS_LVDS_25   U1    ( .I(data_in_P),
                         .IB(data_in_N),
                         .O(data_in)
                       );
```

Port Signals

I = P-channel data input to the LVDS input buffer

IB = N-channel data input to the LVDS input buffer

O = Non-differential input data from LVDS input buffer

Location Constraints

```
NET "data_in_P" LOC= "NS";
```

LVDS Receiver Termination

All LVDS receivers require standard termination. Figure 2-117 is an example of a typical termination for an LVDS receiver on a board with 50Ω transmission lines.

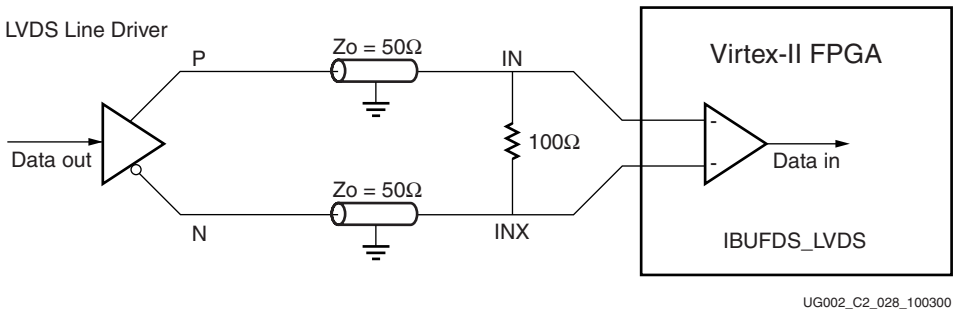


Figure 2-117: **LVDS Receiver Termination**

Creating an LVDS Output Buffer

Figure 2-118 illustrates the LVDS output buffer primitives:

- OBUFDS_LVDS_25
- OBUFDS_LVDS_33
- OBUFDS_LVDSEXT_25
- OBUFDS_LVDSEXT_33

The pin names used are the same as those used in the HDL library primitives.

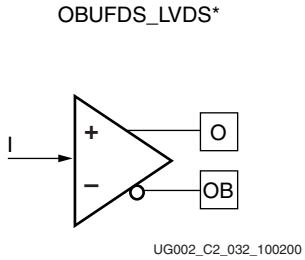


Figure 2-118: **LVDS Output Buffer Primitives**

To create an LVDS output, instantiate the desired mode (2.5, 3.3V, or Extended) LVDS output buffer. Notice that the P and N channels are included in the primitive (O = P, OB = N). Software automatically uses the appropriate pin from an adjacent IOB for the N channel.

LVDS Output HDL Examples

VHDL Instantiation

```
U1: OBUFDS_LVDS_25
  port map  (
              I => data_out,
              O => data_out_P,
              OB => data_out_N
            );
```

Verilog Instantiation

```
OBUFDS_LVDS_25 U1  ( .I(data_out),
                     .O(data_out_P),
                     .OB(data_out_N)
                   );
```

Port Signals

I = data input to the LVDS input buffer

O = P-channel data output

OB = N-channel data output

Location Constraints

```
NET "data_out_P" LOC= "NS";
```

LVDS Transmitter Termination

The Virtex-II LVDS transmitter does not require any termination. Table 2-59 lists primitives that correspond to the Virtex-II LVDS current-mode drivers. Virtex-II LVDS current-mode drivers are a true current source and produce the proper (IEEE/EIA/TIA compliant) LVDS

signal. Figure 2-119 illustrates a Virtex-II LVDS transmitter on a board with 50Ω transmission lines.

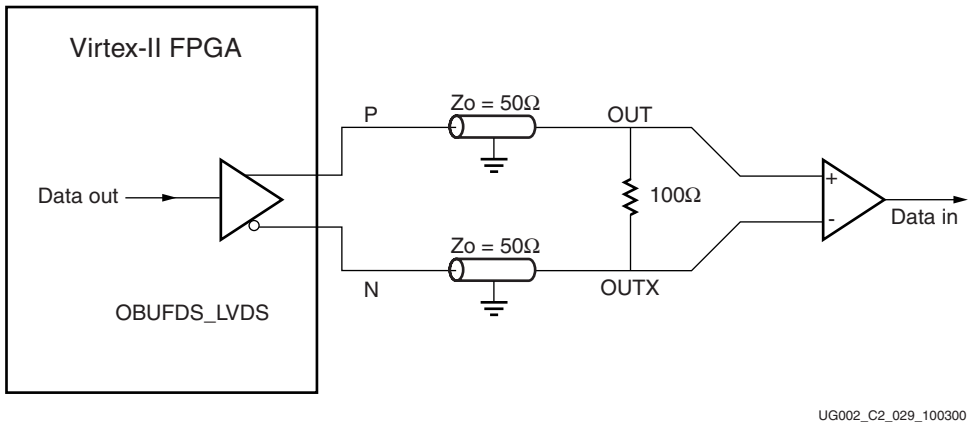


Figure 2-119: **LVDS Transmitter Termination**

Creating an LVDS Output 3-State Buffer

Figure 2-120 illustrates the LVDS 3-State buffer primitives:

- OBUFTDS_LVDS_25
- OBUFTDS_LVDS_33
- OBUFTDS_LVDSEXT_25
- OBUFTDS_LVDSEXT_33

The pin names used are the same as those used in the HDL library primitives.

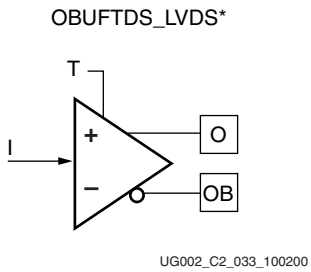


Figure 2-120: **LVDS 3-State Primitives**

To create an LVDS 3-State output, instantiate the desired mode (2.5V, 3.3V, or Extended) LVDS 3-State buffer. Notice that the P and N channels are included in the primitive (O = P, OB = N). Software automatically uses the appropriate pin from an adjacent IOB for the N channel.

LVDS 3-State HDL Example

VHDL Instantiation

```
U1: OBUFTDS_LVDS_25
  port map  (
              I => data_out,
              T => tri,
              O => data_out_P,
              OB => data_out_N
            );
```

Verilog Instantiation

```
OBUFTDS_LVDS_25  U1  (  .I(data_out),
                        .T(tri),
                        .O(data_out_P),
                        .OB(data_out_N)
                     );
```

Port Signals

I = data input to the 3-state output buffer

T = 3-State control signal

O = P-channel data output

OB = N-channel data output

Location Constraints

```
NET "data_out_P" LOC = "NS";
```

LVDS 3-State Termination

The Virtex-II LVDS 3-state buffer does not require any termination. Table 2-59 lists primitives that correspond to Virtex-II LVDS current-mode drivers. These drivers are a true current source, and they produce the proper (IEEE/EIA/TIA compliant) LVDS signal. Figure 2-121 illustrates a simple redundant point-to-point LVDS solution with two LVDS 3-state transmitters sharing a bus with one LVDS receiver and the required termination for the circuit.

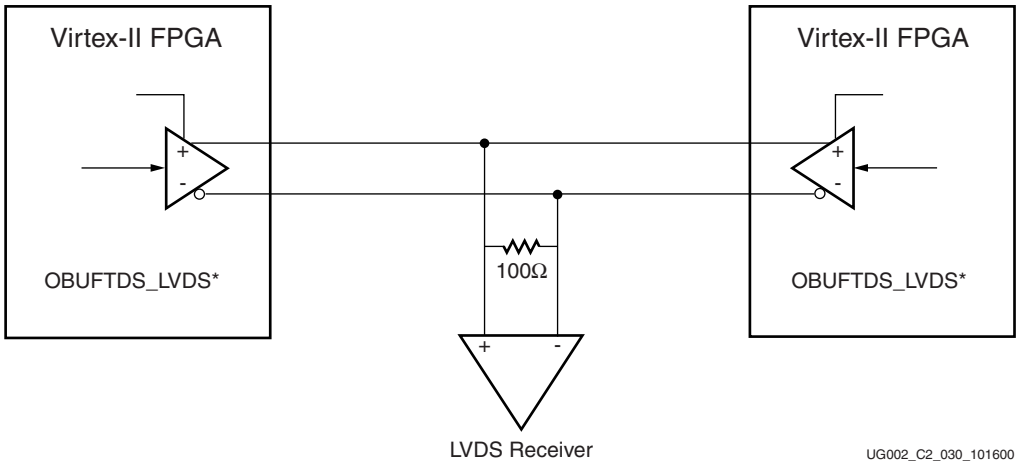


Figure 2-121: **LVDS 3-State Termination**

Creating a Bidirectional LVDS Buffer

Since LVDS is intended for point-to-point applications, BLVDS (Bus-LVDS) is not an IEEE/EIA/TIA standard implementation and requires careful adaptation of I/O and PCB layout design rules. The primitive supplied in the software library for bi-directional LVDS does not use the Virtex-II LVDS current-mode driver. Therefore, source termination is required. Refer to **xapp243** for examples of BLVDS termination.

The following are VHDL and Verilog instantiation examples of Virtex-II BLVDS primitves.

VHDL Instantiation

```
blvds_io: IOBUFDS_BLVDS_25
port map  (
           I => data_out,
           O => data_in,
           T => tri,
           IO => data_IO_P,
           IOB => data_IO_N
          );
```

Verilog Instantiation

```
IOBUFDS_BLVDS_25  blvds_io  (  .I(data_out),
                               .O(data_in),
                               .T(tri),
                               .IO(data_IO_P),
                               .IOB(data_IO_N)
                            );
```

Port Signals

I = data output: internal logic to LVDS I/O buffer

T = 3-State control to LVDS I/O buffer

IO = P-channel data I/O to or from BLVDS pins

IOB = N-channel data I/O to or from BLVDS pins

O = Data input: off-chip data to LVDS I/O buffer

Location Constraints

Only the P or N channel must be constrained. Software automatically places the corresponding channel of the pair on the appropriate pin.

LDT

Lightning Data Transport (LDT) is a new high speed interface and protocol introduced by Advanced Micro Devices. LDT is a differential signaling based interface that is very similar to LVDS. Virtex-II IOBs are equipped with LDT buffers. These buffers also have corresponding software primitives as follows:

IBUFDS_LDT_25

IBUFGDS_LDT_25

OBUFDS_LDT_25

OBUFTDS_LDT_25

LDT Implementation

LDT implementation is the same as LVDS with DDR, so follow all of the rules and guidelines set forth earlier in this chapter for LVDS-DDR, and replace the LVDS buffer with the corresponding LDT buffer. For more information on Virtex-II LDT electrical specification, refer to the **Virtex-II Data Sheet**.

Using Bitstream Encryption

Virtex-II devices have an on-chip decryptor that can be enabled to make the configuration bitstream (and thus the whole logic design) secure. The user can encrypt the bitstream in the Xilinx software, and the Virtex-II chip then performs the reverse operation, decrypting the incoming bitstream, and internally recreating the intended configuration.

This method provides a very high degree of design security. Without knowledge of the encryption/decryption key or keys, potential pirates cannot use the externally intercepted bitstream to analyze, or even to clone the design. System manufacturers can be sure that their Virtex-II implemented designs cannot be copied and reverse engineered. Also, IP Virtex-II chips that contain the correct decryption key.

The Virtex-II devices store the internal decryption keys in a few hundred bits of dedicated RAM, backed up by a small externally connected battery. At <100 nA load, the endurance of the battery is only limited by its shelf life.

The method used to encrypt the data is Data Encryption Standard (DES). This is an official standard supported by the National Institute of Standards and Technology (NIST) and the U. S. Department of Commerce. DES is a symmetric encryption standard that utilizes a 56-bit key. Because of the increased sophistication and speed of today's computing hardware, single DES is no longer considered to be secure. However, the Triple Data Encryption Algorithm (TDEA), otherwise known as triple DES, is authorized for use by U. S. federal organizations to protect sensitive data and is used by many financial institutions to protect their transactions. Triple DES has yet to be cracked. Both DES and triple DES are available in Virtex-II devices.

What DES Is

DES and triple DES are symmetric encryption algorithms. This means that the key to encrypt and the key to decrypt are the same. The security of the data is kept by keeping the key secret. This contrasts to a public key system, like RSA or PGP. One thing to note is that Virtex-II devices use DES in Cipher Block Chaining mode. This means that each block is combined with the previous encrypted block for added security. DES uses a single 56-bit key to encrypt 64-bit blocks one at a time.

How Triple DES is Different

Triple DES uses three keys (known as a key bundle or key set), and the encryption algorithm is repeated for each of those keys. If $E_K(I)$ and $D_K(I)$ denote the encryption and decryption of a data block I using key K, the Triple DES encryption algorithm is as follows (known as E-D-E):

$$Output_{encrypted} = E_{K3}(D_{K2}(E_{K1}(I)))$$

And the decryption algorithm is as follows (known as D-E-D):

$$Output_{decrypted} = D_{K1}(E_{K2}(D_{K3}(I)))$$

$K_1 = K_2 = K_3$ gives the same result as single DES.

For a detailed description of the DES standard, refer to:

http://www.itl.nist.gov/fipspubs/fip46-2.htm

For a popular description of the origin and the basic concept of DES and many other older and newer encryption schemes, see the recent best-seller:

The Code Book by Simon Singh, Doubleday 1999, ISBN 0-385-49531-5

Classification and Export Considerations

Virtex-II FPGAs have been classified by the U. S. Department of Commerce as an FPLD (3A001.a.7), which is the same classification as current FPGAs. Only the decryptor is on-chip and can only be used to decrypt an incoming bitstream, so the classification has not changed and no new paperwork is required. The software has been classified under ECCN#:5D002 and can be exported globally under license exception ENC. No changes to current export practices are necessary.

Creating Keys

For Virtex-II, DES or triple DES (TDEA) can be used. DES uses a single 56-bit key, where triple-DES always uses three such keys. All of the keys can be chosen by the BitGen program at random, or can be explicitly specified by the user.

Virtex-II devices can have six separate keys programmed into the device. A particular Virtex-II device can store two sets of triple-DES keys and can thus accept alternate bitstreams from two competing IP vendors, without providing access to each other's design. However, all of the keys must be programmed at once.

An encrypted bitstream is created by the BitGen program. Keys and key options can be chosen in two ways: by command-line arguments to BitGen, or by specifying a KeyFile (with the –g KeyFile command-line option). The BitGen options relevant to encryption are listed in Table 2-61.:

Table 2-61: **BitGen Encryption Options**

Option	Description	Values (default first where appropriate)
Encrypt	Whether to encrypt the bitstream	No, Yes
Key0	DES Key 0	pick, <hex string>
Key1	DES Key 1	pick, <hex string>
Key2	DES Key 2	pick, <hex string>
Key3	DES Key 3	pick, <hex string>
Key4	DES Key 4	pick, <hex string>
Key5	DES Key 5	pick, <hex string>
KeyFile	Location of separate key definition file	<string>
Keyseq0	Set the key sequence for key 0 (S = single, F = first, M = middle, L = last)	S,F,M,L
Keyseq1	Set the key sequence for key 1	S,F,M,L
Keyseq2	Set the key sequence for key 2	S,F,M,L
Keyseq3	Set the key sequence for key 3	S,F,M,L
Keyseq4	Set the key sequence for key 4	S,F,M,L
Keyseq5	Set the key sequence for key 5	S,F,M,L
StartKey	Key number to start decryption	0,3
StartCBC	Constant Block Chaining start value	pick, <string>

The key sequence (Keyseq) is set to S for single key encryption, F for first key in multi-key encryption, M for middle key in multi-key encryption, and L for last key in multi-key encryption. When the KeyFile option is specified, BitGen looks in that file for all other DES key options listed above. An example for the input KeyFile using triple DES is:

```
# Comment for key file
Key 0 0x9ac28ebeb2d83b;
Key 1 pick;
Key 2 string for my key;
Key 3 0x00000000000000;
Key 4 8774eb3ebb4f84;
Keyseq 0 F;
Keyseq 1 M;
Keyseq 2 L;
Keyseq 3 F;
Keyseq 4 M;
Keyseq 5 L;
Key StartCBC 503f2f655b1b2f82;
StartKey 0;
```


Every key is given in the output key file, with unused key locations set to "0x0000000000000000." The proper key sequence prefix is added for all used keys. The prefix is preserved for unused keys, if the user specified a value. The output key file has the same base file name as the **.bit** file, but with a **.nky** file extension.

The command line equivalent of the input key file above is as follows:

```
bitgen –g Encrypt:Yes –g Key0: 0x9ac28ebeb2d83b –g Key1:pick –g Key2:"
string for my key" –g Key30x00000000000000 –g Key4:8774eb3ebb4f84 –g
Keyseq0:F, -g Keyseq1:M, -gKeyseq2:L –g Keyseq3:F –g Keyseq4:M –g
Keyseq5:L -g StartCBC:503f2f655b1b2f82 –g StartKey:0 myinput.ncd
```

If the key file is used, the command line is as follows:

```
Bitgen –g Encrypt:Yes –g KeyFile: mykeyfile myinput.ncd
```

The output key file from either of the above inputs looks something like this:

```
Device 2v40CS144;
Key 0 0x9ac28ebeb2d83b;
Key 1 0xdb1adb5f08b972;
Key 2 0x5452032773c286;
Key 3 0x00000000000000;
Key 4 0x8774eb3ebb4f84;
Key 5 0x00000000000000;
Keyseq 0 F;
Keyseq 1 M;
Keyseq 2 L;
Keyseq 3 F;
Keyseq 4 M;
Keyseq 5 L;
Key StartCBC 0x503f2f655b1b2f82;
StartKey 0;
```

In the case of the string for Key2, if the keyvalue is a character string, BitGen encodes the string into a 56-bit hex string. The same character string gives the same 56-bit hex string every time. This enables passwords or phrases to be used instead of hex strings.

The above keys are all specified as 64 bits each. The first 8 bits are used by Xilinx as header information and the following 56 bits as the key. BitGen accepts 64 bit keys, but automatically overrides the header, if necessary.

Because of security issues, the **–g** Compress option cannot be used with bitstream encryption. Also, partial reconfiguration is not allowed.

Loading Keys

DES keys can only be loaded through JTAG. The JTAG Programmer and iMPACT™ tools have the capability to take a .nky file and program the device with the keys. In order to program the keys, a "key-access mode" is entered. When this mode is entered, all of the FPGA memory, including the keys and configuration data, is cleared. Once the keys are programmed, they cannot be reprogrammed without clearing the entire device. This "key access mode" is completely transparent to most users.

Keys are programmed using the ISC_PROGRAM instruction, as detailed in the JTAG 1532 specification. SVF generation is also supported, if keys are to be programmed using a different method, such as a microprocessor or JTAG test software.

Loading Encrypted Bitstreams

Once the device has been programmed with the correct keys, the device can be configured with an encrypted bitstream. Non-encrypted bitstreams may also be used to configure the device, and the stored keys are ignored. The method of configuration is not at all affected by encryption. Any of the modes may be used, and the signaling does not change (refer to Chapter 3: Configuration). However, *all* bitstreams must configure the entire device, since partial reconfiguration is not permitted.

Once the device has been configured with an encrypted bitstream, it cannot be reconfigured without toggling the PROG pin, cycling power, or performing the JTAG JSTART instruction. All of these events fully clear the configuration memory, but none of these events reset the keys as long as V_{BATT} or V_{CCAUX} are maintained.

V_{BATT}

V_{BATT} is a separate battery voltage to allow the keys to remain programmed in the Virtex-II device. V_{BATT} draws very little current (on the order of nA) to keep the keys programmed. A small watch battery is suitable (refer to V_{BATT} DC Characteristics in the **Virtex-II Data Sheet** and the battery's specifications to estimate its lifetime).

While the auxiliary voltage (V_{CCAUX}) is applied, V_{BATT} does not draw any current, and the battery can be removed or exchanged.

Using the CORE Generator System

Introduction

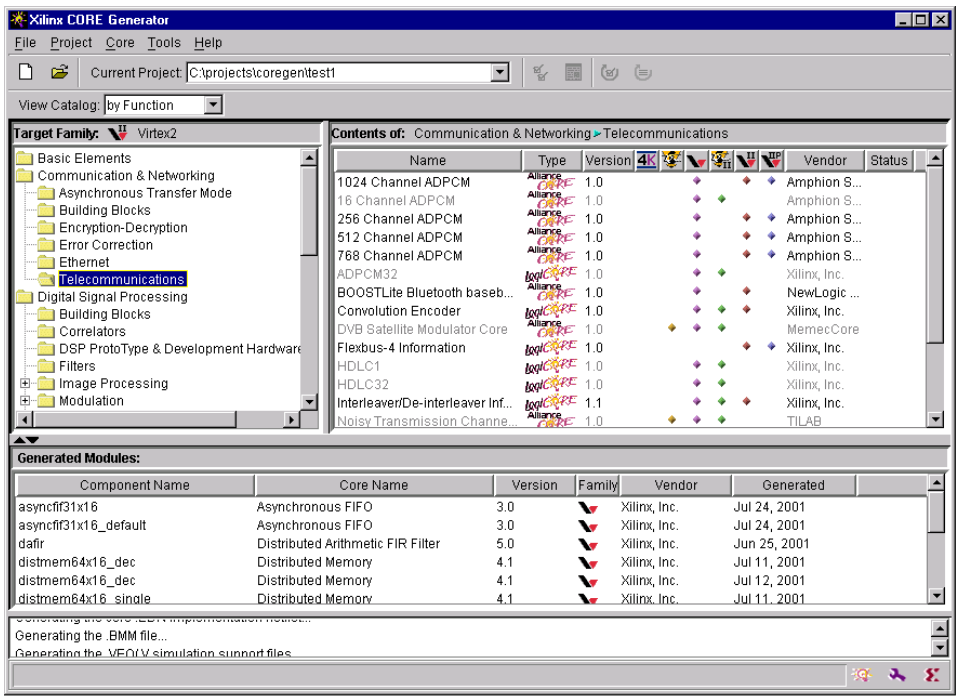
This section on the Xilinx CORE Generator System™ and the Xilinx Intellectual Property (IP) Core offerings is provided as an overview of products that facilitate the Virtex-II design process. For more detailed and complete information, consult the *CORE Generator Guide*, which can be accessed online in the Xilinx software installation, as well as at the http://toolbox.xilinx.com/docsan/xilinx4/manuals.htm site, under the "Design Entry Tools" heading.

The CORE Generator System

The Xilinx CORE Generator System is the cataloging, customization, and delivery vehicle for IP cores targeted to Xilinx FPGAs. This tool is included with all Xilinx ISE BaseX, ISE Foundation, and ISE Alliance Series software packages. The CORE Generator provides centralized access to a catalog of ready-made IP functions ranging in complexity from simple arithmetic operators, such as adders, accumulators, and multipliers, to system-level building blocks, such as filters, transforms, and memories. Cores can be displayed alphabetically, by function, by vendor, or by type. Each core comes with its own data sheet, which documents the core's functionality in detail.

The CORE Generator User Interface (see Figure 2-122) has direct links to key Xilinx web support pages, such as the Xilinx IP Center website (www.xilinx.com/ipcenter) and Xilinx Technical Support, making it very easy to access the latest Virtex-II IP releases and get helpful, up-to-date specifications and information on technical issues. Links to partner IP providers are also built into the informational GUIs for the various partner-supplied AllianceCORE products described under "AllianceCORE Program" on page 331.

The use of CORE Generator IP cores in Virtex-II designs enables designers to shorten design time, and it also helps them realize high levels of performance and area efficiency without any special knowledge of the Virtex-II architecture. The IP cores achieve these high levels of performance and logic density by using Xilinx Smart-IP™ technology.



ug002_c2_068b_100901

Figure 2-122: **Core Generator User Interface**

Smart-IP Technology

Smart-IP technology leverages Xilinx FPGA architectural features, such as look-up tables (LUTs), distributed RAM, segmented routing and floorplanning information, as well as relative location constraints and expert logic mapping to optimize the performance of every core instance in a given Xilinx FPGA design. In the context of Virtex-II cores, Smart-IP technology includes the use of the special high-performance Virtex-II architectural features, such as embedded 18x18 multipliers, block memory, shift register look-up tables (SRL16's), and special wide mux elements.

Smart-IP technology delivers:

- Physical layouts optimized for high performance
- Predictable high performance and efficient resource utilization
- Reduced power requirements through compact design and interconnect minimization
- Performance independent of device size
- Ability to use multiple cores without deterioration of performance
- Reduced compile time over competing architectures

CORE Generator Design Flow

A block diagram of the CORE Generator design flow is shown in Figure 2-123.

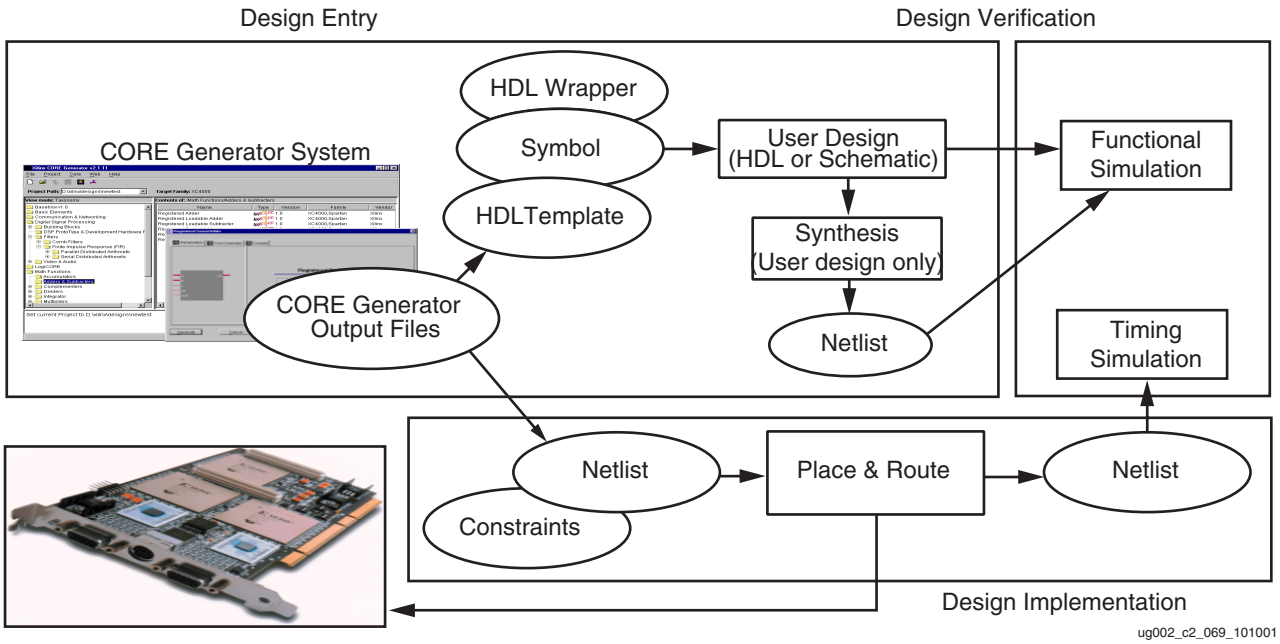


Figure 2-123: **CORE Generator Design Flow**

Note:

1. The outputs produced by the CORE Generator consist of an implementation Netlist and optional schematic symbol, HDL template files, and HDL simulation model wrapper files.

Core Types

Parameterized Cores

The CORE Generator System supplies a wide assortment of parameterized IP cores that can be customized to meet specific Virtex-II design needs and size constraints. See Figure 2-124. For each parameterized core, the CORE Generator System supplies:

- A customized EDIF implementation netlist (.EDN)

- A parameterized Verilog or VHDL behavioral simulation model (.V, .VHD) and corresponding wrapper file (also .V, .VHD)
- Verilog or VHDL templates (.VEO, .VHO)
- An ISE Foundation or Viewlogic® schematic symbol

The EDIF implementation netlist is used by the Xilinx tools to implement the core. The other design files generated depend on the Design Entry settings specified (target CAE vendor, and design flow type -- schematic or HDL). Schematic symbol files are generated when a schematic design flow is specified for the project.

Parameterized HDL simulation models are provided in two separate HDL simulation libraries, one for Verilog functional simulation support, and the other for VHDL functional simulation support. The libraries, which are included as part of the Xilinx installation, are in the following locations:

$XILINX/verilog/src/XilinxCoreLib

$XILINX/vhdl/src/XilinxCoreLib

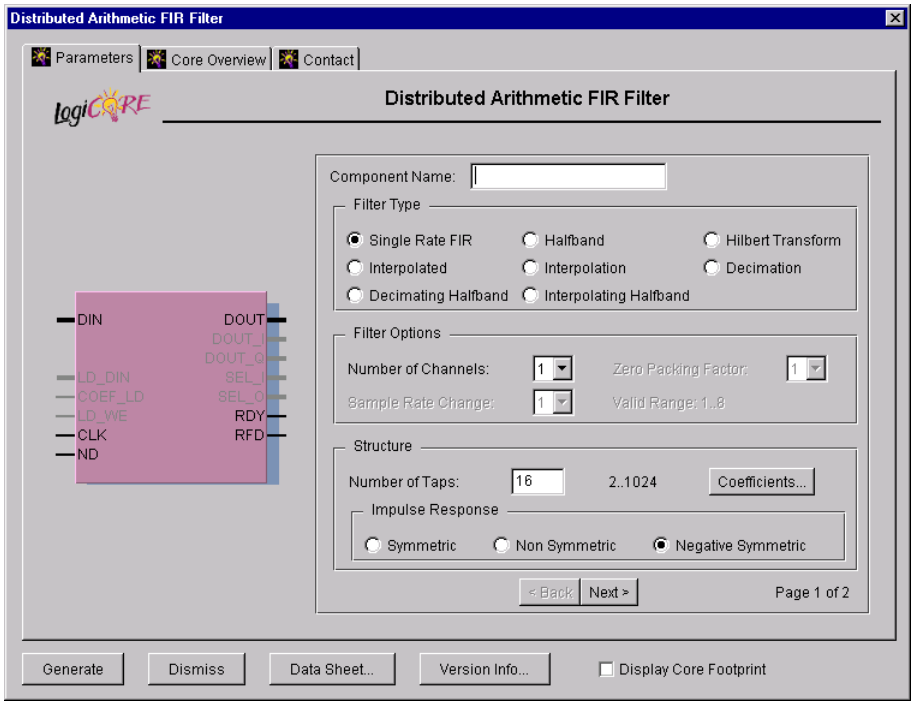


Figure 2-124: **Core Customization Window for a Parameterized Core**

If using a compiled simulator, these libraries must be precompiled before performing a functional simulation of the cores. An analyze_order file describing the required compile order of these models is included with each XilinxCoreLib library, one for Verilog (verilog_analyze_order) and one for VHDL (vhdl_analyze_order).

For an HDL design flow, Verilog and VHDL templates (.VEO and .VHO files) are also provided to facilitate the integration of the core into the design for the purposes of functional simulation, synthesis, and implementation. The Verilog (.V) and VHDL (.VHD) wrapper files are also generated. The wrapper files for a particular core are compiled like normal simulation models. They convey custom parameter values to the corresponding generic, parameterized behavioral model for that core in the XilinxCoreLib library. The custom parameter values are used to tailor the behavior of the customized core.

The following is a sample VHO template:

```
component adder8
    port (
    a: IN std_logic_VECTOR(7 downto 0);
    b: IN std_logic_VECTOR(7 downto 0);
    c: IN std_logic;
    ce: IN std_logic;
    ci: IN std_logic;
    clr: IN std_logic;
    s: OUT std_logic_VECTOR(8 downto 0));
end component;

-- Synplicity black box declaration
attribute black_box : boolean;
attribute black_box of test: component is true;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------

-- The following code must appear in the VHDL architecture
-- body. Substitute your own instance name and net names.

------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG
your_instance_name : adder8
    port map (
    a => a,
    b => b,
    c => c,
    ce => ce,
    ci => ci,
    clr => clr,
    s => s);

-- INST_TAG_END ------ End INSTANTIATION Template ------------
-- You must compile the wrapper file test.vhd when simulating
-- the core, test. When compiling the wrapper file, be sure to
-- reference the XilinxCoreLib VHDL simulation library. For detailed
-- instructions, please refer to the "Core Generator Guide".
```

Fixed Netlist Cores

The other type of Virtex-II core provided by the CORE Generator is the fixed netlist core. These are preset, non-parameterized designs that are shipped with the following:

- A fixed EDIF implementation netlist (as opposed to one that is customized on the fly)
- .VEO and .VHO templates
- Non-parameterized .V and .VHD behavioral simulation models
- Schematic symbol support

Examples include the fixed netlist Xilinx FFTs and most AllianceCORE products.

Since the HDL behavioral models for fixed netlist cores are not parameterized, the corresponding .VEO and .VHO template files are correspondingly simple. They do not need to pass customizing parameter values to a library behavioral model.

Xilinx IP Solutions and the IP Center

The CORE Generator works in conjunction with the Xilinx IP Center on the world wide web to provide the latest IP and software upgrades. To make the most of this resource, Xilinx highly recommends that whenever starting a design, first do a quick search of the Xilinx IP Center (**www.xilinx.com/ipcenter**) to see whether a ready-made core solution is already available.

A complete catalog of Xilinx cores and IP tools resides on the IP Center, including:

- LogiCORE Products
- AllianceCORE Products
- Reference Designs
- XPERTS Partner Consultants
- Design Reuse Tools

When installing the CORE Generator software, the designer gains immediate access to dozens of cores supplied by the LogiCORE Program. In addition, data sheets are available for all AllianceCORE products, and additional, separately licensed, advanced function LogiCORE products are also available. New and updated Virtex-II IP for the CORE Generator can be downloaded from the IP Center and added to the CORE Generator catalog.

LogiCORE Program

LogiCORE products are designed, sold, licensed, and supported by Xilinx. LogiCORE products include a wide selection of generic, parameterized functions, such as muxes, adders, multipliers, and memory cores which are bundled with the Xilinx CORE Generator software at no additional cost to licensed software customers. System-level cores, such as PCI, Reed-Solomon, ADPCM, HDLC, POS-PHY, and Color Space Converters are also available as optional, separately licensed products. Probably, the most common application of the CORE Generator is to use it to quickly generate Virtex-II block and distributed memories. A more detailed listing of available Virtex-II LogiCORE products is available in Table 2-62 and on the Xilinx IP Center website (www.xilinx.com/ipcenter).

Types of IP currently offered by the Xilinx LogiCORE program include:

- Basic Elements: logic gates, registers, multiplexers, adders, multipliers
- Communications and Networking: ADPCM modules, HDLC controllers, ATM building blocks, forward error correction modules, and POS-PHY Interfaces
- DSP and Video Image Processing: cores ranging from small building blocks (e.g., Time Skew Buffers) to larger system-level functions (e.g., FIR Filters and FFTs)
- System Logic: accumulators, adders, subtracters, complementers, multipliers, integrators, pipelined delay elements, single and dual-port distributed and block RAM, ROM, and synchronous and asynchronous FIFOs
- Standard Bus Interfaces: PCI 64/66 (64-bit, 66 MHz), 64/33 (64-bit, 33 MHz), and 32/33 (32-bit, 3 3MHz) Interfaces

AllianceCORE Program

The AllianceCORE program is a cooperative effort between Xilinx and third-party IP developers to provide additional system-level IP cores optimized for Xilinx FPGAs. To ensure a high level of quality, AllianceCORE products are implemented and verified in a Xilinx device as part of the certification process.

Xilinx develops relationships with AllianceCORE partners who can complement the Xilinx LogiCORE product offering. Where Xilinx does not offer a LogiCORE for a particular function, Xilinx partners with an AllianceCORE partner to offer that function. A large percentage of Xilinx AllianceCORE partners focus on data and telecommunication applications, as well as processor and processor peripheral designs.

Together, Xilinx and the AllianceCORE partners are able to provide an extensive library of cores to accelerate the design process. AllianceCORE products include customizable cores which can be configured to exact needs, as well as fixed netlist cores targeted toward specific applications. In many cases, partners can provide cores customized to meet the specific design needs if the primary offerings do not fit the requirements. Additionally, source code versions of the cores are often available from the partners at additional cost for those who need maximum flexibility.

The library of Xilinx and AllianceCORE IP cores allows designers to leverage the expertise of experienced designers who are well-versed in optimizing designs for Virtex-II and other Xilinx architectures. This enables designers to obtain high performance and density in the target Virtex-II device with a faster time to market.

Reference Designs

Xilinx offers two types of reference designs; application notes (XAPPs) developed by Xilinx, and reference designs developed through the Xilinx Reference Design Alliance Program. Both types are extremely valuable to customers looking for guidance when designing systems. Reference designs can often be used as starting points for implementing a broad spectrum of functions in Xilinx programmable logic.

Application notes developed by Xilinx usually include supporting design files. They are supplied free of charge, without technical support or warranty. To see currently available reference designs, visit the **www.xilinx.com/products/logicore/refdes.htm** website.

Reference designs developed through the Xilinx Reference Design Alliance Program are developed, owned, and controlled by the partners in the program. The goal of the program is to form strategic engineering and marketing partnerships with other semiconductor manufacturers and design houses so as to assist in the development of high quality, multicomponent reference designs that incorporate Xilinx devices and demonstrate how they can operate at the system level with other specialized and general purpose semiconductors.

The reference designs in the Xilinx Reference Design Alliance Program are fully functional and applicable to a wide variety of digital electronic systems, including those used for networking, communications, video imaging, and DSP applications. Visit the **www.xilinx.com/company/reference_design/referencepartners.htm** website to see a list of designs currently available through this program.

XPERTS Program

Xilinx established the XPERTS Program to provide customers with access to a worldwide network of certified design consultants proficient with Xilinx Platform FPGAs, software, and IP core integration. All XPERT members are certified and have extensive expertise and experience with Xilinx technology in various vertical applications, such as communications and networking, DSP, video and image processing, system I/O interfaces, and home networking.

XPERTS partners are an integral part of Xilinx strategy to provide customers with cost-efficient design solutions, while accelerating time to market. For more information on Xilinx XPERTS Program, visit the **www.xilinx.com/company/consultants/index.htm** website.

Design Reuse Tools

To facilitate the archiving and sharing of IP created by different individuals and workgroups within a company, Xilinx offers the IP Capture Tool. The IP Capture Tool helps to package design modules created by individual engineers in a standardized format so that they can be cataloged and distributed using the Xilinx CORE Generator. A core can take the form of synthesizable VHDL or Verilog code, or a fixed function netlist. Once it is packaged by the IP Capture Tool and installed into the CORE Generator, the *"captured"* core can be shared with other designers within a company through an internal network. The IP Capture Tool is supplied as a separate utility through the Xilinx IP Center. For more information, see the **www.xilinx.com/ipcenter/designreuse/ipic.htm** website.

CORE Generator Summary

The CORE Generator delivers a complete catalog of IP including behavioral models, synthesis templates, and netlists with performance guaranteed by Xilinx Smart-IP technology. It is a repository for LogiCORE products from Xilinx, AllianceCORE products from Xilinx partners, and it supports Design Reuse for internally developed IP. In addition,

LogiCORE products are continuously updated to add support for new Xilinx architectures, such as Virtex-II. The most current IP updates are available from the Xilinx IP Center.

Utilizing the CORE Generator library of parameterizable cores, designed by Xilinx for Xilinx FPGAs, the designer can enjoy the advantages of design reuse, including faster time to market and lower cost solutions. For more information, visit the Xilinx IP Center www.xilinx.com/ipcenter website.

Virtex-II IP Cores Support

Table 2-62 provides a partial listing of cores available for Virtex-II designs. For a complete catalog of Virtex-II IP, visit the Xilinx IP Center www.xilinx.com/ipcenter website.

Table 2-62: **Virtex-II IP Cores Support**

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
Basic Elements							
BUFE-based Multiplexer Slice	Xilinx	LogiCORE				1-256 bits wide	
BUFT-based Multiplexer Slice	Xilinx	LogiCORE				1-256 bits wide	
Binary Counter	Xilinx	LogiCORE				2-256 bits output width	
Binary Decoder	Xilinx	LogiCORE				2-256 bits output width	
Bit Bus Gate	Xilinx	LogiCORE				1-256 bits wide	
Bit Gate	Xilinx	LogiCORE				1-256 bits wide	
Bit Multiplexer	Xilinx	LogiCORE				1-256 bits wide	
Bus Gate	Xilinx	LogiCORE				1-256 bits wide	
Bus Multiplexer	Xilinx	LogiCORE				IO widths up to 256 bits	
Comparator	Xilinx	LogiCORE				1-256 bits wide	
FD-based Parallel Register	Xilinx	LogiCORE				1-256 bits wide	
FD-based Shift Register	Xilinx	LogiCORE				1-64 bits wide	
LD-based Parallel Latch	Xilinx	LogiCORE				1-256 bits wide	
RAM-based Shift Register	Xilinx	LogiCORE				1-256 bits wide, 1024 words deep	
Communication & Networking							
3G FEC Package	Xilinx	LogiCORE				Viterbi Decoder, Turbo Codec, Convolutional Enc	3G Wireless Infrastructure
3GPP Compliant Turbo Convolutional Decoder	Xilinx	LogiCORE	80%	40	XC2V500	3GPP specs, 2 Mbps, BER=10-6 for 1.5dB SNR	3G Wireless Infrastructure
3GPP Compliant Turbo Convolutional Encoder	Xilinx	LogiCORE	65%	60	XC2V250	Compliant w/ 3GPP, puncturing	3G Wireless Infrastructure
3GPP Turbo Decoder	SysOnChip	AllianceCORE	87%	66	XC2V500-5	3GPP/UMTS compliant, IMT-2000, 2Mbps data	Error correction, wireless

Table 2-62: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
8b/10b Decoder	Xilinx	LogiCORE	1 BRAM	100	XC2V1000	Industry std 8b/10b en/decode for serial data transmission	Physical layer of Fiber Channel
8b/10b Encoder	Xilinx	LogiCORE	1 BRAM	100	XC2V1000	Industry std 8b/10b en/decode for serial data transmission	Physical layer of Fiber Channel
ADPCM 1024 Channel	Amphion	AllianceCORE				G.721, 723, 726, 726a, 727, 727a, u-law, a-law	DECT, VOIP, cordless telephony
ADPCM 256 Channel	Amphion	AllianceCORE				G.721, 723, 726, 726a, 727, 727a, u-law, a-law	DECT, VOIP, cordless telephony
ADPCM 512 Channel	Amphion	AllianceCORE					
ADPCM 768 Channel	Amphion	AllianceCORE	89%	50	XC2V500-5	G.721, 723, 726, 726a, 727, 727a, u-law, a-law	DECT, VOIP, cordless telephony
ADPCM Speech Codec, 32 Channel (DO-DI-ADPCM32)	Xilinx	LogiCORE	62%	25	XC2V500	G.726, G.727, 32 duplex channels	DECT, VOIP, Wireless local loop, DSLAM, PBX
ADPCM Speech Codec, 64 Channel (DO-DI-ADPCM64)	Xilinx	LogiCORE	61%	27	XC2V500	G.726, G.727, 64 duplex channels	DECT, VOIP, wireless local loop, DSLAM, PBX
BOOST LITE Bluetooth Baseband Processor	NewLogic	AllianceCORE	73%	33%	XC2V1000-4	Compliant to Bluetooth v1.1, BQB qualified software for L2CAP, LHP, HC1, voice support	Bluetooth applications
BOOST Lite Bluetooth Baseband Processor	NewLogic	AllianceCORE	73%	33%	XC2V1000-4	Compliant to Bluetooth v1.1, BQB qualified software for L2CAP, LHP, HC1, voice support	Bluetooth applications
Convolutional Encoder	Xilinx	LogiCORE	10%	26	XC2V40	k from 3 to 9, puncturing from 2/3 to 12/13	3G base stations, broadcast, wireless LAN, cable modem, xDSL, satellite com, uwave
DVB-RCS Turbo Decoder	iCODING	AllianceCORE	54%	69	XC2V2000-5	DVB-RCS compliant, 9Mbps, data rate, switchable code rates and frame sizes	Error correction, wireless, DVB, Satellite data link
Flexbus 4 Interface Core, 16-Channel (DO-DI-FLX4C16)	Xilinx	LogiCORE	31%	200	XC2V3000 FG676-5		Line card: terabit routers & optical switches
Flexbus 4 Interface Core, 4-Channel (DO-DI-FLX4C4)	Xilinx	LogiCORE	27%	200	XC2V1000 FG456-5		Line card: terabit routers & optical switches
Flexbus 4 Interface Core, 1-Channel (DO-DI-FLX4C1)	Xilinx	LogiCORE	12%	200	XC2V1000 FG456-5		Line card: terabit routers & optical switches

Table 2-62: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
HDLC Controller Core, 32 Channels	Xilinx	LogiCORE	34%	81	XC2V250	32 full duplex, CRC-16/32, 8/16-bit address insertion/deletion	X.25, POS, cable modems, frame relay switches, video confer. over ISDN
HDLC Controller Core, Single Channel	Xilinx	LogiCORE	15%	115	XC2V250	16/32-bit frame seq, 8/16-bit addr insert/delete, flag/zerop insert/detect	X.25, POS, cable modems, frame relay switches, video conf. over ISDN
Interleaver/De-interleaver	Xilinx	LogiCORE	30%	187	XC2V40	Convolutional, width up to 256 bits, 256 branches	Broadcast, wireless LAN, cable modem, xDSL, satellite com,uwave nets, digital TV
PE-MACMII Dual Speed 10/100 Mbps Ethernet MAC	Alcatel	AllianceCORE	33%	60	XC2V500-4	802.3 compliant, Supports single & multimode fiber optic devices, M11 interfaces, RMON and Etherstate statistics	Networking, Broadband, NIC, SOHO, Home networking, storage, routers, switches, printers,
POS-PHY Level 3 Link Layer Interface Core, 48 Channel (DO-DI-POSL3LINK48A)	Xilinx	LogiCORE	33%	104	XC2V6000 FF1152-4		
POS-PHY L3 Link Layer Interface, 16-Ch (DO-DI-POSL3LINK16)	Xilinx	LogiCORE	40%	104	XC2V1000 FG456-4		Line card: terabit routers & optical switches
POS-PHY L3 Link Layer Interface, 4-Ch (DO-DI-POSL3LINK4)	Xilinx	LogiCORE	15%	104	XC2V1000 FG456-4		Line card: terabit routers & optical switches
POS-PHY L3 Link Layer Interface, 2-Ch (DO-DI-POSL3LINK2)	Xilinx	LogiCORE	55%	104	XCV50E-8		Line card: terabit routers & optical switches
POS-PHY L3 Link Layer Interface, Single Channel	Xilinx	LogiCORE	6%	104	XC2V1000 FG456-4		
POS-PHY L4 Multi-Channel Interface (DO-DI-POSL4MC)	Xilinx	LogiCORE	29%	104	XC2V3000 FG676-5		
Reed-Solomon Decoder	Xilinx	LogiCORE	40%	98	XC2V250	Std or custom coding, 3-12 bit symbol width, up to 4095 symbols	Broadcast, wireless LAN, digital TV, cable modem, xDSL, satellite com,uwave nets
Reed-Solomon Decoder	TILAB	AllianceCORE	56%	61	XC2V1000-5	parameterizable, RTL available	Error correction, wireless, DSL

Table 2-62: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
Reed-Solomon Encoder	Xilinx	LogiCORE	42%	180	XC2V40	Std or cust coding, 3-12 bit width, up to 4095 symbols with 256 check symb.	Broadcast, wireless LAN, digital TV, cable modem, xDSL, satellite com,uwave nets
SDLC Controller	CAST	AllianceCORE	38%	158	XC2V100-5	Like Intel 8XC152 Global Serial Channel, Serial Comm., HDLC apps, telecom	Embedded systems, professional audio, video
SPEEDROUTER Network Processor	IP	AllianceCORE	64%	80 MHz, 2.5 Gbps	XC2V1500-5	Solution requires SPEEDAnalyzer ASIC, 2.5 Gbps fdx wire speed; net processor (NPV)	Networking, edge and access, Switches and routers
Turbo Decoder - 3GPP	SysOnChip	AllianceCORE	88%	65	XC2V2000-5	3GPP/UMTS compliant, 2Mbps data rate	Error correction, wireless
Turbo Encoder	TILAB	AllianceCORE	48%	120	XC2V80-5	3GPP/UMTS compliant, upto 4 interleaver laws	Error correction, wireless
TURBO_DEC Turbo Decoder	TILAB	AllianceCORE	99%	65	XC2V2000-5	3GPP/UMTS compliant, >2Mbps data rate	Error correction, wireless
Viterbi Decoder	Xilinx	LogiCORE	80%	100	XC2V250	Puncturing, serial & parallel architecture,	3G base stations, broadcast, wireless LAN, cable modem, xDSL, satellite com, uwave
Viterbi Decoder, IEEE 802-compatible	Xilinx	LogiCORE	70%	147	XC2V250	Constraint length(k)=7, G0=171, G1=133	L/MMDS, cable modem, broadcast equip, wireless LAN, xDSL, sat com, uwave nets
Digital Signal Processing							
1024-Point Complex FFT IFFT for Virtex-II	Xilinx	LogiCORE	62%	41us, 100 MHz	XC2V500	16 bit complex data, 2's comp, forward and inverse transform	
16-Point Complex FFT IFFT for Virtex-II	Xilinx	LogiCORE	37%	123ns, 130 MHz	XC2V500	16 bit complex data, 2's comp, forward and inverse transform	
256-Point Complex FFT IFFT for Virtex-II	Xilinx	LogiCORE	54%	7.7us, 100 MHz	XC2V500	16 bit complex data, 2's comp, forward and inverse transform	
32 Point Complex FFT/IFFT	Xilinx	LogiCORE					
64-Point Complex FFT IFFT for Virtex-II	Xilinx	LogiCORE	38%	1.9us, 100 MHz	XC2V500	16 bit complex data, 2's comp, forward and inverse transform	
Bit Correlator	Xilinx	LogiCORE				4096 taps, serial/parallel input, 4096 bits width	
Cascaded Integrator Comb (CIC)	Xilinx	LogiCORE				32 bits data width, rate change from 8 to 16384	

Table 2-62: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
Direct Digital Synthesizer	Xilinx	LogiCORE				8-65K samples, 32-bits output precision, phase dithering/offset	
Distributed Arithmetic FIR Filter	Xilinx	LogiCORE				32-bit input/coeff width, 1024 taps, 1-8 chan, polyphase, online coeff reload	
GVA-300 Virtex-II DSP Hardware Accelerator	GV	AllianceCORE	NA	NA		2 Virtex-II, Spartan-II FPGAs, 1 CPLD, Matlab I/F	DSP prototyping
LFSR, Linear Feedback Shift Register	Xilinx	LogiCORE				168 input widths, SRL16/register implementation	
Math Functions							
Accumulator	Xilinx	LogiCORE				1-256s bit wide	
Adder Subtracter	Xilinx	LogiCORE				1-256s bit wide	
DFP2INT Floating Point to Integer Converter	Digital	AllianceCORE	39%	66	XC2V250-5	Full IEEE-754 compliance, 4 pipelines, Single precision real format support	DSP, Math, Arithmetic apps
DFPADD Floating Point Adder	Digital	AllianceCORE	39%	66	XC2V250-5	Full IEEE-754 compliance, 4 pipelines, Single precision real format support	DSP, Math, Arithmetic apps
DFPCOMP Floating Point Comparator	Digital	AllianceCORE	16%	91	XC2V80-5	Full IEEE-754 compliance, 4 pipelines, Single precision real format support	DSP, Math, Arithmetic apps.
DFPDIV Floating Point Divider	Digital	AllianceCORE	99%	53	XC2V250-5	Full IEEE-754 compliance, 15 pipelines, Single precision real format support	DSP, Math, Arithmetic apps
DFPMUL Floating Point Multiplier	Digital	AllianceCORE	44%	74	XC2V250-5	Full IEEE-754 compliance, 7 pipelines,32x32 mult, Single precision real format support	DSP, Math, Arithmetic apps.
DFPSQRT Floating Point Square Root	Digital	AllianceCORE	39%	66	XC2V250-5	Full IEEE-754 compliance, 4 pipelines, Single precision real format support	DSP, Math, Arithmetic apps
DINT2FP Integer to Floating Point Converter	Digital	AllianceCORE	37%	73	XC2V250-5	Full IEEE-754 compliance, double word input, 2 pipelines, Single precision real output	DSP, Math, Arithmetic apps
Multiply Accumulator (MAC)	Xilinx	LogiCORE				Input width up to 32 bits, 65-bit accumulator, truncation rounding	

Table 2-62: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
Multiply Generator	Xilinx	LogiCORE				64-bit input data width, constant, reloadable or variable inputs, parallel/sequential implementation	
Pipelined Divider	Xilinx	LogiCORE				32-bit input data width, multiple clock per output	
Sine Cosine Look Up Table	Xilinx	LogiCORE				3-10 bit in, 4-32 bit out, distributed/block ROM	
Twos Complementer	Xilinx	LogiCORE				Input width up to 256 bits	
Memories & Storage Elements							
Asynchronous FIFO	Xilinx	LogiCORE				1-256 bits, 15-65535 words, DRAM or BRAM, independent I/O clock domains	
Content Addressable Memory (CAM)	Xilinx	LogiCORE				1-512 bits, 2-10K words, SRL16	
Distributed Memory	Xilinx	LogiCORE				1-1024 bit, 16-65536 word, RAM/ROM/SRL16, opt output regs and pipelining	
Dual-Port Block Memory	Xilinx	LogiCORE				1-256 bits, 2-13K words	
Single-Port Block Memory	Xilinx	LogiCORE				1-256 bits, 2-128K words	
Synchronous FIFO	Xilinx	LogiCORE				1-256 bits, 16-256 words, distributed/block RAM	
Microprocessors, Controllers & Peripherals							
10/100 Ethernet MAC	Xilinx	LogiCORE				Interfaces through OPB to MicroBlaze	Networking, comm., processor applications
AX1610 16-bit RISC Processor	Loarant	AllianceCORE	12%	91	XC2V500-5	44 opcode, 64-K word data, program, Harvard arch.	Control functions, State mach, Coprocessor
C165X MicroController	CAST	AllianceCORE	60%	134	XC2V80-5	Microchip 16C5X PIC like	Embedded systems, telecom
C68000 Microprocessor	CAST	AllianceCORE	90%	32	XC2V500-5	MC68000 Compatible	Embedded systems, pro audio, video
CPU FPGA (Virtex-II) MicroEngine Cards	NMI	AllianceCORE	NA	NA	NA	Hitachi SH-3 CPU	Embedded systems
CZ80CPU Microprocessor	CAST	AllianceCORE	55%	72	XC2V500-5	Zilog Z80 compatible, 8-bit processor	Embedded systems, Communications

Table 2-62: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
DDR SDRAM Controller Core	Memec-Core	AllianceCORE	7%	133	XC2V1000-4	DDR SDRAM burst length support for 2,4,8 per access, supports data 16,32, 64, 72.	Digital video, embedded computing , networking
DFPIC125X Fast RISC MicroController	Digital	AllianceCORE	49%	126	XC2V80-5	PIC 12c4x like, 2X faster, 12-bit wide instruction set, 33 instructions	Embedded systems, telecom, audio and video
DFPIC1655X Fast RISC MicroController	Digital	AllianceCORE	79%	140	XC2V80-5	S/W compatible with PIC16C55X, 14-bit instruction set, 35 instructions	Embedded systems, telecom, audio and video
DFPIC165X Fast RISC MicroController	Digital	AllianceCORE	49%	126	XC2V80-5	PIC 12c4x like, 2X faster, 12-bit wide instruction set, 33 instructions	Embedded systems, telecom, audio and video
DI2CM I2C Bus Controller Master	Digital	AllianceCORE	58%	143	XC2V50-5	I2C-like, multi master, fast/std. modes	Embedded systems
DI2CM I2C Bus Controller Slave	Digital	AllianceCORE	28%	157	XC2V50-5	I2C-like, Slave	Embedded
DI2CSB I2C Bus Controller Slave Base	Digital	AllianceCORE	15%	187	XC2V50-5	I2C-like, Slave	Embedded Systems
DR8051 RISC MicroController	Digital	AllianceCORE	68%	73	XC2V250-5	80C31 instruction set, RISC architecture 6.7X faster than standard 8051	Embedded systems, telecom, video
DR8051BASE RISC MicroController	Digital	AllianceCORE	46%	80-90	XC2V250-5	80C31 instruction set, high speed multiplier, RISC architecture 6.7X faster than standard 8051	Embedded systems, telecom, video
DR8052EX RISC MicroController	Digital	AllianceCORE	99%	71	XC2V250-5	80C31 instruction set, high speed mult/div ,RISC 6.7X faster than standard 8051	Embedded systems, telecom, video
e8254 Programmable Interval Timer/Counter	einfochips	AllianceCORE	1%	175	XC2V1000-5	Three 8-bit parallel ports, 24 programmable IO lines, 8-bit bidi data bus	Processor, I/O interface
e8255 Peripheral Interface	einfochips	AllianceCORE	1%	175	XC2V1000-5	Three 8-bit parallel ports, 24 programmable IO lines, 8-bit bidi data bus	Processor, I/O interface
Flip805x-PS Microprocessor	Dolphin	AllianceCORE	39%	38	XC2V1000-5	Avg 8X faster & code compatible v. legacy 8051, verification bus monitor, SFR IF, DSP focused	DSP, Telecom, industrial, high speed control
IIC	Xilinx	LogiCORE				Interfaces through OPB to MicroBlaze	Networking, com, processor applic
LavaCORE Configurable Java Processor Core	Derivation	AllianceCORE	38%	20	XC2V1000-5	32b data/address optional DES	Internet appliance, industrial control
LavaCORE Configurable Java Processor Core	Derivation	AllianceCORE	38%	20	XC2V1000-5	32b data/address optional DES	Internet appliance, industrial control

Table 2-62: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
Lightfoot 32-bit Java Processor Core	Digital	AllianceCORE	33%	40	XC2V1000-5	32bit data, 24 bit address, 3 Stage pipeline, Java/C dev. tools	Internet appliance, industrial control, HAVi multimedia, set top boxes
MicroBlaze Soft RISC Processor	Xilinx	LogiCORE		125		Soft RISC Processor, small footprint	Networking, communications
OPB Arbiter	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB GPIO	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB Interrupt Controller	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB Memory Interface (Flash, SRAM)	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB Timer/Counter	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB UART (16450, 16550)	Xilinx	LogiCORE		125		Interfaces through OPB to MicroBlaze	Processor applications
OPB UART Lite	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB WDT	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
PF3100 PC/104-Plus Reconfigurable Module	Derivation	AllianceCORE	N/A	N/A	XC2V1000 FG256	PC/104 & PC/104+ devlopment board	Internet appliance, industrial control
SPI	Xilinx	LogiCORE				Interfaces through OPB to MicroBlaze	Networking, communications, processor applications
XF-UART Asynchronous Communications Core	Memec-Core	AllianceCORE	15%	50	XCS20-4	UART and baud rate generator	Serial data communication
Standard Bus Interfaces							
PCI-X 64/100 Interface for Virtex-II (DO-DI-PCIX64-VE)	Xilinx	LogiCORE	30%	100	XC2V1000 FG456-5	PCI-X 1.0 comp, 64/32-bit, 66 MHz PCI-X initiator and target IF, PCI 2.2 comp, 64/32-bit, 33 MHz PCI initiator and target IF, 3.3 V PCI-X at 33-66 MHz, 3.3 V PCI at 0-33 MHz	Server,Embedded, gb ethernet,U320 SCSI,Fibre Ch,RAID cntl,graphics

Table 2-62: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
PCI32 Virtex Interface Design Kit (DO-DI-PCI32-DKT)	Xilinx	LogiCORE	6%	66	XC2V1000 FG456-5	Includes PCI32 board, drive development kit, and customer education 3-day training class	
PCI32 Virtex Interface, IP Only (DO-DI-PCI32-IP)	Xilinx	LogiCORE	6%	66	XC2V1000 FG456-5	v2.2 comp, assured PCI timing, 3.3/5-V, 0-waitstate, CPCI hot swap friendly	PC add-in boards, CPCI, Embedded
PCI64 & PCI32, IP Only (DO-DI-PCI-AL)	Xilinx	LogiCORE	6 - 7%	66	XC2V1000 FG456-5	v2.2 comp, assured PCI timing, 3.3/5-V, 0-waitstate, CPCI hot swap friendly	PC boards,CPCI,Embedded,hiperf video,gb ethernet
PCI64 Virtex Interface Design Kit (DO-DI-PCI64-DKT)	Xilinx	LogiCORE	7%	66	XC2V1000 FG456-5	v2.2 comp, assured PCI timing, 3.3/5-V, 0-waitstate, CPCI hot swap friendly	PC boards, CPCI, Embedded, hiperf video, gb ethernet
PCI64 Virtex Interface, IP Only (DO-DI-PCI64-IP)	Xilinx	LogiCORE	7%	66	XC2V1000 FG456-5	v2.2 comp, assured PCI timing, 3.3/5-V, 0-waitstate, CPCI hot swap friendly	PC boards,CPCI,Embedded,hiperf video,gb ethernet
RapidIO 8-bit port LP-LVDS Phy Layer (DO-DI-RIO8-PHY)	Xilinx	LogiCORE	24%	250	XC2V1000 FG456-5	RapidIO Interconnect v1.1 compliant, verified with Motorola's RapidIO bus functional model v1.4	Routers, switches, backplane, control plane, data path, embedded sys, high speed interface to memory and encryption engines, high end video
USB 1.1 Device Controller	Memec-Core	AllianceCORE	21%	12	XC2V1000-5	Compliant with USB1.1 spec., Supports VCI bus, Performs CRC, Supports 1.5 Mbps & 12 Mbps	Scanners, Printers, Handhelds, Mass Storage
Video & Image Processing							
1-D Discrete Cosine Transform	Xilinx	LogiCORE				8-24 bits for coeff & input, 8-64 pts	
2-D DCT/IDCT Forward/Inverse Discrete Cosine Transform	Xilinx	LogiCORE					image, video phone, color laser printers
FASTJPEG_BW Decoder	BARCO-SILEX	AllianceCORE	67%	73	XC2V1000-4	Conforms to ISO/IEC Baseline 10918-1, Gray-Scale	Video editing, digital camera, scanners
FASTJPEG_C Decoder	BARCO-SILEX	AllianceCORE	78%	56	XC2V1000-4	Conforms to ISO/IEC Baseline 10918-1, color, multi-scan, Gray-Scale	Video editing, digital camera, scanners