# Spartan™-3 FPGA Handbook

**July 11, 2003**


This book provides a snapshot of the Spartan-3 technical documentation at the time of printing. For the latest information and technical support, see the Xilinx website at http://www.xilinx.com.

- Spartan-3 Product Information
  - Product overview, links to latest data sheets and application notes
  - http://www.xilinx.com/spartan3
- Answers Database
  - Search database of silicon and software questions and answers
  - http://www.xilinx.com/support/answers.htm
- WebCase Technical Support
  - Create technical support cases, attach design files, review case updates in real time
  - http://www.xilinx.com/support/clearexpress/websupport.htm

XILINX®





## Spartan™-3 FPGA Handbook
## July 11, 2003

The following table shows the revision history for this document..

| | Version | Revision |
|---|---|---|
| 07/11/03 | 1.0a | Initial Xilinx release. |

# *Foreword*

Xilinx, the leader in programmable logic, redefines the logic landscape with the Spartan-3 FPGA family. Built on four generations of proven Spartan success in high volume applications, Spartan-3 FPGAs leverage advanced 90nm technology to give you up to five million system gates with the lowest cost per gate and per I/O of any FPGA. This Spartan-3 FPGA Handbook provides a complete collection of technical documentation for evaluating and using this exciting family. For the latest updates and technical support, see the Xilinx web site at http://www.xilinx.com.

*Kapil Shankar*

Sr. Director, General Products Division Marketing and Applications Engineering

# Acknowledgments

## The Spartan-3 FPGA Handbook Hoplites

**Content Creation/Authors:**

- Marc Baker
- Kim Goldblatt
- Steven Knapp

**Contributors:**

- David Anderson
- Chris Arndt
- Michol Bauer
- Justin Cammon
- Hari Devanath
- Paul Kirk
- Chris Mead
- Stephen Neuhold
- Mark Noble
- Kamal Patel
- Bryan Ramirez
- Shalin Sheth
- Elliot Schei
- Jon Schimeck
- Premduth Vidyanandan
- Roy White
- Derrick Woods
- Henke Yunkins

**Cover Art:**

- Russ Jordan
- Sal Randazzo

**Documentation Services:**

- Elaine Hadad, Technicalities, Inc.

Special thanks go to these teams at Xilinx for all their support:

- Global Services Division (GSD) Applications Engineering
- Advanced Products Division (APD) Applications Engineering
- General Products Division (GPD) System and Design Engineering

XILINX®


SPARTAN™-3
FPGA
MAKE IT YOUR ASIC

# *Table of Contents*

## Index & Sales Office Listing

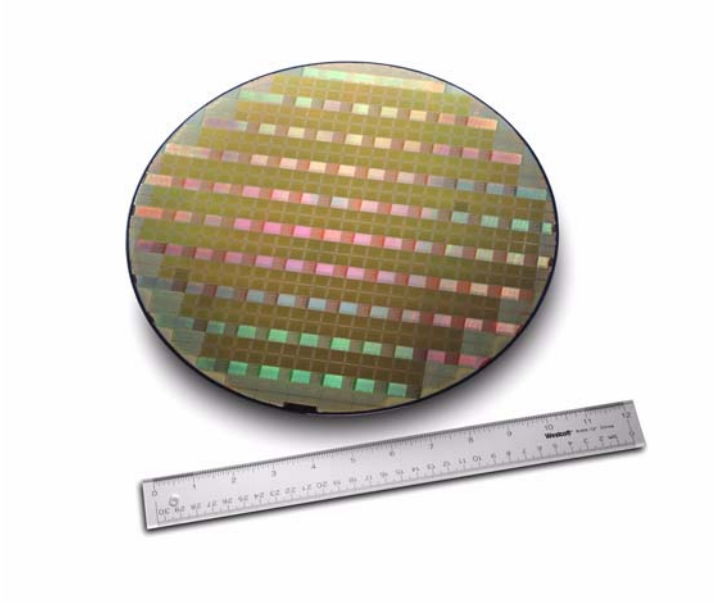
# Spartan-3 Data Sheet

## Introduction and Ordering Information

## Functional Description

## DC and Switching Characteristics

## Pinout Descriptions [Abbreviated]



**90 nm technology + 300 mm wafers = The most gates, I/Os, and advanced features at the lowest price**

# Spartan-3 1.2V FPGA Family: Introduction and Ordering Information

DS099-1 (v1.1) April 24, 2003

**Advance Product Specification**

## Introduction

The 1.2V Spartan™-3 family of Field-Programmable Gate Arrays is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The eight-member family offers densities ranging from 50,000 to five million system gates, as shown in Table 1.

The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions. Numerous enhancements derive from state-of-the-art Virtex™-II technology. These Spartan-3 enhancements, combined with advanced process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry.

Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection and digital television equipment.

The Spartan-3 family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

## Features

- Revolutionary 90-nanometer process technology
- Very low cost, high-performance logic solution for high-volume, consumer-oriented applications
  - Densities as high as 74,880 logic cells
  - 326 MHz system clock rate
  - Three separate power supplies for the core (1.2V), I/Os (1.2V to 3.3V), and special functions (2.5V)
- SelectIO™ signaling
  - Up to 784 I/O pins
  - 622 Mb/s data transfer rate per I/O
  - Seventeen single-ended signal standards
  - Six differential signal standards including LVDS
  - Termination by Digitally Controlled Impedance
  - Signal swing ranging from 1.14V to 3.45V
  - Double Data Rate (DDR) support
- Logic resources
  - Abundant, flexible logic cells with registers
  - Wide multiplexers
  - Fast look-ahead carry logic
  - Dedicated 18 x 18 multipliers
  - JTAG logic compatible with IEEE 1149.1/1532 standards
- SelectRAM™ hierarchical memory
  - Up to 1,872 Kbits of total block RAM
  - Up to 520 Kbits of total distributed RAM
- Digital Clock Manager (up to four DCMs)
  - Clock skew elimination
  - Frequency synthesis
  - High resolution phase shifting
- Eight global clock lines and abundant routing
- Fully supported by Xilinx ISE development system
  - Synthesis, mapping, placement and routing

*Table 1:* **Summary of Spartan-3 FPGA Attributes**

| Device | System Gates | Logic Cells | CLB Array (One CLB = Four Slices) | | | Distributed RAM (bits[1]) | Block RAM (bits[1]) | Dedicated Multipliers | DCMs | Maximum User I/O | Maximum Differential I/O Pairs |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Rows | Columns | Total CLBs | | | | | | |
| XC3S50 | 50K | 1,728 | 16 | 12 | 192 | 12K | 72K | 4 | 2 | 124 | 56 |
| XC3S200 | 200K | 4,320 | 24 | 20 | 480 | 30K | 216K | 12 | 4 | 173 | 76 |
| XC3S400 | 400K | 8,064 | 32 | 28 | 896 | 56K | 288K | 16 | 4 | 264 | 116 |
| XC3S1000 | 1M | 17,280 | 48 | 40 | 1,920 | 120K | 432K | 24 | 4 | 391 | 175 |
| XC3S1500 | 1.5M | 29,952 | 64 | 52 | 3,328 | 208K | 576K | 32 | 4 | 487 | 221 |
| XC3S2000 | 2M | 46,080 | 80 | 64 | 5,120 | 320K | 720K | 40 | 4 | 565 | 270 |
| XC3S4000 | 4M | 62,208 | 96 | 72 | 6,912 | 432K | 1,728K | 96 | 4 | 712 | 312 |
| XC3S5000 | 5M | 74,880 | 104 | 80 | 8,320 | 520K | 1,872K | 104 | 4 | 784 | 344 |

**Notes:**
1. By convention, one Kb is equivalent to 1,024 bits.

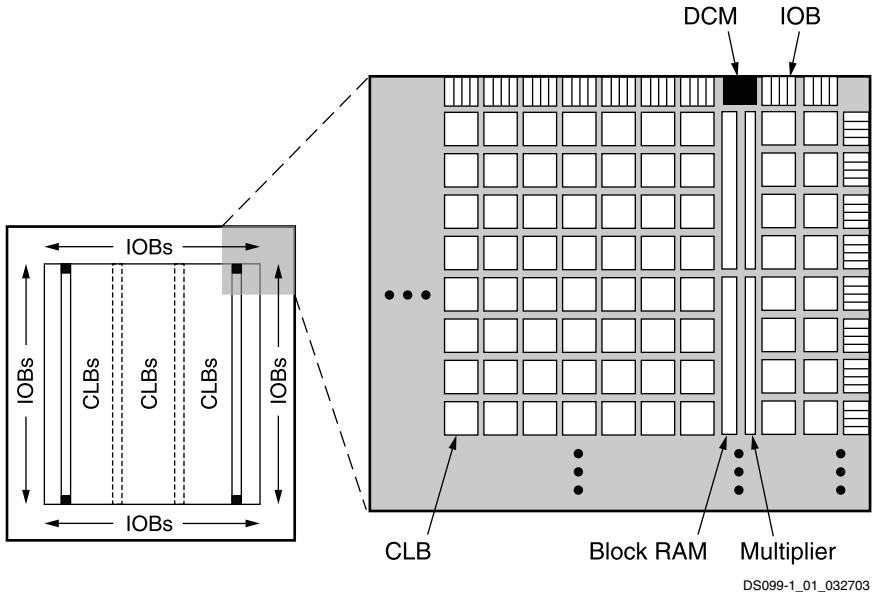
## Architectural Overview

The Spartan-3 family architecture consists of five fundamental programmable functional elements:

- Configurable Logic Blocks (CLBs) contain RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip-flops or latches. CLBs can be programmed to perform a wide variety of logical functions as well as to store data.
- Input/Output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Twenty-three different signal standards, including six high-performance differential standards, are available as shown in Table 2. Double Data-Rate (DDR) registers are included. The Digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.
- Block RAM provides data storage in the form of 18-Kbit dual-port blocks.
- Multiplier blocks accept two 18-bit binary numbers as inputs and calculate the product.
- Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase shifting clock signals.

These elements are organized as shown in Figure 1. A ring of IOBs surrounds a regular array of CLBs. The XC3S50 has a single column of block RAM embedded in the array. Those devices ranging from the XC3S200 to the XC3S2000 have two columns of block RAM. The XC3S4000 and XC3S5000 devices have four RAM columns. Each column is made up of several 18K-bit RAM blocks; each block is associated with a dedicated multiplier. The DCMs are positioned at the ends of each block RAM column.

The Spartan-3 family features a rich network of traces and switches that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.




**Notes:**
1. The two additional block RAM columns of the XC3S4000 and XC3S5000 devices are shown with dashed lines. The XC3S50 has only the block RAM column on the far left.

*Figure 1:* **Spartan-3 Family Architecture**

## Configuration

Spartan-3 FPGAs are programmed by loading configuration data into robust static memory cells that collectively control all functional elements and routing resources. Before powering on the FPGA, configuration data is stored externally in a PROM or some other nonvolatile medium either on or off the board. After applying power, the configuration data is written to the FPGA using any of five different modes: Master Parallel, Slave Parallel, Master Serial, Slave Serial and Boundary Scan (JTAG). The Master and Slave Parallel modes use an 8-bit wide SelectMAP™ Port.

The recommended memory for storing the configuration data is the low-cost Xilinx Platform Flash PROM family, which includes XCF00S PROMs for serial configuration and XCF00P PROMs for parallel configuration.

## I/O Capabilities

The SelectIO feature of Spartan-3 devices supports 17 single-ended standards and six differential standards as listed in Table 2. Table 3 shows the number of user I/Os as well as the number of differential I/O pairs available for each device/package combination.

*Table 2:* **Signal Standards Supported by the Spartan-3 Family**

| Standard Category | Description | $V_{CCO}$ (V) | Class | Symbol |
|---|---|---|---|---|
| **Single-Ended** | | | | |
| GTL | Gunning Transceiver Logic | N/A | Terminated | GTL |
| | | | Plus | GTLP |
| HSTL | High-Speed Transceiver Logic | 1.5 | I | HSTL_I |
| | | | III | HSTL_III |
| | | 1.8 | I | HSTL_I_18 |
| | | | II | HSTL_II_18 |
| | | | III | HSTL_III_18 |
| LVCMOS | Low-Voltage CMOS | 1.2 | N/A | LVCMOS12 |
| | | 1.5 | N/A | LVCMOS15 |
| | | 1.8 | N/A | LVCMOS18 |
| | | 2.5 | N/A | LVCMOS25 |
| | | 3.3 | N/A | LVCMOS33 |
| LVTTL | Low-Voltage Transistor-Transistor Logic | 3.3 | N/A | LVTTL |
| PCI | Peripheral Component Interconnect | 3.0 | 33 MHz | PCI33_3 |
| SSTL | Stub Series Terminated Logic | 1.8 | N/A | SSTL18_I |
| | | 2.5 | I | SSTL2_I |
| | | | II | SSTL2_II |
| **Differential** | | | | |
| LDT | Lightning Data Transport (HyperTransport™) | 2.5 | N/A | LDT_25 |
| LVDS | Low Voltage Differential Signaling | | Standard | LVDS_25 |
| | | | Bus | BLVDS_25 |
| | | | Extended Mode | LVDSEXT_25 |
| | | | Ultra | ULVDS_25 |
| RSDS | Reduced-Swing Differential Signaling | 2.5 | N/A | RSDS_25 |

*Table 3:* **Spartan-3 User I/O Chart**

| Device | Available User I/Os and Differential (Diff) I/O Pairs | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | VQ100 | | TQ144 | | PQ208 | | FT256 | | FG456 | | FG676 | | FG900 | | FG1156 | |
| | User | Diff | User | Diff | User | Diff | User | Diff | User | Diff | User | Diff | User | Diff | User | Diff |
| XC3S50 | 63 | 29 | 97 | 46 | 124 | 56 | - | - | - | - | - | - | - | - | - | - |
| XC3S200 | 63 | 29 | 97 | 46 | 141 | 62 | 173 | 76 | - | - | - | - | - | - | - | - |
| XC3S400 | - | - | 97 | 46 | 141 | 62 | 173 | 76 | 264 | 116 | - | - | - | - | - | - |
| XC3S1000 | - | - | - | - | - | - | 173 | 76 | 333 | 149 | 391 | 175 | - | - | - | - |
| XC3S1500 | - | - | - | - | - | - | - | - | 333 | 149 | 487 | 221 | - | - | - | - |
| XC3S2000 | - | - | - | - | - | - | - | - | - | - | 489 | 221 | 565 | 270 | - | - |
| XC3S4000 | - | - | - | - | - | - | - | - | - | - | - | - | 633 | 300 | 712 | 312 |
| XC3S5000 | - | - | - | - | - | - | - | - | - | - | - | - | 633 | 300 | 784 | 344 |

**Notes:**
1. All device options listed in a given package column are pin-compatible.

## Product Ordering and Availability

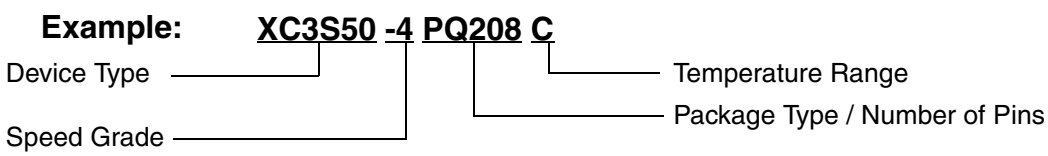
Table 4 shows all valid device ordering combinations of device density, speed grade, package, and temperature range parameters for the Spartan-3 family as well as the availability status of those combinations.

*Table 4:* **Spartan-3 Device Availability**

| Package Type: | VQFP | TQFP | PQFP | FTBGA | FBGA | | | |
|---|---|---|---|---|---|---|---|---|
| No. of Pins: | 100 | 144 | 208 | 256 | 456 | 676 | 900 | 1156 |
| Code: | VQ100 | TQ144 | PQ208 | FT256 | FG456 | FG676 | FG900 | FG1156 |
| Device(1) | | | | | | | | |
| XC3S50 | (C, I) | (C, I) | (C, I) | - | - | - | - | - |
| XC3S200 | (C, I) | (C, I) | (C, I) | (C, I) | - | - | - | - |
| XC3S400 | - | (C, I) | (C, I) | (C, I) | (C, I) | - | - | - |
| XC3S1000 | - | - | - | (C, I) | (C, I) | (C, I) | - | - |
| XC3S1500 | - | - | - | - | (C, I) | (C, I) | - | - |
| XC3S2000 | - | - | - | - | - | (C, I) | (C, I) | - |
| XC3S4000 | - | - | - | - | - | - | (C, I) | (C, I) |
| XC3S5000 | - | - | - | - | - | - | (C, I) | (C, I) |

**Notes:**
1. Commercial devices are offered in the -4 and -5 speed grades; industrial devices are only in the -4 speed grade.
2. C = Commercial, $T_J$ = 0° to +85°C; I = Industrial, $T_J$ = –40°C to +100°C.
3. Parentheses indicate that a given device is not yet released to production. Contact your local sales office for availability information.

## Ordering Information




| Device |
|---|
| XC3S50 |
| XC3S200 |
| XC3S400 |
| XC3S1000 |
| XC3S1500 |
| XC3S2000 |
| XC3S4000 |
| XC3S5000 |

| Speed Grade | |
|---|---|
| -4 | Standard Performance |
| -5 | High Performance |

| Package Type / Number of Pins | |
|---|---|
| VQ100 | 100-pin Very Thin Quad Flat Pack (VQFP) |
| TQ144 | 144-pin Thin Quad Flat Pack (TQFP) |
| PQ208 | 208-pin Plastic Quad Flat Pack (PQFP) |
| FT256 | 256-ball Fine-Pitch Thin Ball Grid Array (FTBGA) |
| FG456 | 456-ball Fine-Pitch Ball Grid Array (FBGA) |
| FG676 | 676-ball Fine-Pitch Ball Grid Array (FBGA) |
| FG900 | 900-ball Fine-Pitch Ball Grid Array (FBGA) |
| FG1156 | 1156-ball Fine-Pitch Ball Grid Array (FBGA) |

| Temperature Range ($T_J$) | |
|---|---|
| C | Commercial (0°C to 85°C) |
| I | Industrial (–40°C to 100°C) |

## Revision History

| Date | Version No. | Description |
|---|---|---|
| 04/11/03 | 1.0 | Initial Xilinx release. |
| 04/24/03 | 1.1 | Updated block RAM, DCM, and multiplier counts for the XC3S50. |

## The Spartan-3 Family Data Sheet

**DS099-1**, ***Spartan-3 1.2V FPGA Family: Introduction and Ordering Information*** (Module 1)

**DS099-2**, ***Spartan-3 1.2V FPGA Family: Functional Description*** (Module 2)

**DS099-3**, ***Spartan-3 1.2V FPGA Family: DC and Switching Characteristics*** (Module 3)

**DS099-4**, ***Spartan-3 1.2V FPGA Family: Pinout Tables*** (Module 4)

# IOBs

## IOB Overview

The Input/Output Block (IOB) provides a programmable, bidirectional interface between an I/O pin and the FPGA's internal logic.

A simplified diagram of the IOB's internal structure appears in Figure 1. There are three main signal paths within the IOB: the output path, input path, and 3-state path. Each path has its own pair of storage elements that can act as either registers or latches. For more information, see the Storage Element Functions section. The three main signal paths are as follows:

- The input path carries data from the pad, which is bonded to a package pin, through an optional programmable delay element directly to the I line. After the delay element, there are alternate routes through a pair of storage elements to the IQ1 and IQ2 lines. The IOB outputs I, IQ1, and IQ2 all lead to the FPGA's internal logic. The delay element can be set to ensure a hold time of zero.
- The output path, starting with the O1 and O2 lines, carries data from the FPGA's internal logic through a multiplexer and then a three-state driver to the IOB pad. In addition to this direct path, the multiplexer provides the option to insert a pair of storage elements.
- The 3-state path determines when the output driver is high impedance. The T1 and T2 lines carry data from the FPGA's internal logic through a multiplexer to the output driver. In addition to this direct path, the multiplexer provides the option to insert a pair of storage elements.
- All signal paths entering the IOB, including those associated with the storage elements, have an inverter option. Any inverter placed on these paths is automatically absorbed into the IOB.

## Storage Element Functions

There are three pairs of storage elements in each IOB, one pair for each of the three paths. It is possible to configure each of these storage elements as an edge-triggered D-type flip-flop (FD) or a level-sensitive latch (LD).

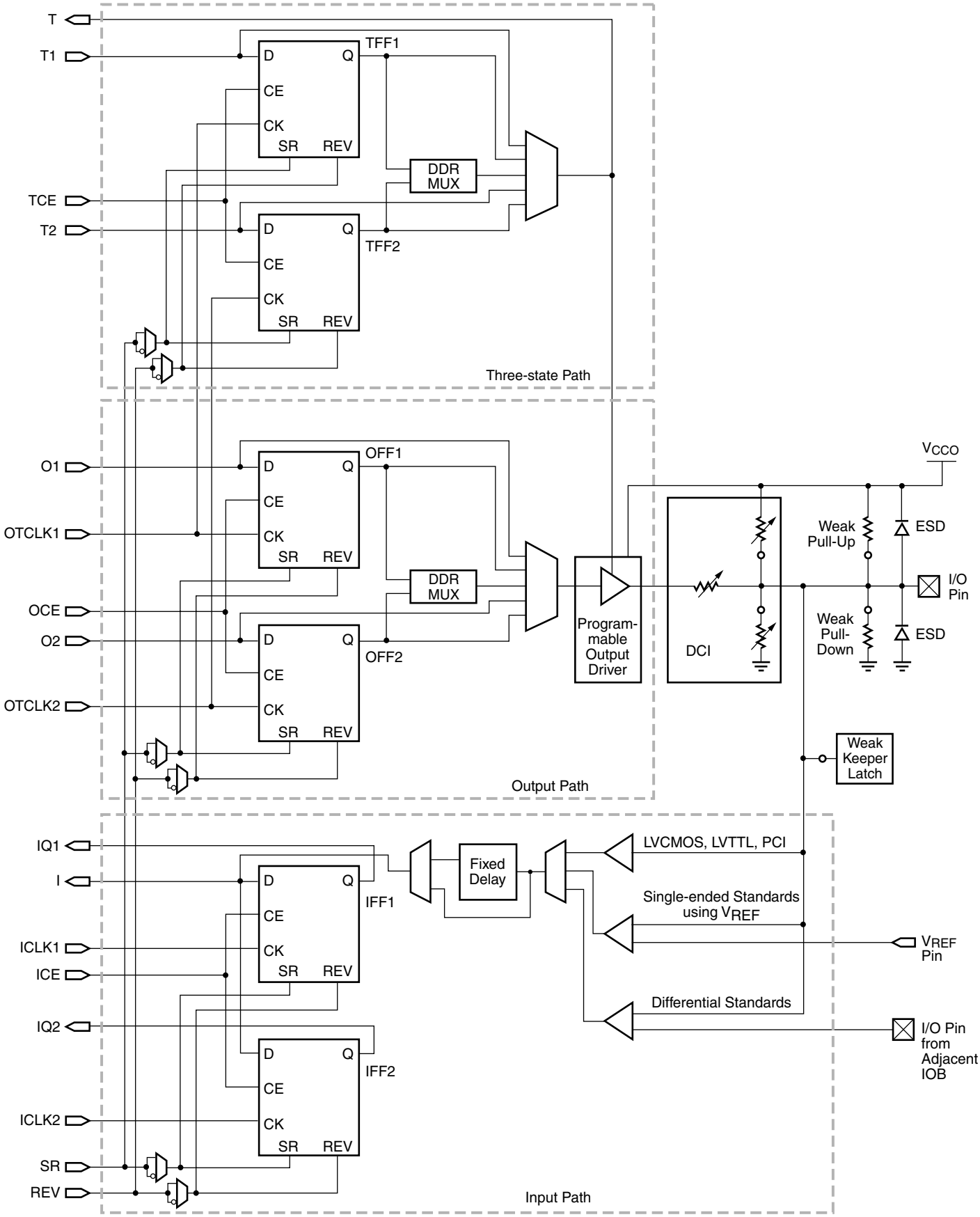
The storage-element-pair on either the Output path or the Three-State path can be used together with a special multiplexer to produce Double-Data-Rate (DDR) transmission. This is accomplished by taking data synchronized to the clock signal's rising edge and converting them to bits synchronized on both the rising and the falling edge. The combination of two registers and a multiplexer is referred to as a Double-Data-Rate D-type flip-flop (FDDR).

See **Double-Data-Rate Transmission**, page 19 for more information.

The signal paths associated with the storage element are described in Table 1.

*Table 1:* **Storage Element Signal Description**

| Storage Element Signal | Description | Function |
|---|---|---|
| D | Data input | Data at this input is stored on the active edge of CK enabled by CE. For latch operation when the input is enabled, data passes directly to the output Q. |
| Q | Data output | The data on this output reflects the state of the storage element. For operation as a latch in transparent mode, Q will mirror the data at D. |
| CK | Clock input | A signal's active edge on this input with CE asserted, loads data into the storage element. |
| CE | Clock Enable input | When asserted, this input enables CK. If not connected, CE defaults to the asserted state. |
| SR | Set/Reset | Forces storage element into the state specified by the SRHIGH/SRLOW attributes. The SYNC/ASYNC attribute setting determines if the SR input is synchronized to the clock or not. |
| REV | Reverse | Used together with SR. Forces storage element into the state opposite from what SR does. |




Note: All IOB signals communicating with the FPGA's internal logic have the option of inverting polarity.

DS099_01_040703

*Figure 1:* **Simplified IOB Diagram**

According to Figure 1, the clock line OTCLK1 connects the CK inputs of the upper registers on the output and three-state paths. Similarly, OTCLK2 connects the CK inputs for the lower registers on the output and three-state paths. The upper and lower registers on the input path have independent clock lines: ICLK1 and ICLK2.

The enable line OCE connects the CE inputs of the upper and lower registers on the output path. Similarly, TCE connects the CE inputs for the register pair on the three-state path and ICE does the same for the register pair on the input path.

The Set/Reset (SR) line entering the IOB is common to all six registers, as is the Reverse (REV) line.

Each storage element supports numerous options in addition to the control over signal polarity described in the IOB Overview section. These are described in Table 2.

*Table 2:* **Storage Element Options**

| Option Switch | Function | Specificity |
|---|---|---|
| FF/Latch | Chooses between an edge-sensitive flip-flop or a level-sensitive latch | Independent for each storage element. |
| SYNC/ASYNC | Determines whether SR is synchronous or asynchronous | Independent for each storage element. |
| SRHIGH/SRLOW | Determines whether SR acts as a Set, which forces the storage element to a logic “1" (SRHIGH) or a Reset, which forces a logic “0” (SRLOW). | Independent for each storage element, except when using FDDR. In the latter case, the selection for the upper element (OFF1 or TFF2) will apply to both elements. |
| INIT1/INIT0 | In the event of a Global Set/Reset, after configuration or upon activation of the GTS net, this switch decides whether to set or reset a storage element. By default, choosing SRLOW also selects INIT0; choosing SRHIGH also selects INIT1. | Independent for each storage element, except when using FDDR. In the latter case, selecting INIT0 for one element applies to both elements (even though INIT1 is selected for the other). |

## Double-Data-Rate Transmission

Double-Data-Rate (DDR) transmission describes the technique of synchronizing signals to both the rising and falling edges of the clock signal. Spartan-3 devices use register-pairs in all three IOB paths to perform DDR operations.

The pair of storage elements on the IOB’s Output path (OFF1 and OFF2), used as registers, combine with a special multiplexer to form a DDR D-type flip-flop (FDDR). This primitive permits DDR transmission where output data bits are synchronized to both the rising and falling edges of a clock. It is possible to access this function by placing either an FDDRRSE or an FDDRCPE component or symbol into the design. DDR operation requires two clock signals (50% duty cycle), one the inverted form of the other. These signals trigger the two registers in alternating fashion, as shown in Figure 2. Commonly, the Digital Clock Manager (DCM) generates the two clock signals by mirroring an incoming signal, then shifting it 180 degrees. This approach ensures minimal skew between the two signals.

The storage-element-pair on the Three-State path (TFF1 and TFF2) can also be combined with a local multiplexer to form an FDDR primitive. This permits synchronizing the output enable to both the rising and falling edges of a clock. This DDR operation is realized in the same way as for the output path.

The storage-element-pair on the input path (IFF1 and IFF2) allows an I/O to receive a DDR signal. An incoming DDR clock signal triggers one register and the inverted clock signal triggers the other register. In this way, the registers take turns capturing bits of the incoming DDR data signal.

Aside from high bandwidth data transfers, DDR can also be used to reproduce, or “mirror”, a clock signal on the output. This approach is used to transmit clock and data signals together. A similar approach is used to reproduce a clock signal at multiple outputs. The advantage for both approaches is that skew across the outputs will be minimal.

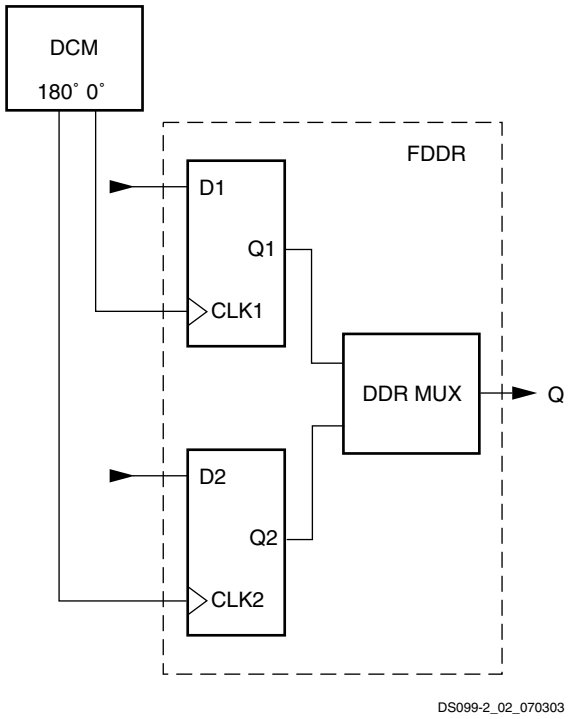



*Figure 2:* **Clocking the DDR Register**

## Pull-Up and Pull-Down Resistors

The optional pull-up and pull-down resistors are intended to establish High and Low levels, respectively, at unused I/Os. The weak pull-up resistor optionally connects each IOB pad to $V_{CCO}$. A weak pull-down resistor optionally connects each pad to GND. These resistors are placed in a design using the PULLUP and PULLDOWN symbols in a schematic, respectively. They can also be instantiated as components, set as constraints or passed as attributes in HDL code. These resistors can also be selected for all unused I/O using the Bitstream Generator (BitGen) option UnusedPin. A Low logic level on HSWAP_EN activates the pull-up resistors on all I/Os during configuration.

## Weak-Keeper Circuit

Each I/O has an optional weak-keeper circuit that retains the last logic level on a line after all drivers have been turned off. This is useful to keep bus lines from floating when all connected drivers are in a high-impedance state. This function is placed in a design using the KEEPER symbol. Pull-up and pull-down resistors override the weak-keeper circuit.

## ESD Protection

Clamp diodes protect all device pads against damage from Electro-Static Discharge (ESD) as well as excessive voltage transients. Each I/O has two clamp diodes: One diode extends P-to-N from the pad to $V_{CCO}$ and a second diode extends N-to-P from the pad to GND. During operation, these diodes are normally biased in the off state. These clamp diodes are always connected to the pad, regardless of the signal standard selected. The presence of diodes limits the ability of Spartan-3 I/Os to tolerate high signal voltages. The $V_{IN}$ absolute maximum rating in **Table 1, Module 3** specifies the voltage range that I/Os can tolerate.

## Slew Rate Control and Drive Strength

Two options, FAST and SLOW, control the output slew rate. The FAST option supports output switching at a high rate. The SLOW option reduces bus transients. These options are only available when using one of the LVCMOS or LVTTL standards, which also provide up to seven different levels of current drive strength: 2, 4, 6, 8, 12, 16, and 24 mA. Choosing the appropriate drive strength level is yet another means to minimize bus transients.

Table 3 shows the drive strengths that the LVCMOS and LVTTL standards support. The Fast option is indicated by appending an "F" attribute after the output buffer symbol OBUF or the bidirectional buffer symbol IOBUF. The Slow option appends an "S" attribute. The drive strength in milliamperes follows the slew rate attribute. For example, OBUF_LVCMOS18_S_6 or IOBUF_LVCMOS25_F_16.

*Table 3:* **Programmable Output Drive Current**

| Signal Standard | Current Drive (mA) | | | | | | |
|---|---|---|---|---|---|---|---|
| | 2 | 4 | 6 | 8 | 12 | 16 | 24 |
| LVCMOS12 | ✓ | ✓ | ✓ | - | - | - | - |
| LVCMOS15 | ✓ | ✓ | ✓ | ✓ | ✓ | - | - |
| LVCMOS18 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - |
| LVCMOS25 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| LVCMOS33 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| LVTTL | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

## Boundary-Scan Capability

All Spartan-3 IOBs support boundary-scan testing compatible with IEEE 1149.1 standards. See **Boundary-Scan (JTAG) Mode**, page 52 for more information.

## SelectIO Signal Standards

The IOBs support 17 different single-ended signal standards, as listed in Table 4. Furthermore, the majority of IOBs can be used in specific pairs supporting any of six differential signal standards, as shown in Table 5. The desired standard is selected by placing the appropriate I/O library symbol or component into the FPGA design. For example, the symbol named IOBUF_LVCMOS15_F_8 represents a bidirectional I/O to which the 1.5V LVCMOS signal standard has been assigned. The slew rate and current drive are set to Fast and 8 mA, respectively.

Together with placing the appropriate I/O symbol, two externally applied voltage levels, $V_{CCO}$ and $V_{REF}$ select the desired signal standard. The $V_{CCO}$ lines provide current to the output driver. The voltage on these lines determines the

output voltage swing for all standards except GTL and GTLP.

All single-ended standards except the LVCMOS modes require a Reference Voltage ($V_{REF}$) to bias the input-switching threshold. Once a configuration data file is loaded into the FPGA that calls for the I/Os of a given bank to use such a signal standard, a few specifically reserved I/O pins on the same bank automatically convert to $V_{REF}$ inputs. When using one of the LVCMOS standards, these pins remain I/Os because the $V_{CCO}$ voltage biases the input-switching threshold, so there is no need for $V_{REF}$. Select the $V_{CCO}$ and $V_{REF}$ levels to suit the desired single-ended standard according to Table 4.

Differential standards employ a pair of signals, one the opposite polarity of the other. The noise canceling (e.g., Common-Mode Rejection) properties of these standards permit exceptionally high data transfer rates. This section introduces the differential signaling capabilities of Spartan-3 devices.

Each device-package combination designates specific I/O pairs that are specially optimized to support differential standards. A unique “L-number”, part of the pin name, identifies the line-pairs associated with each bank (see **Module 4**). For each pair, the letters “P” and “N” designate the true and inverted lines, respectively. For example, the pin names IO_L43P_7 and IO_L43N_7 indicate the true and inverted lines comprising the line pair L43 on Bank 7. The differential Output Voltage ($V_{OD}$) parameter measures the voltage difference the High and Low logic levels that a pair of differential outputs drive. The $V_{OD}$ range for each of the differential standards is listed in Table 5. The $V_{CCO}$ lines provide current to the outputs. The $V_{REF}$ lines are not used. Select the $V_{CCO}$ level to suit the desired differential standard according to Table 5.

*Table 4:* **Single-Ended I/O Standards (Values in Volts)**

| Signal Standard | $V_{CCO}$ For Outputs | $V_{CCO}$ For Inputs | $V_{REF}$ for Inputs[1] | Board Termination Voltage ($V_{TT}$) |
|---|---|---|---|---|
| GTL | Note 2 | Note 2 | 0.8 | 1.2 |
| GTLP | Note 2 | Note 2 | 1 | 1.5 |
| HSTL_I | 1.5 | - | 0.75 | 0.75 |
| HSTL_III | 1.5 | - | 0.9 | 1.5 |
| HSTL_I_18 | 1.8 | - | 0.9 | 0.9 |
| HSTL_II_18 | 1.8 | - | 0.9 | 0.9 |
| HSTL_III_18 | 1.8 | - | 1.1 | 1.8 |
| LVCMOS12 | 1.2 | 1.2 | - | - |
| LVCMOS15 | 1.5 | 1.5 | - | - |
| LVCMOS18 | 1.8 | 1.8 | - | - |
| LVCMOS25 | 2.5 | 2.5 | - | - |
| LVCMOS33 | 3.3 | 3.3 | - | - |
| LVTTL | 3.3 | 3.3 | - | - |

*Table 4:* **Single-Ended I/O Standards (Values in Volts)**

| Signal Standard | $V_{CCO}$ For Outputs | $V_{CCO}$ For Inputs | $V_{REF}$ for Inputs[1] | Board Termination Voltage ($V_{TT}$) |
|---|---|---|---|---|
| PCI33_3 | 3.0 | 3.0 | - | - |
| SSTL18_I | 1.8 | - | 0.9 | 0.9 |
| SSTL2_I | 2.5 | - | 1.25 | 1.25 |
| SSTL2_II | 2.5 | - | 1.25 | 1.25 |

**Notes:**
1. Banks 4 and 5 of any Spartan-3 device in a VQ100 package do not support signal standards using $V_{REF}$.
2. The $V_{CCO}$ level used for the GTL and GTLP standards must be no lower than the termination voltage ($V_{TT}$), nor can it be lower than the voltage at the I/O pad.
3. See Table 6 for a listing of the single-ended DCI standards.

*Table 5:* **Differential I/O Standards**

| Signal Standard | $V_{CCO}$ (Volts) For Outputs | $V_{CCO}$ (Volts) For Inputs | $V_{REF}$ for Inputs (Volts) | $V_{OD}$[1] (mV) Min. | $V_{OD}$[1] (mV) Max. |
|---|---|---|---|---|---|
| LDT_25 | 2.5 | - | - | 430 | 670 |
| LVDS_25 | 2.5 | - | - | 250 | 400 |
| BLVDS_25 | 2.5 | - | - | 250 | 450 |
| LVDSEXT_25 | 2.5 | - | - | 330 | 700 |
| ULVDS_25 | 2.5 | - | - | 430 | 670 |
| RSDS_25 | 2.5 | - | - | 100 | 400 |

**Notes:**
1. Measured with a termination resistor value (RT) of 100 Ohms.
2. See Table 6 for a listing of the differential DCI standards.

The need to supply $V_{REF}$ and $V_{CCO}$ imposes constraints on which standards can be used in the same bank. See **The Organization of IOBs into Banks** section for additional guidelines concerning the use of the $V_{CCO}$ and $V_{REF}$ lines.

## Digitally Controlled Impedance (DCI)

When the round-trip delay of an output signal — i.e., from output to input and back again — exceeds rise and fall times, it is common practice to add termination resistors to the line carrying the signal. These resistors effectively match the impedance of a device’s I/O to the characteristic impedance of the transmission line, thereby preventing reflections that adversely affect signal integrity. However, with the high I/O counts supported by modern devices, adding resistors requires significantly more components and board area. Furthermore, for some packages — e.g., ball grid arrays — it may not always be possible to place resistors close to pins.

DCI answers these concerns by providing two kinds of on-chip terminations: Parallel terminations make use of an integrated resistor network. Series terminations result from controlling the impedance of output drivers. DCI actively adjusts both parallel and series terminations to accurately

match the characteristic impedance of the transmission line. This adjustment process compensates for differences in I/O impedance that can result from normal variation in the ambient temperature, the supply voltage and the manufacturing process. When the output driver turns off, the series termination, by definition, approaches a very high impedance; in contrast, parallel termination resistors remain at the targeted values.

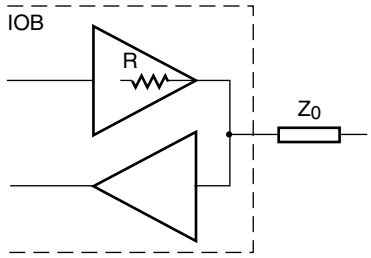
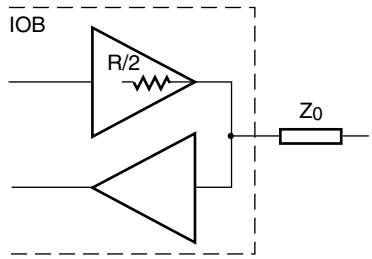
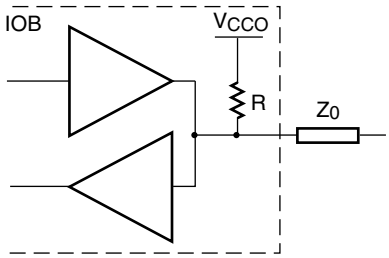
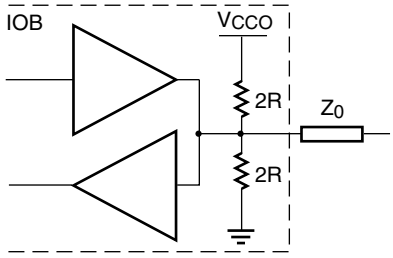
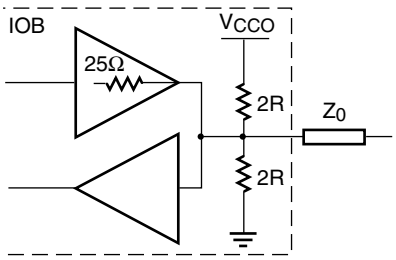
DCI is available only for certain I/O standards, as listed in Table 6. DCI is selected by applying the appropriate I/O standard extensions to symbols or components. There are five basic ways to configure terminations, as shown in Table 7. The DCI I/O standard determines which of these terminations is put into effect.

*Table 6:* **DCI I/O Standards**

| Category of Signal Standard | Signal Standard | $V_{CCO}$ (V) | | $V_{REF}$ for Inputs (V) | Termination Type | |
|---|---|---|---|---|---|---|
| | | For Outputs | For Inputs | | At Output | At Input |
| **Single-Ended** | | | | | | |
| Gunning Transceiver Logic | GTL_DCI | 1.2 | 1.2 | 0.8 | Single | Single |
| | GTLP_DCI | 1.5 | 1.5 | 1.0 | | |
| High-Speed Transceiver Logic | HSTL_I_DCI | 1.5 | 1.5 | 0.75 | None | Split |
| | HSTL_III_DCI | 1.5 | 1.5 | 0.9 | None | Single |
| | HSTL_I_DCI_18 | 1.8 | 1.8 | 0.9 | None | Split |
| | HSTL_II_DCI_18 | 1.8 | 1.8 | 0.9 | Split | |
| | HSTL_III_DCI_18 | 1.8 | 1.8 | 1.1 | None | Single |
| Low-Voltage CMOS | LVDCI_15 | 1.5 | 1.5 | - | Controlled impedance driver | None |
| | LVDCI_18 | 1.8 | 1.8 | - | | |
| | LVDCI_25 | 2.5 | 2.5 | - | | |
| | LVDCI_33 | 3.3 | 3.3 | - | | |
| | LVDCI_DV2_15 | 1.5 | 1.5 | - | Controlled driver with half-impedance | |
| | LVDCI_DV2_18 | 1.8 | 1.8 | - | | |
| | LVDCI_DV2_25 | 2.5 | 2.5 | - | | |
| | LVDCI_DV2_33 | 3.3 | 3.3 | - | | |
| Stub Series Terminated Logic | SSTL18_I_DCI | 1.8 | 1.8 | 0.9 | 25-Ohm driver | Split |
| | SSTL2_I_DCI | 2.5 | 2.5 | 1.25 | 25-Ohm driver | |
| | SSTL2_II_DCI | 2.5 | 2.5 | 1.25 | Split with 25-Ohm driver | |
| **Differential** | | | | | | |
| Low-Voltage Differential Signalling | LVDS_25_DCI | 2.5 | 2.5 | - | None | Split on each line of pair |
| | LVDSEXT_25_DCI | 2.5 | 2.5 | - | | |

**Notes:**
1. Bank 5 of any Spartan-3 device in a VQ100 or TQ144 package does not support DCI signal standards.

*Table 7:* **DCI Terminations**

| Termination | Schematic[1] | I/O Standards |
|---|---|---|
| Controlled impedance output driver |  | LVDCI_15<br>LVDCI_18<br>LVDCI_25<br>LVDCI_33 |
| Controlled output driver with half impedance |  | LVDCI_DV2_15<br>LVDCI_DV2_18<br>LVDCI_DV2_25<br>LVDCI_DV2_33 |
| Single resistor |  | GTL_DCI<br>GTLP_DCI<br>HSTL_III_DCI[2]<br>HSTL_III_DCI_18[2] |
| Split resistors |  | HSTL_I_DCI[2]<br>HSTL_I_DCI_18[2]<br>HSTL_II_DCI_18<br>LVDS_25_DCI<br>LVDSEXT_25_DCI |
| Split resistors with output driver impedance fixed to 25Ω |  | SSTL18_I_DCI[3]<br>SSTL2_I_DCI[3]<br>SSTL2_II_DCI |

**Notes:**
1. The value of R is equivalent to the characteristic impedance of the line connected to the I/O. It is also equal to half the value of $R_{REF}$ for the DV2 standards and $R_{REF}$ for all other DCI standards.
2. For DCI using HSTL Classes I and III, terminations only go into effect at inputs (not at outputs).
3. For DCI using SSTL Class I, the split termination only goes into effect at inputs (not at outputs).

The DCI feature operates independently for each of the device's eight banks. Each bank has an "N" reference pin (VRN) and a "P" reference pin, (VRP), to calibrate driver and termination resistance. Only when using a DCI standard on a given bank do these two pins function as VRN and VRP. When not using a DCI standard, the two pins function as user I/Os. As shown in Figure 3, add an external reference resistor to pull the VRN pin up to $V_{CCO}$ and another reference resistor to pull the VRP pin down to GND. Both resistors have the same value — commonly 50 Ohms — with one-percent tolerance, which is either the characteristic impedance of the line or twice that, depending on the DCI standard in use. Standards having a symbol name that contains the letters "DV2" use a reference resistor value that is twice the line impedance. DCI adjusts the output driver impedance to match the reference resistors' value or half that, according to the standard. DCI always adjusts the on-chip termination resistors to directly match the reference resistors' value.

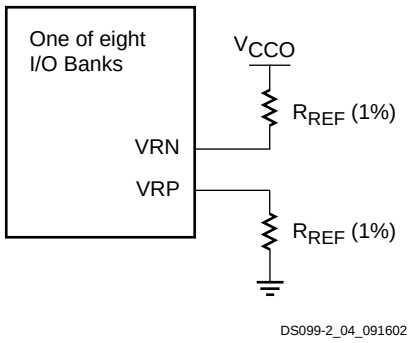



*Figure 3:* **Connection of Reference Resistors ($R_{REF}$)**

The rules guiding the use of DCI standards on banks are as follows:

1. No more than one DCI I/O standard with a Single Termination is allowed per bank.
2. No more than one DCI I/O standard with a Split Termination is allowed per bank.
3. Single Termination, Split Termination, Controlled-Impedance Driver, and Controlled-Impedance Driver with Half Impedance can co-exist in the same bank.

See also **The Organization of IOBs into Banks**, page 24.

## The Organization of IOBs into Banks

IOBs are allocated among eight banks, so that each side of the device has two banks, as shown in Figure 4. For all packages, each bank has independent $V_{REF}$ lines. For example, $V_{REF}$ Bank 3 lines are separate from the $V_{REF}$ lines going to all other banks.

For the Very Thin Quad Flat Pack (VQ), Plastic Quad Flat Pack (PQ), Fine Pitch Thin Ball Grid Array (FT), and Fine Pitch Ball Grid Array (FG) packages, each bank has dedicated $V_{CCO}$ lines. For example, the $V_{CCO}$ Bank 7 lines are separate from the $V_{CCO}$ lines going to all other banks. Thus, Spartan-3 devices in these packages support eight independent $V_{CCO}$ supplies.

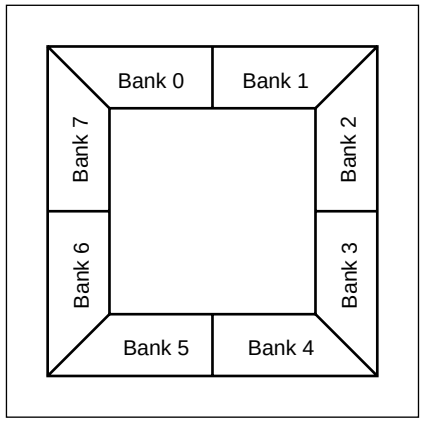



*Figure 4:* **Spartan-3 I/O Banks (top view)**

In contrast, the 144-pin Thin Quad Flat Pack (TQ144) package ties $V_{CCO}$ together internally for the pair of banks on each side of the device. For example, the $V_{CCO}$ Bank 0 and the $V_{CCO}$ Bank 1 lines are tied together. The interconnected bank-pairs are 0/1, 2/3, 4/5, and 6/7. As a result, Spartan-3 devices in the TQ144 package support four independent $V_{CCO}$ supplies.

## Spartan-3 Compatibility

Within the Spartan-3 family, all devices are pin-compatible by package. When the need for future logic resources outgrows the capacity of the Spartan-3 device in current use, a larger device in the same package can serve as a direct replacement. Larger devices may add extra $V_{REF}$ and $V_{CCO}$ lines to support a greater number of I/Os. In the larger device, more pins can convert from user I/Os to $V_{REF}$ lines. Also, additional $V_{CCO}$ lines are bonded out to pins that were "not connected" in the smaller device. Thus, it is important to plan for future upgrades at the time of the board's initial design by laying out connections to the extra pins.

The Spartan-3 family is not pin-compatible with any previous Xilinx FPGA family.

## Rules Concerning Banks

When assigning I/Os to banks, it is important to follow the following $V_{CCO}$ rules:

1. Leave no $V_{CCO}$ pins unconnected on the FPGA.
2. Set all $V_{CCO}$ lines associated with the (interconnected) bank to the same voltage level.
3. The $V_{CCO}$ levels used by all standards assigned to the I/Os of the (interconnected) bank(s) must agree. The Xilinx development software checks for this. Tables 4, 5, and 6 describe how different standards use the $V_{CCO}$ supply.

4. If none of the standards assigned to the I/Os of the (interconnected) bank(s) use $V_{CCO}$, tie all associated $V_{CCO}$ lines to 2.5V.
5. In general, apply 2.5V to $V_{CCO}$ Bank 4 from power-on to the end of configuration. Apply the same voltage to $V_{CCO}$ Bank 5 during parallel configuration or a Readback operation. For information on how to program the FPGA using 3.3V signals and power, see the **3.3V-Tolerant Configuration Interface** section.

If any of the standards assigned to the Inputs of the bank use $V_{REF}$, then observe the following additional rules:

1. Leave no $V_{REF}$ pins unconnected on any bank.
2. Set all $V_{REF}$ lines associated with the bank to the same voltage level.
3. The $V_{REF}$ levels used by all standards assigned to the Inputs of the bank must agree. The Xilinx development software checks for this. Tables 4 and 6 describe how different standards use the $V_{REF}$ supply.

If none of the standards assigned to the Inputs of a bank use $V_{REF}$ for biasing input switching thresholds, all associated $V_{REF}$ pins function as User I/Os.

## Exceptions to Banks Supporting I/O Standards

Bank 5 of any Spartan-3 device in a VQ100 or TQ144 package does not support DCI signal standards. In this case, bank 5 has neither VRN nor VRP pins.

Furthermore, banks 4 and 5 of any Spartan-3 device in a VQ100 package do not support signal standards using $V_{REF}$ (see Table 4). In this case, the two banks do not have any $V_{REF}$ pins.

## Supply Voltages for the IOBs

Three different supplies power the IOBs:

1. The $V_{CCO}$ supplies, one for each of the FPGA's I/O banks, power the output drivers, except when using the GTL and GTLP signal standards. The voltage on the $V_{CCO}$ pins determines the voltage swing of the output signal.
2. $V_{CCINT}$ is the main power supply for the FPGA's internal logic.
3. The $V_{CCAUX}$ is an auxiliary source of power, primarily to optimize the performance of various FPGA functions such as I/O switching.

## The I/Os During Power-On, Configuration, and User Mode

With no power applied to the FPGA, all I/Os are in a high-impedance state. The $V_{CCINT}$ (1.2V), $V_{CCAUX}$ (2.5V), and $V_{CCO}$ supplies may be applied in any order. Before power-on can finish, $V_{CCINT}$, $V_{CCO}$ Bank 4, and $V_{CCAUX}$ must have reached their respective minimum recommended operating levels (see **Table 2 in Module 3**). At this time, all I/O drivers also will be in a high-impedance state. $V_{CCO}$ Bank 4, $V_{CCINT}$, and $V_{CCAUX}$ serve as inputs to the internal Power-On Reset circuit (POR).

A Low level applied to HSWAP_EN input enables weak pull-up resistors on User I/Os from power-on throughout configuration. A High level on HSWAP_EN disables the pull-up resistors, allowing the I/Os to float. As soon as power is applied, the FPGA begins initializing its configuration memory. At the same time, the FPGA internally asserts the Global Set-Reset (GSR), which asynchronously resets all IOB storage elements to a Low state.

Upon the completion of initialization, INIT_B goes High, sampling the M0, M1, and M2 inputs to determine the configuration mode. At this point, the configuration data is loaded into the FPGA. The I/O drivers remain in a high-impedance state (with or without pull-up resistors, as determined by the HSWAP_EN input) throughout configuration.

The Global Three State (GTS) net is released during Start-Up, marking the end of configuration and the beginning of design operation in the User mode. At this point, those I/Os to which signals have been assigned go active while all unused I/Os remain in a high-impedance state. The release of the GSR net, also part of Start-up, leaves the IOB registers in a Low state by default, unless the loaded design reverses the polarity of their respective RS inputs.

In User mode, all weak, internal pull-up resistors on the I/Os are disabled and HSWAP_EN becomes a "don't care" input. If it is desirable to have weak pull-up or pull-down resistors on I/Os carrying signals, the appropriate symbol — e.g., PULLUP, PULLDOWN — must be placed at the appropriate pads in the design. The Bitstream Generator (Bitgen) option UnusedPin available in the Xilinx development software determines whether unused I/Os collectively have pull-up resistors, pull-down resistors, or no resistors in User mode.

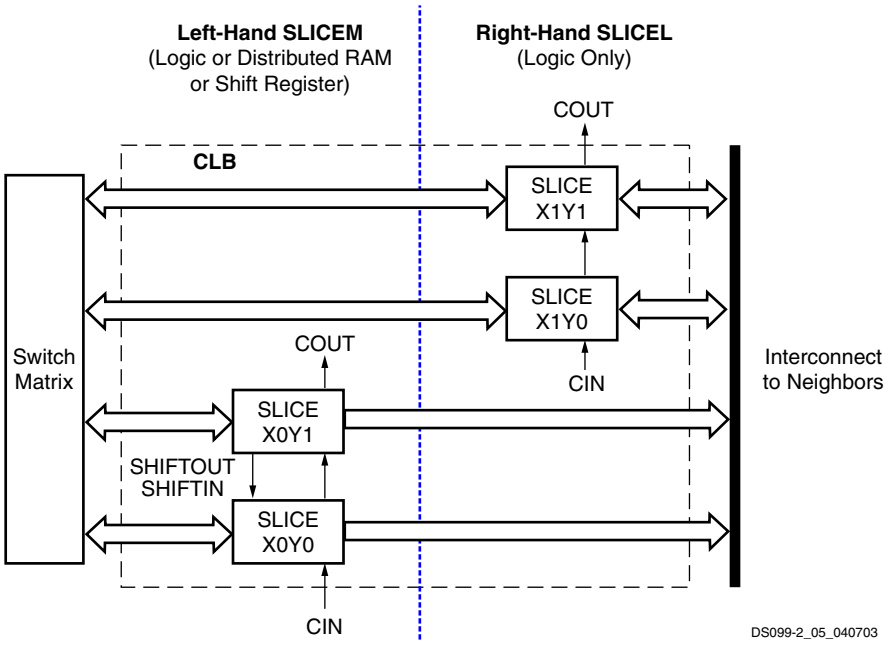



*Figure 5:* **Arrangement of Slices within the CLB**

## CLB Overview

The Configurable Logic Blocks (CLBs) constitute the main logic resource for implementing synchronous as well as combinatorial circuits. Each CLB comprises four interconnected slices, as shown in Figure 5. These slices are grouped in pairs. Each pair is organized as a column with an independent carry chain.

The nomenclature that the FPGA Editor — part of the Xilinx development software — uses to designate slices is as follows: The letter "X" followed by a number identifies columns of slices. The "X" number counts up in sequence from the left side of the die to the right. The letter "Y" followed by a number identifies the position of each slice in a pair as well as indicating the CLB row. The "Y" number counts slices starting from the bottom of the die according to the sequence: 0, 1, 0, 1 (the first CLB row); 2, 3, 2, 3 (the second CLB row); etc. Figure 5 shows the CLB located in the lower left-hand corner of the die. Slices X0Y0 and X0Y1 make up the column-pair on the left where as slices X1Y0 and X1Y1 make up the column-pair on the right. For each CLB, the term “left-hand” (or SLICEM) is used to indicated the pair of slices labeled with an even "X" number, such as X0, and the term “right-hand” (or SLICEL) designates the pair of slices with an odd "X" number, e.g., X1.

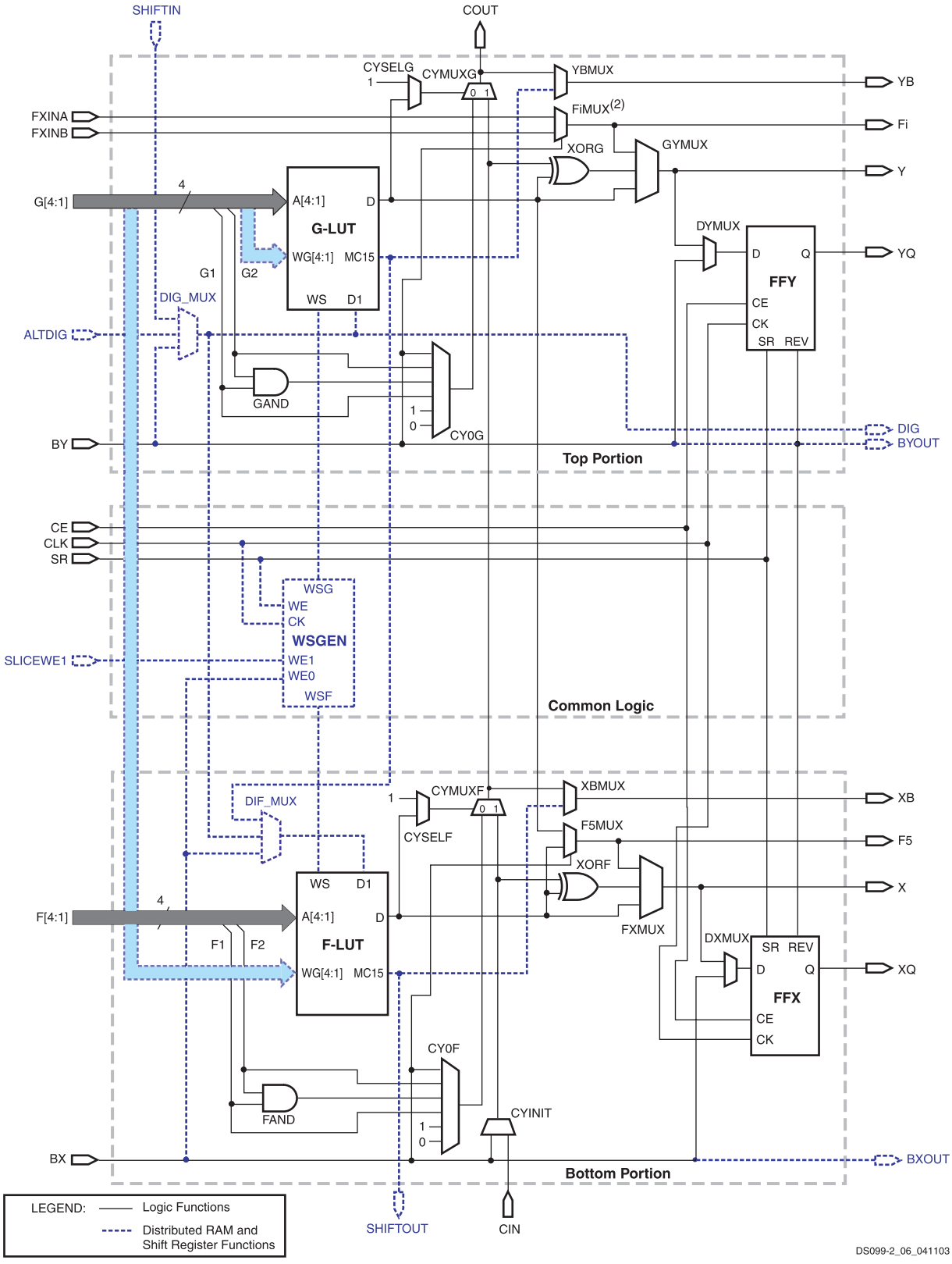
### Elements Within a Slice

All four slices have the following elements in common: two logic function generators, two storage elements, wide-function multiplexers, carry logic, and arithmetic gates, as shown in Figure 6. Both the left-hand and right-hand slice pairs use these elements to provide logic, arithmetic, and ROM functions. Besides these, the left-hand pair supports two additional functions: storing data using Distributed RAM and shifting data with 16-bit registers. Figure 6 is a diagram of the left-hand slice; therefore, it represents a superset of the elements and connections to be found in all slices. See **Function Generator**, page 28 for more information.

The RAM-based function generator — also known as a Look-Up Table or LUT — is the main resource for implementing logic functions. Furthermore, the LUTs in each left-hand slice pair can be configured as Distributed RAM or a 16-bit shift register. For information on the former, see **XAPP464**: *Using Look-Up Tables as Distributed RAM in Spartan-3 FPGAs*; for information on the latter, refer to **XAPP465**: *Using Look-Up Tables as Shift Registers (SRL16) in Spartan-3 FPGAs*. The function generators located in the upper and lower portions of the slice are referred to as the "G" and "F", respectively.

The storage element, which is programmable as either a D-type flip-flop or a level-sensitive latch, provides a means for synchronizing data to a clock signal, among other uses. The storage elements in the upper and lower portions of the slice are called FFY and FFX, respectively.

Wide-function multiplexers effectively combine LUTs in order to permit more complex logic operations. Each slice has two of these multiplexers with F5MUX in the lower portion of the slice and FXMUX in the upper portion. Depending on the slice, FXMUX takes on the name F6MUX, F7MUX, or F8MUX. For more details on the multiplexers, see **XAPP466**: *Using Dedicated Multiplexers in Spartan-3 FPGAs*.




**Notes:**

1. Options to invert signal polarity as well as other options that enable lines for various functions are not shown.
2. The index i can be 6, 7, or 8, depending on the slice. In this position, the upper right-hand slice has an F8MUX, and the upper left-hand slice has an F7MUX. The lower right-hand and left-hand slices both have an F6MUX.

*Figure 6:* **Simplified Diagram of the Left-Hand SLICEM**

The carry chain, together with various dedicated arithmetic logic gates, support fast and efficient implementations of math operations. The carry chain enters the slice as CIN and exits as COUT. Five multiplexers control the chain: CYINIT, CY0F, and CYMUXF in the lower portion as well as CY0G and CYMUXG in the upper portion. The dedicated arithmetic logic includes the exclusive-OR gates XORF and XORG (upper and lower portions of the slice, respectively) as well as the AND gates GAND and FAND (upper and lower portions, respectively).

## Main Logic Paths

Central to the operation of each slice are two nearly identical data paths, distinguished using the terms *top* and *bottom*. The description that follows uses names associated with the bottom path. (The top path names appear in parentheses.) The basic path originates at an interconnect-switch matrix outside the CLB. Four lines, F1 through F4 (or G1 through G4 on the upper path), enter the slice and connect directly to the LUT. Once inside the slice, the lower 4-bit path passes through a function generator "F" (or "G") that performs logic operations. The function generator's Data output, "D", offers five possible paths:

1. Exit the slice via line "X" (or "Y") and return to interconnect.
2. Inside the slice, "X" (or "Y") serves as an input to the DXMUX (DYMUX) which feeds the data input, "D", of the FFY (FFX) storage element. The "Q" output of the storage element drives the line XQ (or YQ) which exits the slice.
3. Control the CYMUXF (or CYMUXG) multiplexer on the carry chain.
4. With the carry chain, serve as an input to the XORF (or XORG) exclusive-OR gate that performs arithmetic operations, producing a result on "X" (or "Y").
5. Drive the multiplexer F5MUX to implement logic functions wider than four bits. The "D" outputs of both the F-LUT and G-LUT serve as data inputs to this multiplexer.

In addition to the main logic paths described above, there are two bypass paths that enter the slice as BX and BY. Once inside the FPGA, BX in the bottom half of the slice (or BY in the top half) can take any of several possible branches:

1. Bypass both the LUT and the storage element, then exit the slice as BXOUT (or BYOUT) and return to interconnect.
2. Bypass the LUT, then pass through a storage element via the D input before exiting as XQ (or YQ).
3. Control the wide function multiplexer F5MUX (or F6MUX).
4. Via multiplexers, serve as an input to the carry chain.
5. Drives the DI input of the LUT. See Distributed RAM section.
6. BY can control the REV inputs of both the FFY and FFX storage elements. See Storage Element Section.
7. Finally, the DIG_MUX multiplexer can switch BY onto to the DIG line, which exits the slice.

Other slice signals shown in Figure 6, page 27 are discussed in the sections that follow.

## Function Generator

Each of the two LUTs (F and G) in a slice have four logic inputs (A1-A4) and a single output (D). This permits any four-variable Boolean logic operation to be programmed into them. Furthermore, wide function multiplexers can be used to effectively combine LUTs within the same CLB or across different CLBs, making logic functions with still more input variables possible.

The LUTs in both the right-hand and left-hand slice-pairs not only support the logic functions described above, but also can function as ROM that is initialized with data at the time of configuration.

The LUTs in the left-hand slice-pair (even-numbered columns such as X0 in Figure 5) of each CLB support two additional functions that the right-hand slice-pair (odd-numbered columns such as X1) do not.

First, it is possible to program the "left-hand LUTs" as distributed RAM. This type of memory affords moderate amounts of data buffering anywhere along a data path. One left-hand LUT stores 16 bits. Multiple left-hand LUTs can be combined in various ways to store larger amounts of data. A dual port option combines two LUTs so that memory access is possible from two independent data lines. A Distributed ROM option permits pre-loading the memory with data during FPGA configuration For more information, see the Distributed RAM section.

Second, it is possible to program each left-hand LUT as a 16-bit shift register. Used in this way, each LUT can delay serial data anywhere from one to 16 clock cycles. The four left-hand LUTs of a single CLB can be combined to produce delays up to 64 clock cycles. The SHIFTIN and SHIFTOUT lines cascade LUTs to form larger shift registers. It is also possible to combine shift registers across more than one CLB. The resulting programmable delays can be used to balance the timing of data pipelines.

# Block RAM Overview

All Spartan-3 devices support block RAM, which is organized as configurable, synchronous 18Kbit blocks. Block RAM stores relatively large amounts of data more efficiently than the distributed RAM feature described earlier. (The latter is better suited for buffering small amounts of data anywhere along signal paths.) This section describes basic Block RAM functions. For more information, see **XAPP463**: *Using Block RAM in Spartan-3 FPGAs*.

The aspect ratio — i.e., width vs. depth — of each block RAM is configurable. Furthermore, multiple blocks can be cascaded to create still wider and/or deeper memories.

A choice among primitives determines whether the block RAM functions as dual- or single-port memory. A name of the form RAM16_S[$w_A$]_S[$w_B$] calls out the dual-port primitive, where the integers $w_A$ and $w_B$ specify the total data path width at ports $w_A$ and $w_B$, respectively. Thus, a RAM16_S9_S18 is a dual-port RAM with a 9-bit-wide Port A and an 18-bit-wide Port B. A name of the form RAM16_S[w] identifies the single-port primitive, where the integer w specifies the total data path width of the lone port. A RAM16_S18 is a single-port RAM with an 18-bit-wide port. Other memory functions — e.g., FIFOs, data path width conversion, ROM, etc. — are readily available using the CORE Generator™ system, part of the Xilinx development software.

## Arrangement of RAM Blocks on Die

The XC3S50 has one column of block RAM. The Spartan-3 devices ranging from the XC3S200 to XC3S2000 have two columns of block RAM. The XC3S4000 and XC3S5000 have four columns. The position of the columns on the die is shown in **Figure 1 in Module 1**. For a given device, the total available RAM blocks are distributed equally among the columns. Table 8 shows the number of RAM blocks, the data storage capacity, and the number of columns for each device.

*Table 8:* **Number of RAM Blocks by Device**

| Device | Total Number of RAM Blocks | Total Addressable Locations (bits) | Number of Columns |
|---|---|---|---|
| XC3S50 | 4 | 73,728 | 1 |
| XC3S200 | 12 | 221,184 | 2 |
| XC3S400 | 16 | 294,912 | 2 |
| XC3S1000 | 24 | 442,368 | 2 |
| XC3S1500 | 32 | 589,824 | 2 |
| XC3S2000 | 40 | 737,280 | 2 |
| XC3S4000 | 96 | 1,769,472 | 4 |
| XC3S5000 | 104 | 1,916,928 | 4 |

## The Internal Structure of the Block RAM

The block RAM has a dual port structure. The two identical data ports called A and B permit independent access to the common RAM block, which has a maximum capacity of 18,432 bits — or 16,384 bits when no parity lines are used. Each port has its own dedicated set of data, control and clock lines for synchronous read and write operations. There are four basic data paths, as shown in Figure 7: (1) write to and read from Port A, (2) write to and read from Port B, (3) data transfer from Port A to Port B, and (4) data transfer from Port B to Port A.

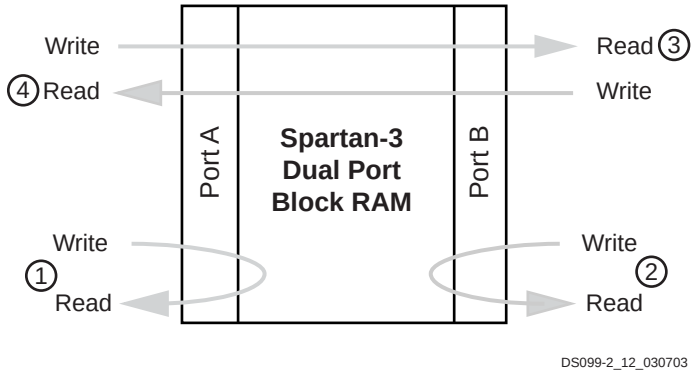
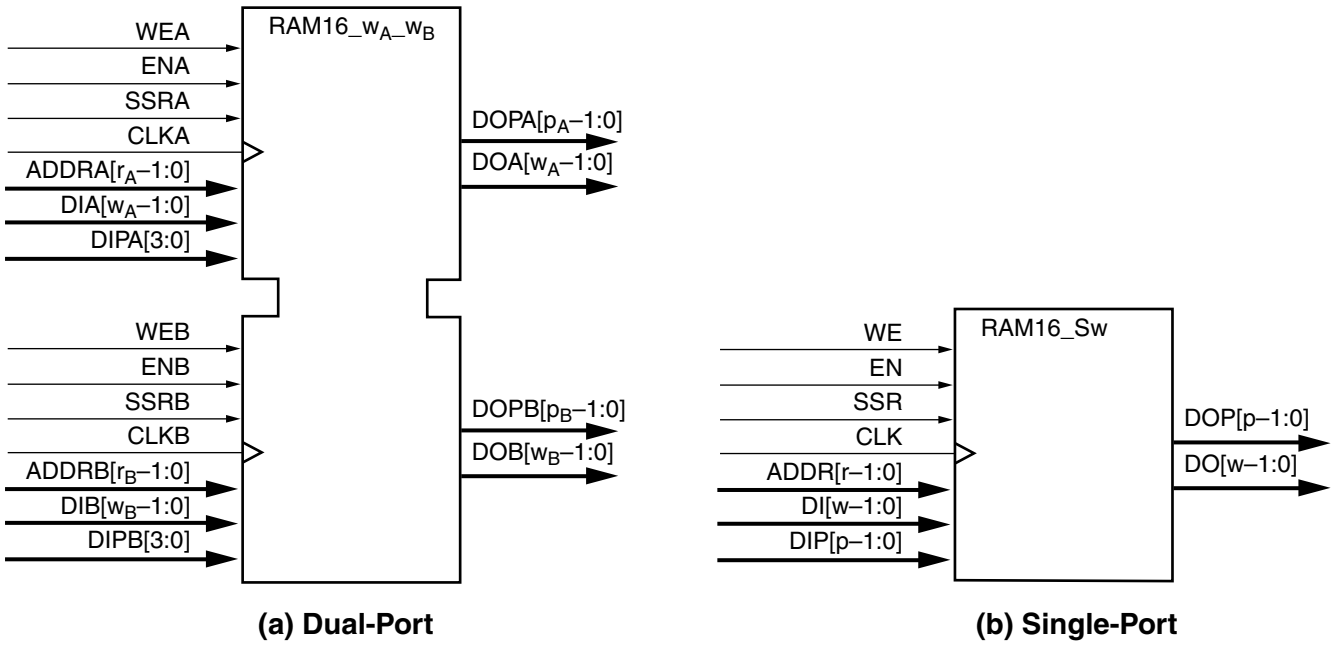



*Figure 7:* **Block RAM Data Paths**

## Block RAM Port Signal Definitions

Representations of the dual-port primitive RAM16_S[$w_A$]_S[$w_B$] and the single-port primitive RAM16_S[w] with their associated signals are shown in Figure 8a and Figure 8b, respectively. These signals are defined in Table 9.




**Notes:**
1. $w_A$ and $w_B$ are integers representing the total data path width (i.e., data bits plus parity bits) at ports A and B, respectively.
2. $p_A$ and $p_B$ are integers that indicate the number of data path lines serving as parity bits.
3. $r_A$ and $r_B$ are integers representing the address bus width at ports A and B, respectively.
4. The control signals CLK, WE, EN, and SSR on both ports have the option of inverted polarity.

*Figure 8:* **Block RAM Primitives**

*Table 9:* **Block RAM Port Signals**

| Signal Description | Port A Signal Name | Port B Signal Name | Direction | Function |
|---|---|---|---|---|
| Address Bus | ADDRA | ADDRB | Input | The Address Bus selects a memory location for read or write operations. The width (w) of the port's associated data path determines the number of available address lines (r). |
| Data Input Bus | DIA | DIB | Input | Data at the DI input bus is written to the addressed memory location addressed on an enabled active CLK edge.<br>It is possible to configure a port's total data path width (w) to be 1, 2, 4, 9, 18, or 36 bits. This selection applies to both the DI and DO paths of a given port. Each port is independent. For a port assigned a width (w), the number of addressable locations will be 16,384/(w-p) where "p" is the number of parity bits. Each memory location will have a width of "w" (including parity bits). See the DIP signal description for more information of parity. |
| Parity Data Input(s) | DIPA | DIPB | Input | Parity inputs represent additional bits included in the data input path to support error detection. The number of parity bits "p" included in the DI (same as for the DO bus) depends on a port's total data path width (w). See Table 10. |

*Table 9:* **Block RAM Port Signals *(Continued)***

| Signal Description | Port A Signal Name | Port B Signal Name | Direction | Function |
|---|---|---|---|---|
| Data Output Bus | DOA | DOB | Output | Basic data access occurs whenever WE is inactive. The DO outputs mirror the data stored in the addressed memory location.<br>Data access with WE asserted is also possible if one of the following two attributes is chosen: WRITE_FIRST accesses data before the write takes place. READ_FIRST accesses data after the write occurs.<br>A third attribute, NO_CHANGE, latches the DO outputs upon the assertion of WE.<br>It is possible to configure a port's total data path width (w) to be 1, 2, 4, 9, 18, or 36 bits. This selection applies to both the DI and DO paths. See the DI signal description. |
| Parity Data Output(s) | DOPA | DOPB | Output | Parity inputs represent additional bits included in the data input path to support error detection. The number of parity bits "p" included in the DI (same as for the DO bus) depends on a port's total data path width (w). See Table 10. |
| Write Enable | WEA | WEB | Input | When asserted together with EN, this input enables the writing of data to the RAM. In this case, the data access attributes WRITE_FIRST, READ_FIRST or NO_CHANGE determines if and how data is updated on the DO outputs. See the DO signal description.<br>When WE is inactive with EN asserted, read operations are still possible. In this case, a transparent latch passes data from the addressed memory location to the DO outputs. |
| Clock Enable | ENA | ENB | Input | When asserted, this input enables the CLK signal to synchronize Block RAM functions as follows: the writing of data to the DI inputs (when WE is also asserted), the updating of data at the DO outputs as well as the setting/resetting of the DO output latches.<br>When de-asserted, the above functions are disabled. |
| Set/Reset | SSRA | SSRB | Input | When asserted, this pin forces the DO output latch to the value that the SRVAL attribute is set to. A Set/Reset operation on one port has no effect on the other ports functioning, nor does it disturb the memory's data contents. It is synchronized to the CLK signal. |
| Clock | CLKA | CLKB | Input | This input accepts the clock signal to which read and write operations are synchronized. All associated port inputs are required to meet setup times with respect to the clock signal's active edge. The data output bus responds after a clock-to-out delay referenced to the clock signal's active edge. |

## Port Aspect Ratios

On a given port, it is possible to select a number of different possible widths (w – p) for the DI/DO buses as shown in Table 10. These two buses always have the same width. This data bus width selection is independent for each port. If the data bus width of Port A differs from that of Port B, the Block RAM automatically performs a bus-matching function. When data are written to a port with a narrow bus, then read from a port with a wide bus, the latter port will effectively combine "narrow" words to form "wide" words. Similarly, when data are written into a port with a wide bus, then read from a port with a narrow bus, the latter port will divide

"wide" words to form "narrow" words. When the data bus width is eight bits or greater, extra parity bits become available. The width of the total data path (w) is the sum of the DI/DO bus width and any parity bits (p).

The width selection made for the DI/DO bus determines the number of address lines according to the relationship expressed below:

$$r = 14 - [\log(w-p)/\log(2)] \quad \textbf{(1)}$$

In turn, the number of address lines delimits the total number (n) of addressable locations or depth according to the following equation:

$$n = 2^r \quad \textbf{(2)}$$

The product of w and n yields the total block RAM capacity. Equations (1) and (2) show that as the data bus width increases, the number of address lines along with the number of addressable memory locations decreases. Using the permissible DI/DO bus widths as inputs to these equations provides the bus width and memory capacity measures shown in Table 10.

*Table 10:* **Port Aspect Ratios for Port A or B**

| DI/DO Bus Width (w – p bits) | DIP/DOP Bus Width (p bits) | Total Data Path Width (w bits) | ADDR Bus Width (r bits) | No. of Addressable Locations (n) | Block RAM Capacity (bits) |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 14 | 16,384 | 16,384 |
| 2 | 0 | 2 | 13 | 8,192 | 16,384 |
| 4 | 0 | 4 | 12 | 4,096 | 16,384 |
| 8 | 1 | 9 | 11 | 2,048 | 18,432 |
| 16 | 2 | 18 | 10 | 1,024 | 18,432 |
| 32 | 4 | 36 | 9 | 512 | 18,432 |

## Block RAM Data Operations

Writing data to and accessing data from the block RAM are synchronous operations that take place independently on each of the two ports.

The waveforms for the write operation are shown in the top half of the Figure 9, Figure 10, and Figure 11. When the WE and EN signals enable the active edge of CLK, data at the DI input bus is written to the block RAM location addressed by the ADDR lines.

There are a number of different conditions under which data can be accessed at the DO outputs. Basic data access always occurs when the WE input is inactive. Under this condition, data stored in the memory location addressed by the ADDR lines passes through a transparent output latch to the DO outputs. The timing for basic data access is shown in the portions of Figure 9, Figure 10, and Figure 11 during which WE is Low.

Data can also be accessed on the DO outputs when asserting the WE input. This is accomplished using two different attributes:

Choosing the WRITE_FIRST attribute, data is written to the addressed memory location on an enabled active CLK edge and is also passed to the DO outputs. WRITE_FIRST timing is shown in the portion of Figure 9 during which WE is High.

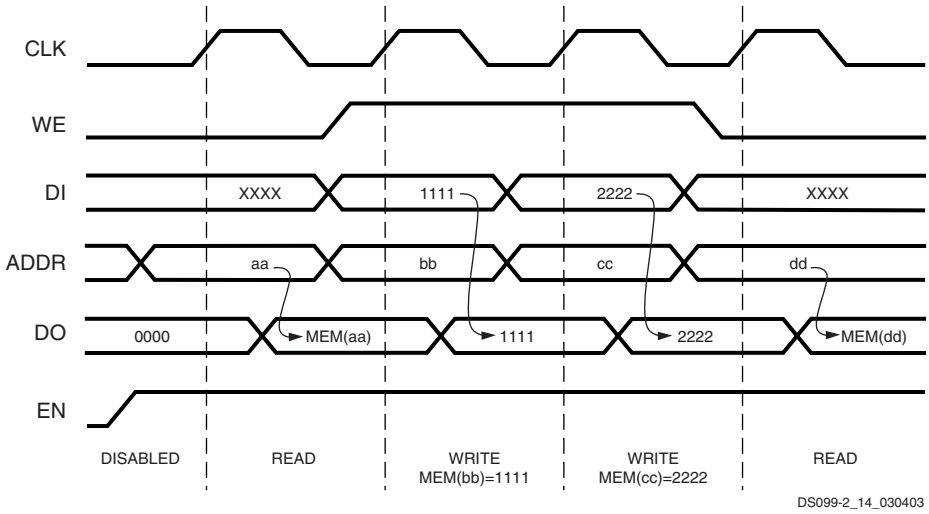



*Figure 9:* **Waveforms of Block RAM Data Operations with WRITE_FIRST Selected**

Choosing the READ_FIRST attribute, data already stored in the addressed location pass to the DO outputs before that location is over-written with new data from the DI inputs on an enabled active CLK edge. READ_FIRST timing is shown in the portion of Figure 10 during which WE is High.

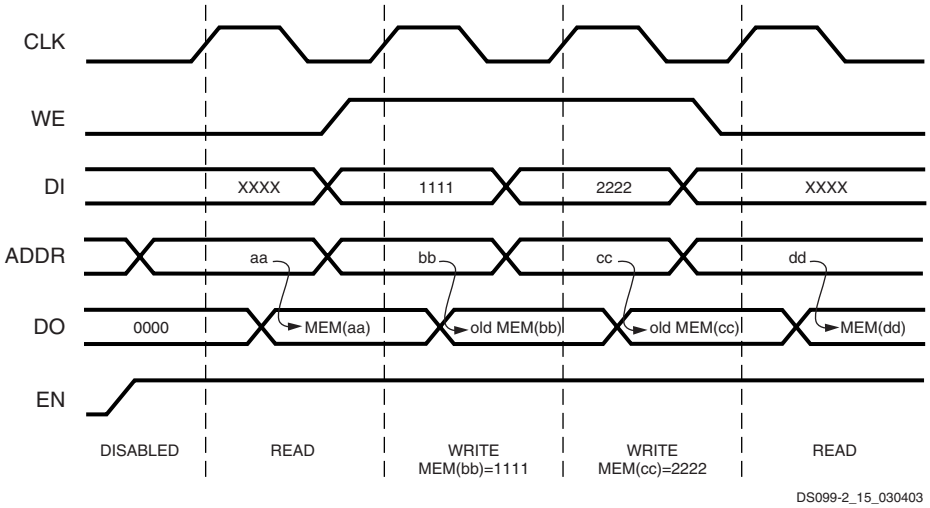



*Figure 10:* **Waveforms of Block RAM Data Operations with READ_FIRST Selected**

Choosing a third attribute called NO_CHANGE puts the DO outputs in a latched state when asserting WE. Under this condition, the DO outputs will retain the data driven just before WE was asserted. NO_CHANGE timing is shown in the portion of Figure 11 during which WE is High.

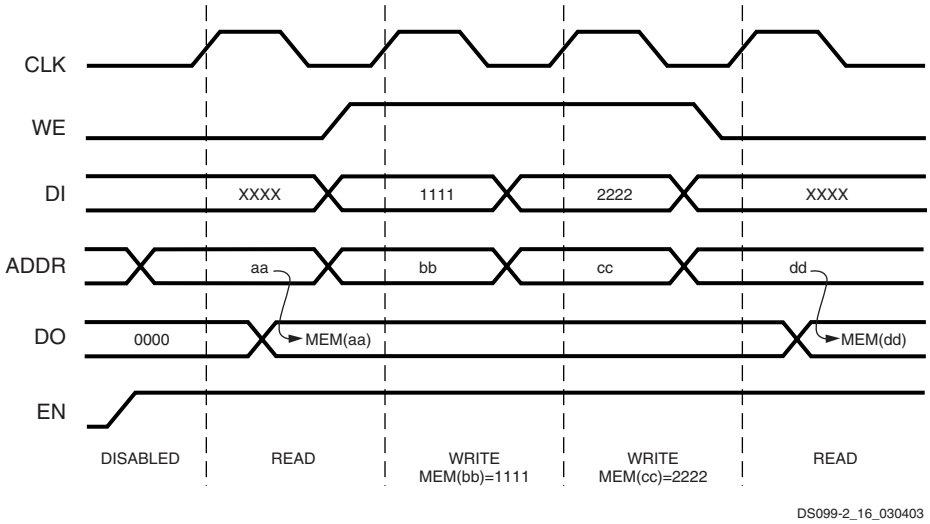


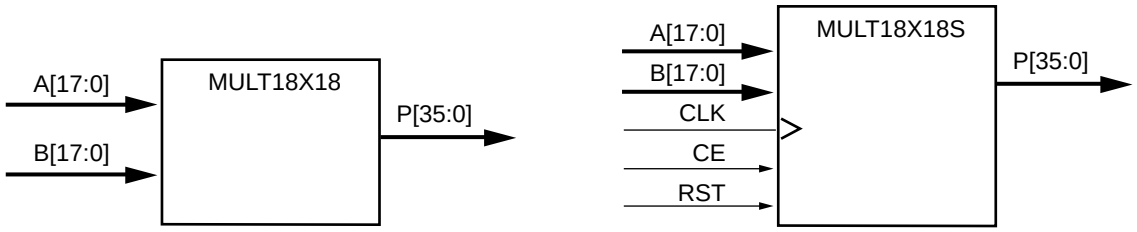

*Figure 11:* **Waveforms of Block RAM Data Operations with NO_CHANGE Selected**

## Dedicated Multipliers

All Spartan-3 devices provide embedded multipliers that accept two 18-bit words as inputs to produce a 36-bit product. This section provides an introduction to multipliers. For further details, see **XAPP467**: *Using Embedded Multipliers in Spartan-3 FPGAs*.

The input buses to the multiplier accept data in two's-complement form (either 18-bit signed or 17-bit unsigned). One such multiplier is matched to each block RAM on the die. The close physical proximity of the two ensures efficient data handling. Cascading multipliers permits multiplicands more than three in number as well as wider than 18-bits. The multiplier is placed in a design using one of two primitives: an asynchronous version called MULT18X18 and a version with a register at the outputs called MULT18X18S, as shown in Figure 12a and Figure 12b, respectively. The signals for these primitives are defined in Table 11.

The CORE Generator system produces multipliers based on these primitives that can be configured to suit a wide range of requirements.




**(a) Asynchronous 18-bit Multiplier**

**(b) 18-bit Multiplier with Register at Outputs**

*Figure 12:* **Embedded Multiplier Primitives**

*Table 11:* **Embedded Multiplier Primitives Descriptions**

| Signal Name | Direction | Function |
|---|---|---|
| A[17:0] | Input | Apply one 18-bit multiplicand to these inputs. The MULT18X18S primitive requires a setup time before the enabled rising edge of CLK. |
| B[17:0] | Input | Apply the other 18-bit multiplicand to these inputs. The MULT18X18S primitive requires a setup time before the enabled rising edge of CLK. |
| P[35:0] | Output | The output on the P bus is a 36-bit product of the multiplicands A and B. In the case of the MULT18X18S primitive, an enabled rising CLK edge updates the P bus. |
| CLK | Input | CLK is only an input to the MULT18X18S primitive. The clock signal applied to this input when enabled by CE, updates the output register that drives the P bus. |
| CE | Input | CE is only an input to the MULT18X18S primitive. Enable for the CLK signal. Asserting this input enables the CLK signal to update the P bus. |
| RST | Input | RST is only an input to the MULT18X18S primitive. Asserting this input resets the output register on an enabled, rising CLK edge, forcing the P bus to all zeroes. |

**Notes:**
1. The control signals CLK, CE and RST have the option of inverted polarity.

## Digital Clock Manager (DCM)

Spartan-3 devices provide flexible, complete control over clock frequency, phase shift and skew through the use of the DCM feature. To accomplish this, the DCM employs a Delay-Locked Loop (DLL), a fully digital control system that uses feedback to maintain clock signal characteristics with a high degree of precision despite normal variations in operating temperature and voltage. This section provides a fundamental description of the DCM. For further information, see **XAPP462**: *Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs*.

Each member of the Spartan-3 family has four DCMs, except the smallest, the XC3S50, which has two DCMs. The DCMs are located at the ends of the outermost Block RAM column(s). See **Figure 1 in Module 1**. The Digital Clock Manager is placed in a design as the “DCM” primitive.

The DCM supports three major functions:

- **Clock-skew Elimination:** Clock skew describes the extent to which clock signals may, under normal circumstances, deviate from zero-phase alignment. It occurs when slight differences in path delays cause the clock signal to arrive at different points on the die at different times. This clock skew can increase set-up and hold time requirements as well as clock-to-out time, which may be undesirable in applications operating at a high frequency, when timing is critical. The DCM eliminates clock skew by aligning the output clock signal it generates with another version of the clock signal that is fed back. As a result, the two clock signals establish a zero-phase relationship. This effectively cancels out clock distribution delays that may lie in the signal path leading from the clock output of the DCM to its feedback input.
- **Frequency Synthesis:** Provided with an input clock signal, the DCM can generate a wide range of different output clock frequencies. This is accomplished by either multiplying and/or dividing the frequency of the input clock signal by any of several different factors.
- **Phase Shifting:** The DCM provides the ability to shift the phase of all its output clock signals with respect to its input clock signal.

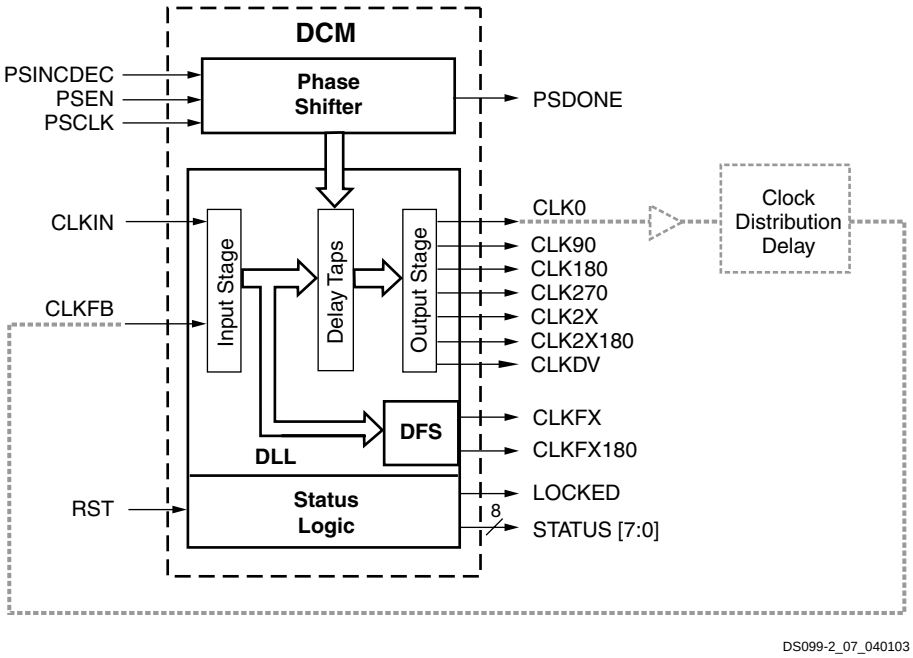



*Figure 13:* **DCM Functional Blocks and Associated Signals**

The DCM has four functional components: the Delay-Locked Loop (DLL), the Digital Frequency Synthesizer (DFS), the Phase Shifter (PS), and the Status Logic. Each component has its associated signals, as shown in Figure 13.

## Delay-Locked Loop (DLL)

The most basic function of the DLL component is to eliminate clock skew. The main signal path of the DLL consists of an input stage, followed by a series of discrete delay elements or *taps,* which in turn leads to an output stage. This path together with logic for phase detection and control forms a system complete with feedback as shown in Figure 14.

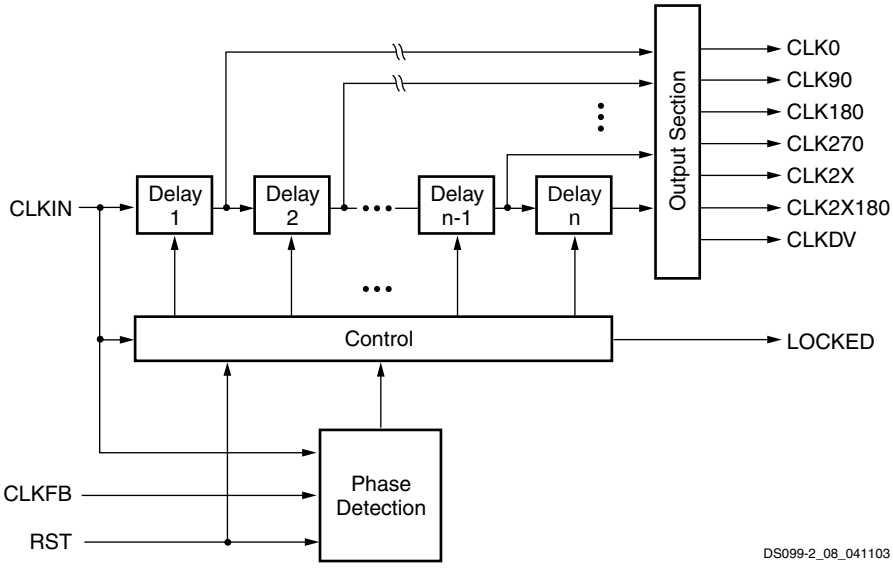



*Figure 14:* **Simplified Functional Diagram of DLL**

The DLL component has two clock inputs, CLKIN and CLKFB, as well as seven clock outputs, CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, and CLKDV as described in Table 12. The clock outputs drive simultaneously; however, the High Frequency mode only supports a subset of the outputs available in the Low Frequency mode. See **DLL Frequency Modes**, page 39. Signals that initialize and report the state of the DLL are discussed in **The Status Logic Component**, page 44.

*Table 12:* **DLL Signals**

| Signal | Direction | Description | Mode Support | |
|---|---|---|---|---|
| | | | Low Frequency | High Frequency |
| CLKIN | Input | Accepts original clock signal. | Yes | Yes |
| CLKFB | Input | Accepts either CLK0 or CLK2X as feed back signal. (Set CLK_FEEDBACK attribute accordingly). | Yes | Yes |
| CLK0 | Output | Generates clock signal with same frequency and phase as CLKIN. | Yes | Yes |
| CLK90 | Output | Generates clock signal with same frequency as CLKIN, only phase-shifted 90°. | Yes | No |
| CLK180 | Output | Generates clock signal with same frequency as CLKIN, only phase-shifted 180°. | Yes | Yes |
| CLK270 | Output | Generates clock signal with same frequency as CLKIN, only phase-shifted 270°. | Yes | No |
| CLK2X | Output | Generates clock signal with same phase as CLKIN, only twice the frequency. | Yes | No |
| CLK2X180 | Output | Generates clock signal with twice the frequency of CLKIN, phase-shifted 180° with respect to CLKIN. | Yes | No |
| CLKDV | Output | Divides the CLKIN frequency by CLKDV_DIVIDE value to generate lower frequency clock signal that is phase-aligned to CLKIN. | Yes | Yes |

The clock signal supplied to the CLKIN input serves as a reference waveform, with which the DLL seeks to align the feedback signal at the CLKFB input. When eliminating clock skew, the common approach to using the DLL is as follows: The CLK0 signal is passed through the clock distribution network to all the registers it synchronizes. These registers are either internal or external to the FPGA. After passing through the clock distribution network, the clock signal returns to the DLL via a feedback line called CLKFB. The control block inside the DLL measures the phase error between CLKFB and CLKIN. This phase error is a measure of the clock skew that the clock distribution network introduces. The control block activates the appropriate number of delay elements to cancel out the clock skew. Once the DLL has brought the CLK0 signal in phase with the CLKIN signal, it asserts the LOCKED output, indicating a “lock” on to the CLKIN signal.

### *DLL Attributes and Related Functions*

A number of different functional options can be set for the DLL component through the use of the attributes described in Table 13. Each attribute is described in detail in the sections that follow:

*Table 13:* **DLL Attributes**

| Attribute | Description | Values |
|---|---|---|
| CLK_FEEDBACK | Chooses either the CLK0 or CLK2X output to drive the CLKFB input | NONE, 1X, 2X |
| DLL_FREQUENCY_MODE | Chooses between High Frequency and Low Frequency modes | LOW, HIGH |
| CLKIN_DIVIDE_BY_2 | Halves the frequency of the CLKIN signal just as it enters the DCM | TRUE, FALSE |
| CLKDV_DIVIDE | Selects constant used to divide the CLKIN input frequency to generate the CLKDV output frequency | 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6.0, 6.5, 7.0, 7.5, 8, 9, 10, 11, 12, 13, 14, 15, and 16. |
| DUTY_CYCLE_CORRECTION | Enables 50% duty cycle correction for the CLK0, CLK90, CLK180, and CLK270 outputs | TRUE, FALSE |

### *DLL Clock Input Connections*

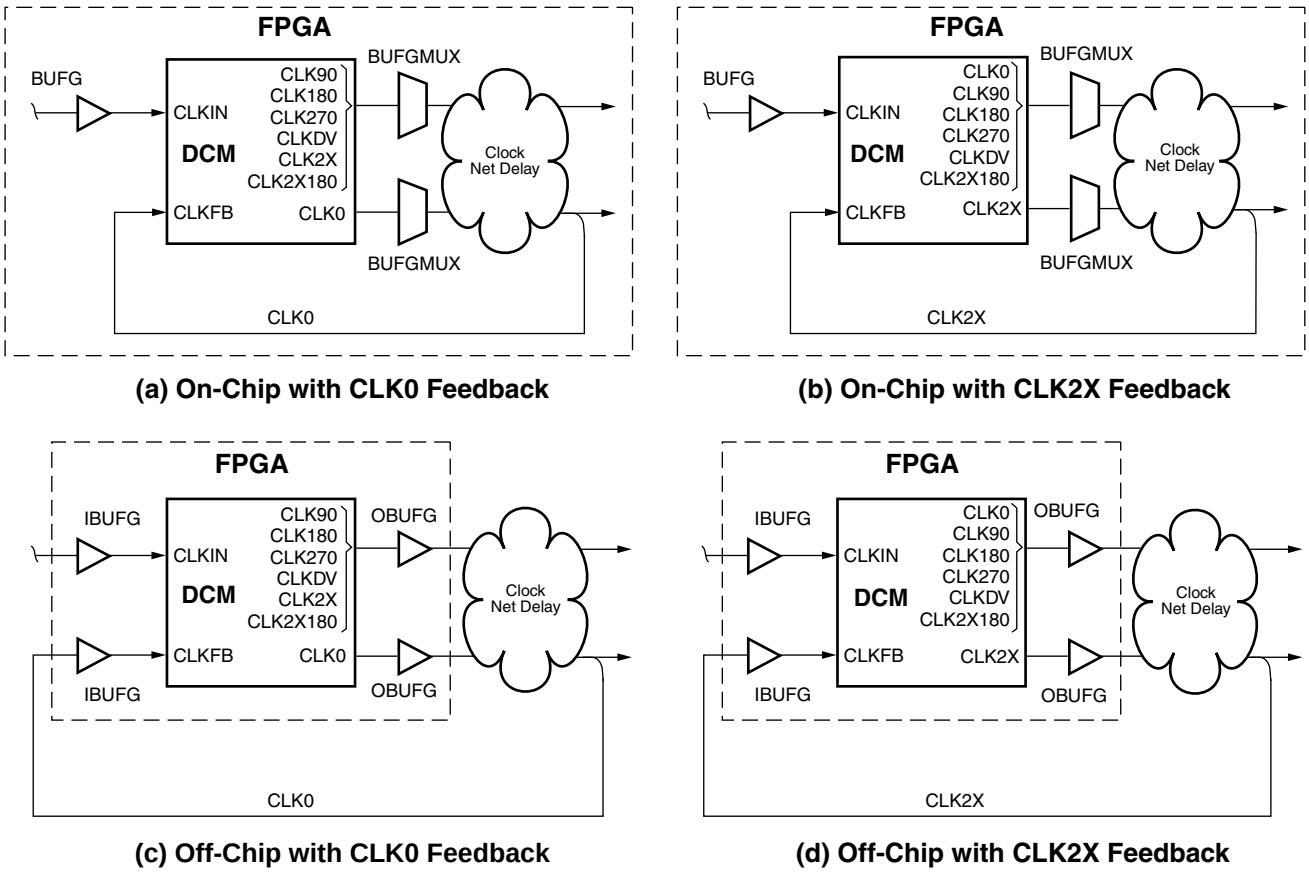
An external clock source enters the FPGA using a Global Clock Input Buffer (IBUFG), which directly accesses the global clock network or an Input Buffer (IBUF). Clock signals within the FPGA drive a global clock net using a Global Clock Multiplexer Buffer (BUFGMUX). The global clock net connects directly to the CLKIN input. The internal and external connections are shown in Figure 15a and Figure 15c, respectively. A differential clock (e.g., LVDS) can serve as an input to CLKIN.

### *DLL Clock Output and Feedback Connections*

As many as four of the nine DCM clock outputs can simultaneously drive the four BUFGMUX buffers on the same die edge (top or bottom). All DCM clock outputs can simultaneously drive general routing resources, including interconnect leading to OBUF buffers.

The feedback loop is essential for DLL operation and is established by driving the CLKFB input with either the CLK0 or the CLK2X signal so that any undesirable clock distribution delay is included in the loop. It is possible to use either of these two signals for synchronizing any of the seven DLL outputs: CLK0, CLK90, CLK180, CLK270, CLKDV, CLK2X, or CLK2X180. The value assigned to the CLK_FEEDBACK attribute must agree with the physical feedback connection: a value of 1X for the CLK0 case, 2X for the CLK2X case. If the DCM is used in an application that does not require the DLL — i.e., only the DFS is used — then there is no feedback loop so CLK_FEEDBACK is set to NONE.

There are two basic cases that determine how to connect the DLL clock outputs and feedback connections: on-chip synchronization and off-chip synchronization, which are illustrated in Figure 15a through Figure 15d.




**(a) On-Chip with CLK0 Feedback**

**(b) On-Chip with CLK2X Feedback**

**(c) Off-Chip with CLK0 Feedback**

**(d) Off-Chip with CLK2X Feedback**

DS099-2_09_071003

**Notes:**

1. In the Low Frequency mode, all seven DLL outputs are available. In the High Frequency mode, only the CLK0, CLK180, and CLKDV outputs are available.

*Figure 15:* **Input Clock, Output Clock, and Feedback Connections for the DLL**

In the on-chip synchronization case (Figure 15a and Figure 15b), it is possible to connect any of the DLL's seven output clock signals through general routing resources to the FPGA's internal registers. Either a Global Clock Buffer (BUFG) or a BUFGMUX affords access to the global clock network. As shown in Figure 15a, the feedback loop is created by routing CLK0 (or CLK2X, in Figure 15b) to a global clock net, which in turn drives the CLKFB input.

In the off-chip synchronization case (Figure 15c and Figure 15d), CLK0 (or CLK2X) plus any of the DLL's other output clock signals exit the FPGA using output buffers (OBUF) to drive an external clock network plus registers on the board. As shown in Figure 15c, the feedback loop is formed by feeding CLK0 (or CLK2X, in Figure 15d) back into the FPGA using an IBUFG, which directly accesses the global clock network, or an IBUF. Then, the global clock net is connected directly to the CLKFB input.

### DLL Frequency Modes

The DLL supports two distinct operating modes, High Frequency and Low Frequency, with each specified over a different clock frequency range. The DLL_FREQUENCY_MODE attribute chooses between the two modes. When the attribute is set to LOW, the Low Frequency mode permits all seven DLL clock outputs to operate over a low-to-moderate frequency range. When the attribute is set to HIGH, the High Frequency mode allows the CLK0, CLK180 and CLKDV outputs to operate at the highest possible frequencies. The remaining DLL clock outputs are not available for use in High Frequency mode.

### Accommodating High Input Frequencies

If the frequency of the CLKIN signal is high such that it exceeds the maximum permitted, divide it down to an acceptable value using the CLKIN_DIVIDE_BY_2 attribute. When this attribute is set to TRUE, the CLKIN frequency is divided by a factor of two just as it enters the DCM.

### Coarse Phase Shift Outputs of the DLL Component

In addition to CLK0 for zero-phase alignment to the CLKIN signal, the DLL also provides the CLK90, CLK180 and CLK270 outputs for 90°, 180° and 270° phase-shifted signals, respectively. These signals are described in Table 12.

Their relative timing in the Low Frequency Mode is shown in Figure 16. The CLK90, CLK180 and CLK270 outputs are not available when operating in the High Frequency mode. (See the description of the DLL_FREQUENCY_MODE attribute in Table 13.) For control in finer increments than 90°, see the **Phase Shifter (PS)**, page 42 section.

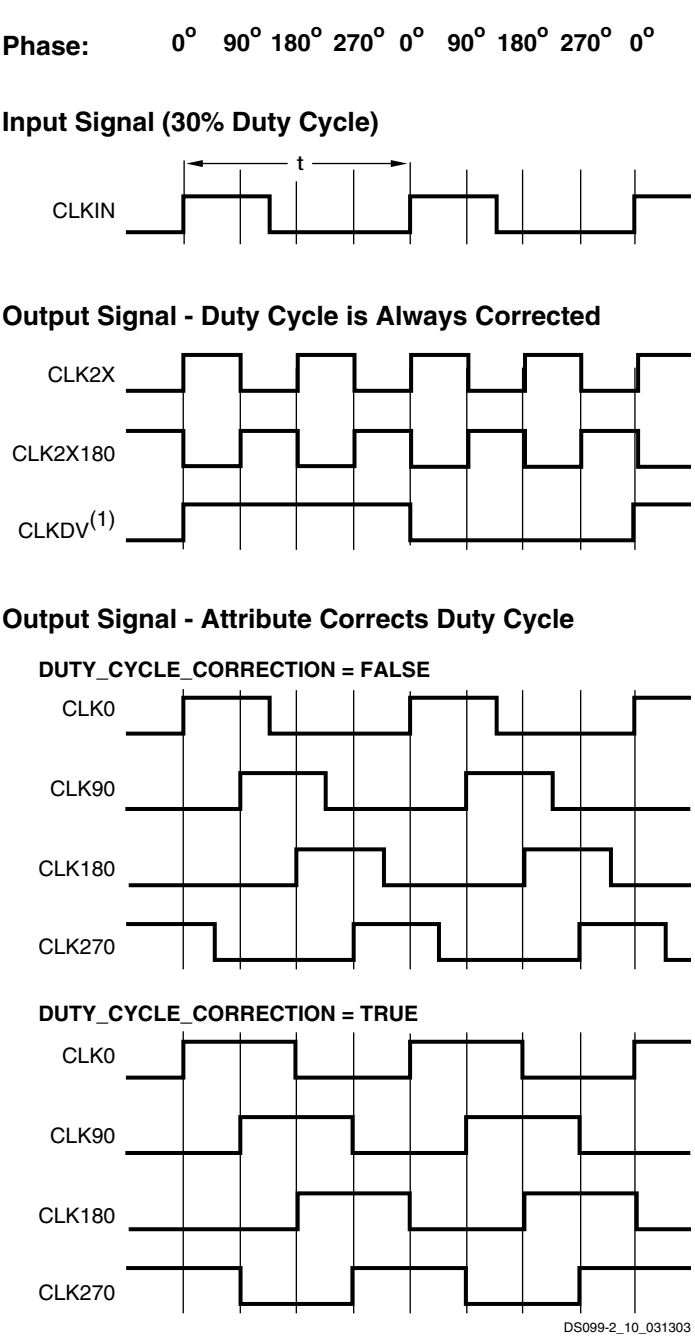
### *Basic Frequency Synthesis Outputs of the DLL Component*

The DLL component provides basic options for frequency multiplication and division in addition to the more flexible synthesis capability of the DFS component, described in a later section. These operations result in output clock signals with frequencies that are either a fraction (for division) or a multiple (for multiplication) of the incoming clock frequency. The CLK2X output produces an in-phase signal that is twice the frequency of CLKIN. The CLK2X180 output also doubles the frequency, but is 180° out-of-phase with respect to CLKIN. The CLKDIV output generates a clock frequency that is a predetermined fraction of the CLKIN frequency. The CLKDV_DIVIDE attribute determines the factor used to divide the CLKIN frequency. The attribute can be set to various values as described in Table 13. The basic frequency synthesis outputs are described in Table 12. Their relative timing in the Low Frequency Mode is shown in Figure 16.

The CLK2X and CLK2X180 outputs are not available when operating in the High Frequency mode. (See the description of the DLL_FREQUENCY_MODE attribute in Table 14.)

### *Duty Cycle Correction of DLL Clock Outputs*

The CLK2X(1), CLK2X180, and CLKDV(2) output signals ordinarily exhibit a 50% duty cycle – even if the incoming CLKIN signal has a different duty cycle. Fifty-percent duty cycle means that the High and Low times of each clock cycle are equal. The DUTY_CYCLE_CORRECTION attribute determines whether or not duty cycle correction is applied to the CLK0, CLK90, CLK180 and CLK270 outputs. If DUTY_CYCLE_CORRECTION is set to TRUE, then the duty cycle of these four outputs is corrected to 50%. If DUTY_CYCLE_CORRECTION is set to FALSE, then these outputs exhibit the same duty cycle as the CLKIN signal. Figure 16 compares the characteristics of the DLL's output signals to those of the CLKIN signal.




**Notes:**
1. The DLL attribute CLKDV_DIVIDE is set to 2.

*Figure 16:* **Characteristics of the DLL Clock Outputs**

1. The CLK2X output generates a 25% duty cycle clock at the same frequency as the CLKIN signal until the DLL has achieved lock.
2. The duty cycle of the CLKDV outputs may differ somewhat from 50% (i.e., the signal will be High for less than 50% of the period) when the CLKDV_DIVIDE attribute is set to a non-integer value *and* the DLL is operating in the High Frequency mode.

## Digital Frequency Synthesizer (DFS)

The DFS component generates clock signals the frequency of which is a product of the clock frequency at the CLKIN input and a ratio of two user-determined integers. Because of the wide range of possible output frequencies such a ratio permits, the DFS feature provides still further flexibility than the DLL's basic synthesis options as described in the preceding section. The DFS component's two dedicated outputs, CLKFX and CLKFX180, are defined in Table 15.

The signal at the CLKFX180 output is essentially an inversion of the CLKFX signal. These two outputs always exhibit a 50% duty cycle. This is true even when the CLKIN signal does not. These DFS clock outputs are driven at the same time as the DLL's seven clock outputs.

The numerator of the ratio is the integer value assigned to the attribute CLKFX_MULTIPLY and the denominator is the integer value assigned to the attribute CLKFX_DIVIDE. These attributes are described in Table 14.

The output frequency ($f_{CLKFX}$) can be expressed as a function of the incoming clock frequency ($f_{CLKIN}$) as follows:

$$f_{CLKFX} = f_{CLKIN}*(CLKFX\_MULTIPLY/CLKFX\_DIVIDE) \quad \textbf{(3)}$$

Regarding the two attributes, it is possible to assign any combination of integer values, provided that two conditions are met:

1. The two values fall within their corresponding ranges, as specified in Table 14.
2. The $f_{CLKFX}$ frequency calculated from the above expression accords with the DCM's operating frequency specifications.

For example, if CLKFX_MULTIPLY = 5 and CLKFX_DIVIDE = 3, then the frequency of the output clock signal would be 5/3 that of the input clock signal.

### *DFS Frequency Modes*

The DFS supports two operating modes, High Frequency and Low Frequency, with each specified over a different clock frequency range. The DFS_FREQUENCY_MODE attribute chooses between the two modes. When the attribute is set to LOW, the Low Frequency mode permits the two DFS outputs to operate over a low-to-moderate frequency range. When the attribute is set to HIGH, the High Frequency mode allows both these outputs to operate at the highest possible frequencies.

### *DFS With or Without the DLL*

The DFS component can be used with or without the DLL component:

Without the DLL, the DFS component multiplies or divides the CLKIN signal frequency according to the respective CLKFX_MULTIPLY and CLKFX_DIVIDE values, generating a clock with the new target frequency on the CLKFX and CLKFX180 outputs. Though classified as belonging to the DLL component, the CLKIN input is shared with the DFS component. This case does not employ feedback loop; therefore, it cannot correct for clock distribution delay.

With the DLL, the DFS operates as described in the preceding case, only with the additional benefit of eliminating the clock distribution delay. In this case, a feedback loop from the CLK0 output to the CLKFB input must be present.

The DLL and DFS components work together to achieve this phase correction as follows: Given values for the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes, the DLL selects the delay element for which the output clock edge coincides with the input clock edge whenever mathematically possible. For example, when CLKFX_MULTIPLY = 5 and CLKFX_DIVIDE = 3, the input and output clock edges will coincide every three input periods, which is equivalent in time to five output periods.

Smaller CLKFX_MULTIPLY and CLKFX_DIVIDE values achieve faster lock times. With no factors common to the two attributes, alignment will occur once with every number of cycles equal to the CLKFX_DIVIDE value. Therefore, it is recommended that the user reduce these values by factoring wherever possible. For example, given CLKFX_MULTIPLY = 9 and CLKFX_DIVIDE = 6, removing a factor of three yields CLKFX_MULTIPLY = 3 and CLKFX_DIVIDE = 2. While both value-pairs will result in the multiplication of clock frequency by 3/2, the latter value-pair will enable the DLL to lock more quickly.

*Table 14:* **DFS Attributes**

| Attribute | Description | Values |
|---|---|---|
| DFS_FREQUENCY_MODE | Chooses between High Frequency and Low Frequency modes | Low, High |
| CLKFX_MULTIPLY | Frequency multiplier constant | Integer from 2 to 32 |
| CLKFX_DIVIDE | Frequency divisor constant | Integer from 1 to 32 |

*Table 15:* **DFS Signals**

| Signal | Direction | Description |
|---|---|---|
| CLKFX | Output | Multiplies the CLKIN frequency by the attribute-value ratio (CLKFX_MULTIPLY/CLKFX_DIVIDE) to generate a clock signal with a new target frequency. |
| CLKFX180 | Output | Generates a clock signal with same frequency as CLKFX, only shifted 180° out-of-phase. |

### DFS Clock Output Connections

There are two basic cases that determine how to connect the DFS clock outputs: on-chip and off-chip, which are illustrated in Figure 15a and Figure 15c, respectively. This is similar to what has already been described for the DLL component. See the **DLL Clock Output and Feedback Connections**, page 38 section.

In the on-chip case, it is possible to connect either of the DFS's two output clock signals through general routing resources to the FPGA's internal registers. Either a Global Clock Buffer (BUFG) or a BUFGMUX affords access to the global clock network. The optional feedback loop is formed in this way, routing CLK0 to a global clock net, which in turn drives the CLKFB input.

In the off-chip case, the DFS's two output clock signals, plus CLK0 for an optional feedback loop, can exit the FPGA using output buffers (OBUF) to drive a clock network plus registers on the board. The feedback loop is formed by feeding the CLK0 signal back into the FPGA using an IBUFG, which directly accesses the global clock network, or an IBUF. Then, the global clock net is connected directly to the CLKFB input.

## Phase Shifter (PS)

The DCM provides two approaches to controlling the phase of a DCM clock output signal relative to the CLKIN signal: First, there are nine clock outputs that employ the DLL to achieve a desired phase relationship: CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKDV CLKFX, and CLKFX180. These outputs afford "coarse" phase control.

The second approach uses the PS component described in this section to provide a still finer degree of control. The PS component accomplishes this by introducing a "fine phase shift" ($T_{PS}$) between the CLKFB and CLKIN signals inside the DLL component. The user can control this fine phase shift down to a resolution of 1/256 of a CLKIN cycle or one tap delay (DCM_TAP), whichever is greater. When in use, the PS component shifts the phase of all nine DCM clock output signals together. If the PS component is used together with a DCM clock output such as the CLK90, CLK180, CLK270, CLK2X180 and CLKFX180, then the fine phase shift of the former gets added to the coarse phase shift of the latter.

### PS Component Enabling and Mode Selection

The CLKOUT_PHASE_SHIFT attribute enables the PS component for use in addition to selecting between two operating modes. As described in Table 16, this attribute has three possible values: NONE, FIXED and VARIABLE. When CLKOUT_PHASE_SHIFT is set to NONE, the PS component is disabled and its inputs, PSEN, PSCLK, and PSINCDEC, must be tied to GND. The set of waveforms in Figure 17a shows the disabled case, where the DLL maintains a zero-phase alignment of signals CLKFB and CLKIN upon which the PS component has no effect. The PS component is enabled by setting the attribute to either the FIXED or VARIABLE values, which select the Fixed Phase mode and the Variable Phase mode, respectively. These two modes are described in the sections that follow

### Determining the Fine Phase Shift

The user controls the phase shift of CLKFB relative to CLKIN by setting and/or adjusting the value of the PHASE_SHIFT attribute. This value must be an integer ranging from –255 to +255. The PS component uses this value to calculate the desired fine phase shift ($T_{PS}$) as a fraction of the CLKIN period ($T_{CLKIN}$). Given values for PHASE-SHIFT and $T_{CLKIN}$, it is possible to calculate $T_{PS}$ as follows:

$$T_{PS} = (\text{PHASE\_SHIFT}/256)*T_{CLKIN} \quad \textbf{(4)}$$

Both the Fixed Phase and Variable Phase operating modes employ this calculation. If the PHASE_SHIFT value is zero, then CLKFB and CLKIN will be in phase, the same as when the PS component is disabled. When the PHASE_SHIFT value is positive, the CLKFB signal will be shifted later in time with respect to CLKIN. If the attribute value is negative, the CLKFB signal will be shifted earlier in time with respect to CLKIN.

### The Fixed Phase Mode

This mode fixes the desired fine phase shift to a fraction of the $T_{CLKIN}$, as determined by Equation (4) and its user-selected PHASE_SHIFT value P. The set of waveforms in Figure 17b illustrates the relationship between CLKFB and CLKIN in the Fixed Phase mode. In the Fixed Phase mode, the PSEN, PSCLK and PSINCDEC inputs are not used and must be tied to GND.

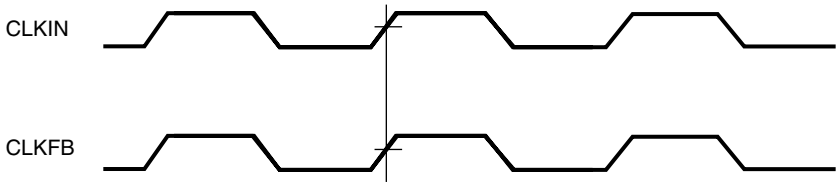
*Table 16:* **PS Attributes**

| Attribute | Description | Values |
|---|---|---|
| CLKOUT_PHASE_SHIFT | Disables PS component or chooses between Fixed Phase and Variable Phase modes. | NONE, FIXED, VARIABLE |
| PHASE_SHIFT | Determines size and direction of initial fine phase shift. | Integers from –255 to +255(1) |

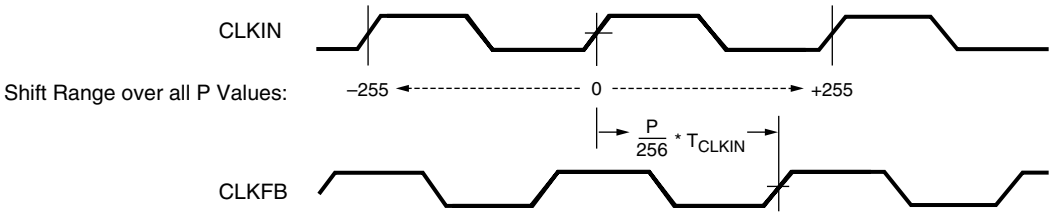
**Notes:**

1. The practical range of values will be less when $T_{CLKIN}$ > FINE_SHIFT_RANGE in the Fixed Phase mode, also when $T_{CLKIN}$ > (FINE_SHIFT_RANGE)/2 in the Variable Phase mode. the FINE_SHIFT_RANGE represents the sum total delay of all taps.

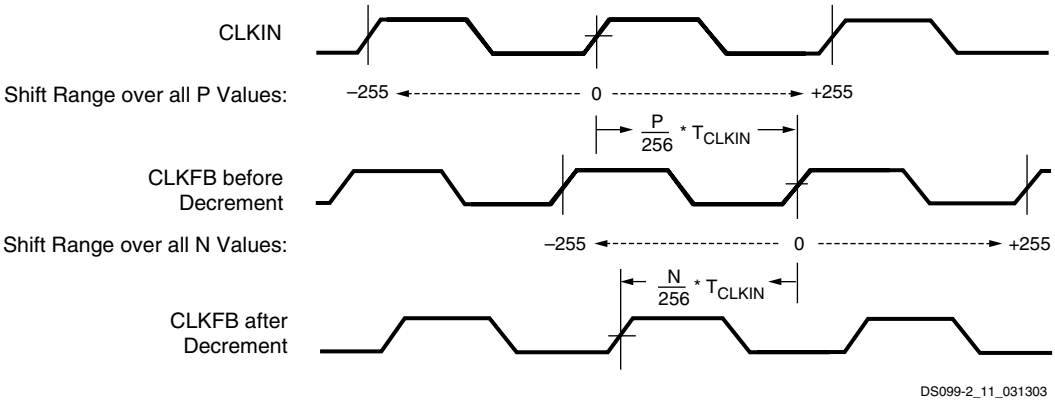
**a. CLKOUT_PHASE_SHIFT = NONE**




**b. CLKOUT_PHASE_SHIFT = FIXED**




**c. CLKOUT_PHASE_SHIFT = VARIABLE**




**Notes:**

1. P represents the integer value ranging from –255 to +255 to which the PHASE_SHIFT attribute is assigned.
2. N is an integer value ranging from –255 to +255 that represents the net phase shift effect from a series of increment and/or decrement operations.

   N = {Total number of increments} – {Total number of decrements}

   A positive value for N indicates a net increment; a negative value indicates a net decrement.

*Figure 17:* **Phase Shifter Waveforms**

*Table 17:* **Signals for Variable Phase Mode**

| Signal | Direction | Description |
|---|---|---|
| PSEN(1) | Input | Enables PSCLK for variable phase adjustment. |
| PSCLK(1) | Input | Clock to synchronize phase shift adjustment. |
| PSINCDEC(1) | Input | Chooses between increment and decrement for phase adjustment. It is synchronized to the PSCLK signal. |
| PSDONE | Output | Goes High to indicate that present phase adjustment is complete and PS component is ready for next phase adjustment request. It is synchronized to the PSCLK signal. |

**Notes:**
1. It is possible to program this input for either a true or inverted polarity

### *The Variable Phase Mode*

The "Variable Phase" mode dynamically adjusts the fine phase shift over time using three inputs to the PS component, namely PSEN, PSCLK and PSINCDEC, as defined in Table 17.

Just following device configuration, the PS component initially determines $T_{PS}$ by evaluating Equation (4) for the value assigned to the PHASE_SHIFT attribute. Then to dynamically adjust that phase shift, use the three PS inputs to increase or decrease the fine phase shift.

PSINCDEC is synchronized to the PSCLK clock signal, which is enabled by asserting PSEN. It is possible to drive the PSCLK input with the CLKIN signal or any other clock signal. A request for phase adjustment is entered as follows: For each PSCLK cycle that PSINCDEC is High, the PS component adds 1/256 of a CLKIN cycle to $T_{PS}$. Similarly, for each enabled PSCLK cycle that PSINCDEC is Low, the PS component subtracts 1/256 of a CLKIN cycle from $T_{PS}$. The phase adjustment may require as many as 100 CLKIN cycles plus three PSCLK cycles to take effect, at which point the output PSDONE goes High for one PSCLK cycle. This pulse indicates that the PS component has finished the present adjustment and is now ready for the next request. Asserting the Reset (RST) input, returns $T_{PS}$ to its original shift time, as determined by the PHASE_SHIFT attribute value. The set of waveforms in Figure 17c illustrates the relationship between CLKFB and CLKIN in the Variable Phase mode.

## The Status Logic Component

The Status Logic component not only reports on the state of the DCM but also provides a means of resetting the DCM to an initial known state. The signals associated with the Status Logic component are described in Table 18.

As a rule, the Reset (RST) input is asserted only upon configuring the device or changing the CLKIN frequency. A DCM reset does not affect attribute values (e.g., CLKFX_MULTIPLY and CLKFX_DIVIDE). If not used, RST must be tied to GND.

The eight bits of the STATUS bus are defined in Table 19.

*Table 18:* **Status Logic Signals**

| Signal | Direction | Description |
|---|---|---|
| RST | Input | A High resets the entire DCM to its initial power-on state. Initializes the DLL taps for a delay of zero. Sets the LOCKED output Low. This input is asynchronous. |
| STATUS[7:0] | Output | The bit values on the STATUS bus provide information regarding the state of DLL and PS operation |
| LOCKED | Output | Indicates that the CLKIN and CLKFB signals are in phase by going High. The two signals are out-of-phase when Low. |

*Table 19:* **DCM STATUS Bus**

| Bit | Name | Description |
|---|---|---|
| 0 | Phase Shift Overflow | A value of 1 indicates a phase shift overflow when one of two conditions occur:<br>• Incrementing (or decrementing) TPS beyond 255/256 of a CLKIN cycle.<br>• The DLL is producing its maximum possible phase shift (i.e., all delay taps are active).(1) |
| 1 | CLKIN Activity | A value of 1 indicates that the CLKIN signal is not toggling. A value of 0 indicates toggling. This bit functions only when the CLKFB input is connected.(2) |
| 2 | Reserved | - |
| 3 | Reserved | - |
| 4 | Reserved | - |
| 5 | Reserved | - |
| 6 | Reserved | - |
| 7 | Reserved | - |

**Notes:**
1. The DLL phase shift with all delay taps active is specified as the parameter FINE_SHIFT_RANGE.
2. If only the DFS clock outputs are used, but none of the DLL clock outputs, this bit will not go High when the CLKIN signal stops.

*Table 20:* **Status Attributes**

| Attribute | Description | Values |
|---|---|---|
| STARTUP_WAIT | Delays transition from configuration to user mode until lock condition is achieved. | TRUE, FALSE |

### *Stabilizing DCM Clocks Before User Mode*

It is possible to delay the completion of device configuration until after the DLL has achieved a lock condition using the STARTUP_WAIT attribute described in Table 20. This option ensures that the FPGA does not enter user mode — i.e., begin functional operation — until all system clocks generated by the DCM are stable. In order to achieve the delay, it is necessary to set the attribute to TRUE as well as set the BitGen option LCK_cycle to one of the six cycles making up the Startup phase of configuration. The selected cycle defines the point at which configuration will halt until the LOCKED output goes High.

## Global Clock Network

Spartan-3 devices have eight Global Clock inputs called GCLK0 - GCLK7. These inputs provide access to a low-capacitance, low-skew network that is well-suited to carrying high-frequency signals. The Spartan-3 clock network is shown in Figure 18. GCLK0 through GCLK3 are placed at the center of the die's bottom edge. GCLK4 through GCLK7 are placed at the center of the die's top edge. It is possible to route each of the eight Global Clock inputs to any CLB on the die.

Eight Global Clock Multiplexers (also called BUFGMUX elements) are provided that accept signals from Global Clock inputs and route them to the internal clock network as well as DCMs. Four BUFGMUX elements are placed at the center of the die's bottom edge, just above the GCLK0 - GCLK4 inputs. The remaining four BUFGMUX elements are placed at the center of the die's top edge, just below the GCLK4 - GCLK7 inputs.

Each BUFGMUX element is a 2-to-1 multiplexer that can receive signals from any of the four following sources:

1. One of the four Global Clock inputs on the same side of the die — top or bottom — as the BUFGMUX element in use.
2. Any of four nearby horizontal Double lines.
3. Any of four outputs from the DCM in the right-hand quadrant that is on the same side of the die as the BUFGMUX element in use.
4. Any of four outputs from the DCM in the left-hand quadrant that is on the same side of the die as the BUFGMUX element in use.

Sources 3 and 4 are not available on the XC3S50 die that lacks DCMs.

Each BUFGMUX can switch incoming clock signals to two possible destinations:

1. The vertical spine belonging to the same side of the die — top or bottom — as the BUFGMUX element in use. The two spines — top and bottom — each comprise four vertical clock lines, each running from one of the

BUFGMUX elements on the same side towards the center of the die. At the center of the die, clock signals reach the eight-line horizontal spine, which spans the width of the die. In turn, the horizontal spine branches out into a subsidiary clock interconnect that accesses the CLBs.

2. The clock input of either DCM on the same side of the die — top or bottom — as the BUFGMUX element in use.

A Global clock input is placed in a design using either a BUFGMUX element or the BUFG (Global Clock Buffer) element. For the purpose of minimizing the dynamic power dissipation of the clock network, the Xilinx development software automatically disables all clock line segments that a design does not use.

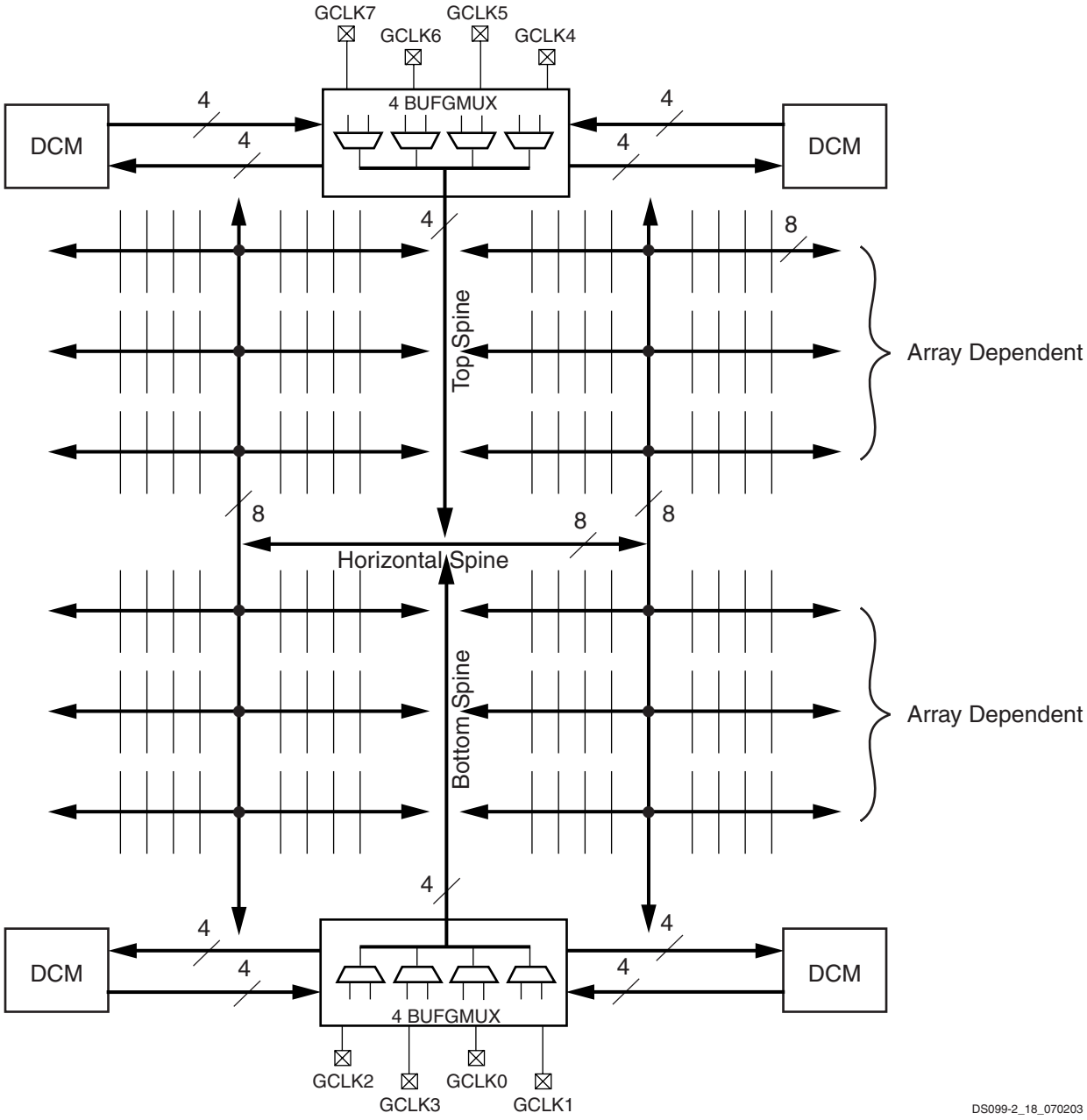



*Figure 18:* **Spartan-3 Clock Network (Top View)**

## Interconnect

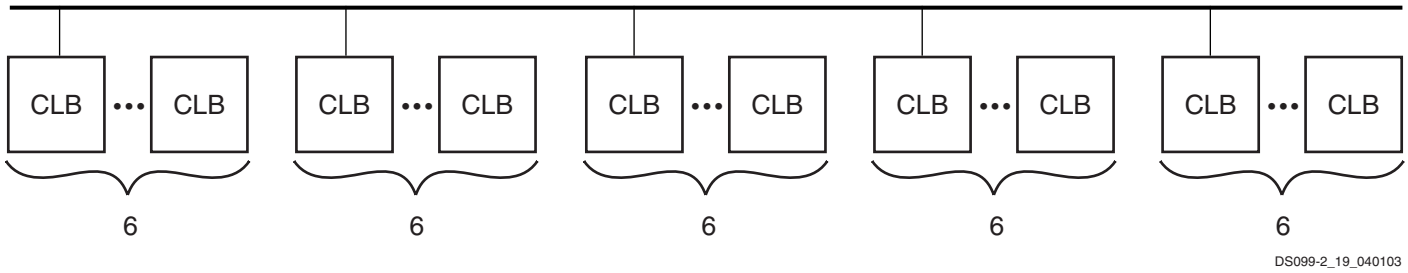
Interconnect (or routing) passes signals among the various functional elements of Spartan-3 devices. There are four kinds of interconnect: Long lines, Hex lines, Double lines, and Direct lines.

Long lines connect to one out of every six CLBs (see Figure 19a). Because of their low capacitance, these lines are well-suited for carrying high-frequency signals with minimal loading effects (e.g. skew). If all eight Global Clock Inputs are already committed and there remain additional clock signals to be assigned, Long lines serve as a good alternative.

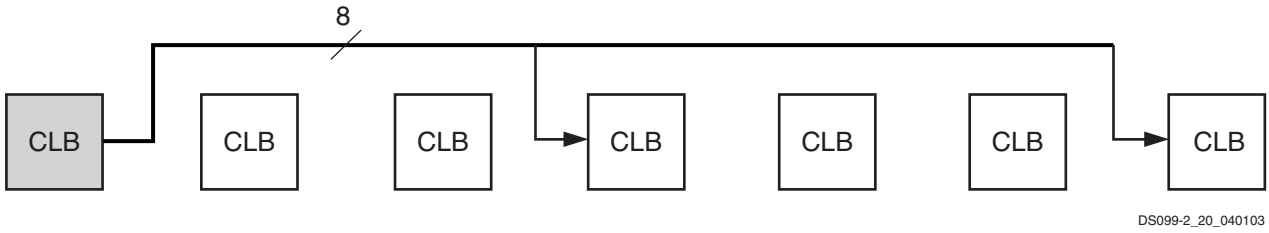
Hex lines connect one out of every three CLBs (see Figure 19b). These lines fall between Long lines and Double lines in terms of capability: Hex lines approach the high-frequency characteristics of Long lines at the same time, offering greater connectivity.

Double lines connect to every other CLB (see Figure 19c). Compared to the types of lines already discussed, Double lines provide a higher degree of flexibility when making connections.

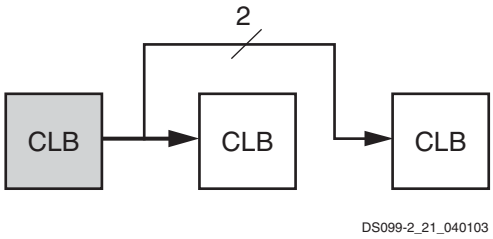
Direct lines afford any CLB direct access to neighboring CLBs (see Figure 19d). These lines are most often used to conduct a signal from a "source" CLB to a Double, Hex, or Long line and then from the longer interconnect back to a Direct line accessing a "destination" CLB.



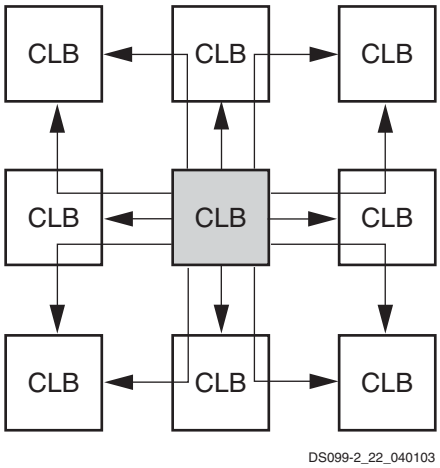

**(a) Long Line**




**(b) Hex Line**




**(c) Double Line**




**(d) Direct Lines**

*Figure 19:* **Types of Interconnect**

# Configuration

Spartan-3 devices are configured by loading application specific configuration data into the internal configuration memory. Configuration is carried out using a subset of the device pins, some of which are "Dedicated" to one function only, while others, indicated by the term "Dual-Purpose", can be re-used as general-purpose User I/Os once configuration is complete.

Depending on the system design, several configuration modes are supported, selectable via mode pins. The mode pins M0, M1, and M2 are Dedicated pins. The mode pin settings are shown in Table 21.

*Table 21:* **Spartan-3 Configuration Mode Pin Settings**

| Configuration Mode(1) | M0 | M1 | M2 | Synchronizing Clock | Data Width | Serial DOUT(2) |
|---|---|---|---|---|---|---|
| Master Serial | 0 | 0 | 0 | CCLK Output | 1 | Yes |
| Slave Serial | 1 | 1 | 1 | CCLK Input | 1 | Yes |
| Master Parallel | 1 | 1 | 0 | CCLK Output | 8 | No |
| Slave Parallel | 0 | 1 | 1 | CCLK Input | 8 | No |
| JTAG | 1 | 0 | 1 | TCK Input | 1 | No |

**Notes:**
1. The voltage levels on the M0, M1, and M2 pins select the configuration mode.
2. The daisy chain is possible only in the Serial modes when DOUT is used.

An additional pin, HSWAP_EN, is used in conjunction with the mode pins to select whether user I/O pins have pull-ups during configuration. By default, HSWAP_EN is tied High (internal pull-up) which shuts off the pull-ups on the user I/O pins during configuration. When HSWAP_EN is tied Low, user I/Os have pull-ups during configuration. Other Dedicated pins are CCLK (the configuration clock pin), DONE, PROG_B, and the boundary-scan pins: TDI, TDO, TMS, and TCK. Depending on the configuration mode chosen, CCLK can be an output generated by the FPGA, or an input accepting an externally generated clock.

A persist option is available which can be used to force the configuration pins to retain their configuration function even after device configuration is complete. If the persist option is not selected then the configuration pins with the exception of CCLK, PROG_B, and DONE can be used as user I/O in normal operation. The persist option does not apply to the boundary-scan related pins. The persist feature is valuable in applications that readback configuration data after entering the User mode.

Table 22 lists the total number of bits required to configure each FPGA as well as the PROMs suitable for storing those bits. See **DS123**: *Platform Flash In-System Programmable Configuration PROMs* data sheet for more information.

*Table 22:* **Spartan-3 Configuration Data**

| Device | File Sizes | Xilinx Platform Flash PROM | |
|---|---|---|---|
| | | Serial Configuration | Parallel Configuration |
| XC3S50 | 439,264 | XCF01S | XCF08P |
| XC3S200 | 1,047,616 | XCF01S | XCF08P |
| XC3S400 | 1,699,136 | XCF02S | XCF08P |
| XC3S1000 | 3,223,488 | XCF04S | XCF08P |
| XC3S1500 | 5,214,784 | XCF08P | XCF08P |
| XC3S2000 | 7,673,024 | XCF08P | XCF08P |
| XC3S4000 | 11,316,864 | XCF16P | XCF16P |
| XC3S5000 | 13,271,936 | XCF16P | XCF16P |

## The Standard Configuration Interface

Configuration signals belong to one of two different categories: Dedicated or Dual-Purpose. Which category determines which of the FPGA's power rails supplies the signal's driver and, thus, helps describe the electrical at the pin.

The Dedicated configuration pins include PROG_B, HSWAP_EN, TDI, TMS, TCK, TDO, CCLK, DONE, and M0-M2. These pins use the $V_{CCAUX}$ lines for power.

The Dual-Purpose configuration pins comprise INIT_B, DOUT, BUSY, RDWR_B, CS_B, and DIN/D0-D7. Each of these pins, according to its bank placement, uses the $V_{CCO}$ lines for either Bank 4 (VCCO_4) or Bank 5 (VCCO_5). All the signals used in the serial configuration modes rely on VCCO_4 power. Signals used in the parallel configuration modes and Readback require from VCCO_5 as well as from VCCO_4.

Both the Dedicated and Dual-Purpose signals described above constitute the configuration interface. In the standard case, this interface is 2.5V-LVCMOS-compatible. This means that 2.5V is applied to the $V_{CCAUX}$, VCCO_4, and VCCO_5 lines (this last in the parallel or Readback case only). One need only apply 2.5 Volts to these $V_{CCO}$ lines from power-on to the end of configuration. Upon entering the User mode, it is possible to switch to supply voltage permitting signal swings other than 2.5V.

## 3.3V-Tolerant Configuration Interface

It is possible to achieve 3.3V-tolerance at the configuration interface simply by adding a few external resistors. This approach may prove useful when it is undesirable to switch the VCCO_4 and VCCO_5 voltages from 2.5V to 3.3V after configuration.

The 3.3V-tolerance is implemented as follows (a similar approach can be used for other supply voltage levels):

First, to power the Dual-Purpose configuration pins, apply 3.3V to the VCCO_4 and (as needed) the VCCO_5 lines. This scales the output voltages and input thresholds associated with these pins so that they become 3.3V-compatible.

Second, to power the Dedicated configuration pins, apply 2.5V to the $V_{CCAUX}$ lines (the same as for the standard interface). In order to achieve 3.3V-tolerance, the Dedicated inputs will require series resistors that limit the incoming current to 10mA or less. The Dedicated outputs will need pull-up resistors to ensure adequate noise margin when the FPGA is driving a High logic level into another device's 3.3V receiver. Choose a power regulator or supply that can tolerate reverse current on the $V_{CCAUX}$ lines.

## Configuration Modes

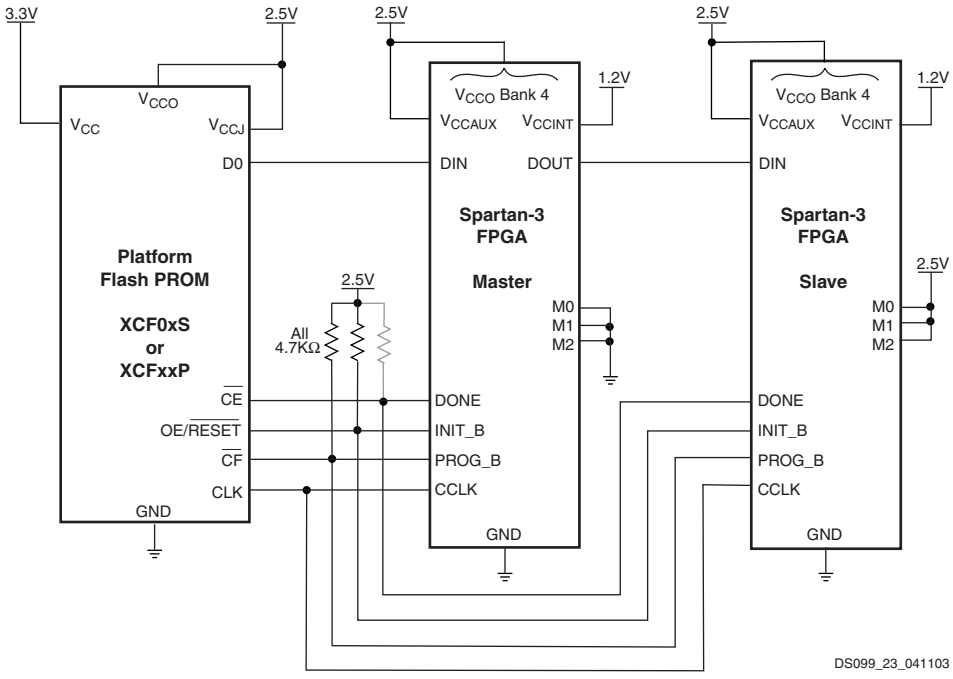
Spartan-3 supports the following five configuration modes:

- Slave Serial mode
- Master Serial mode
- Slave Parallel mode
- Master Parallel mode
- Boundary-Scan (JTAG) mode (IEEE 1532/IEEE 1149.1)

### *Slave Serial Mode*

In Slave Serial mode, the FPGA receives configuration data in bit-serial form from a serial PROM or other serial source of configuration data. The FPGA on the far right of Figure 20 is set for the Slave Serial mode. The CCLK pin on the FPGA is an input in this mode. The serial bitstream must be setup at the DIN input pin a short time before each rising edge of the externally generated CCLK.

Multiple FPGAs can be daisy-chained for configuration from a single source. After a particular FPGA has been configured, the data for the next device is routed internally to the DOUT pin. The data on the DOUT pin changes on the rising edge of CCLK.




**Notes:**

1. There are two ways to use the DONE line. First, one may set the BitGen option DriveDone to "Yes" only for the last FPGA to be configured in the chain shown above (or for the single FPGA as may be the case). This enables the DONE pin to drive High; thus, no pull-up resistor is necessary. DriveDone is set to "No" for the remaining FPGAs in the chain. Second, DriveDone can be set to "No" for all FPGAs. Then all DONE lines are open-drain and require the pull-up resistor shown in grey. In most cases, a value between 3.3KΩ to 4.7KΩ is sufficient. However, when using DONE synchronously with a long chain of FPGAs, cumulative capacitance may necessitate lower resistor values (e.g. down to 330Ω) in order to ensure a rise time within one clock cycle.
2. For information on how to program the FPGA using 3.3V signals and power, see **3.3V-Tolerant Configuration Interface**.

*Figure 20:* **Connection Diagram for Master and Slave Serial Configuration**

Slave Serial mode is selected by applying <111> to the mode pins (M0, M1, and M2). A weak pull-up on the mode pins makes slave serial the default mode if the pins are left unconnected.

### *Master Serial Mode*

In Master Serial mode, the CCLK pin is an output pin. The FPGA just to the right of the PROM in Figure 20 is set for Master Serial mode. It is the FPGA that drives the configuration clock on the CCLK pin to a Xilinx Serial PROM which in turn feeds bit-serial data to the DIN input. The FPGA accepts this data on each rising CCLK edge. After the FPGA has been loaded, the data for the next device in a daisy-chain is presented on the DOUT pin after the rising CCLK edge.

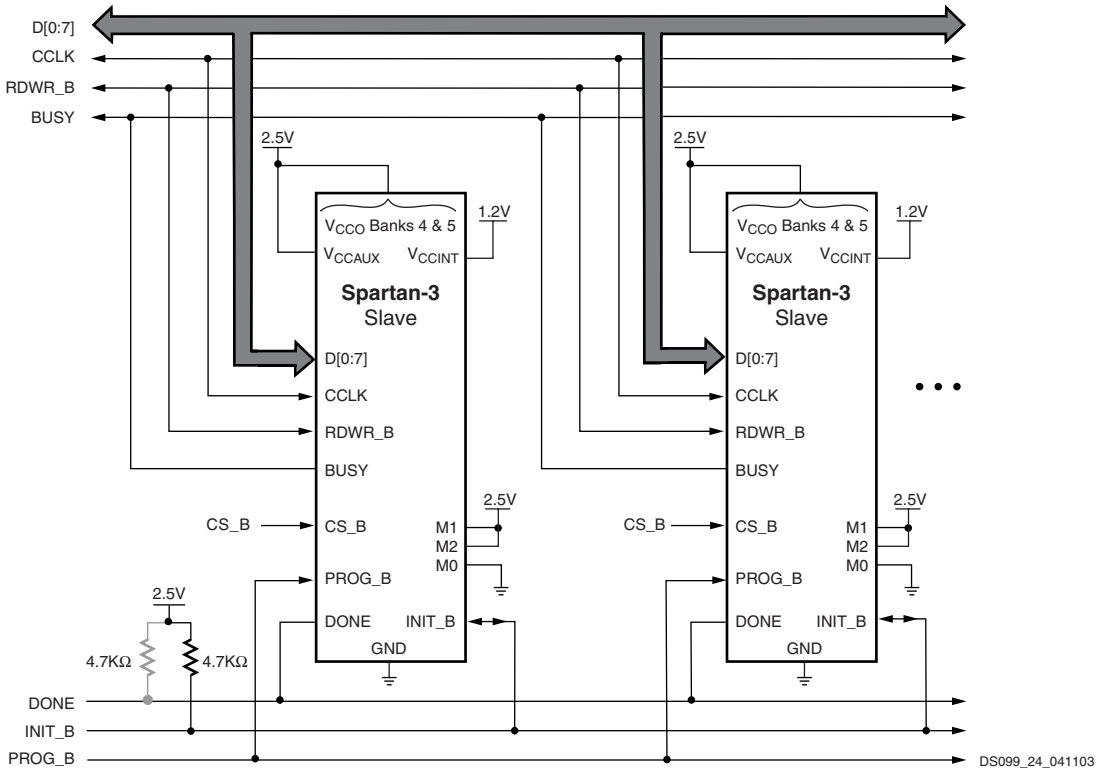
The interface is identical to slave serial except that an internal oscillator is used to generate the configuration clock (CCLK). A wide range of frequencies can be selected for CCLK which always starts at a default frequency of 6 MHz. Configuration bits then switch CCLK to a higher frequency for the remainder of the configuration.

### *Slave Parallel Mode*

The Parallel modes support the fastest configuration. Byte-wide data is written into the FPGA with a BUSY flag controlling the flow of data. An external source provides 8-bit-wide data, CCLK, an active-Low Chip Select (CS_B) signal and an active-Low Write signal (RDWR_B). If BUSY is asserted (High) by the FPGA, the data must be held until BUSY goes Low. Data can also be read using the Slave Parallel mode. If RDWR_B is asserted, configuration data is read out of the FPGA as part of a readback operation.

After configuration, it is possible to use any of the Multipurpose pins (DIN/D0-D7, DOUT/BUSY, INITB, CS_B, and RDWR_B) as User I/Os. To do this, simply set the BitGen option *Persist* to *No* and assign the desired signals to multipurpose configuration pins using the Xilinx development software. Alternatively, it is possible to continue using the configuration port (e.g. all configuration pins taken together) when operating in the User mode. This is accomplished by setting the *Persist* option to *Yes*.

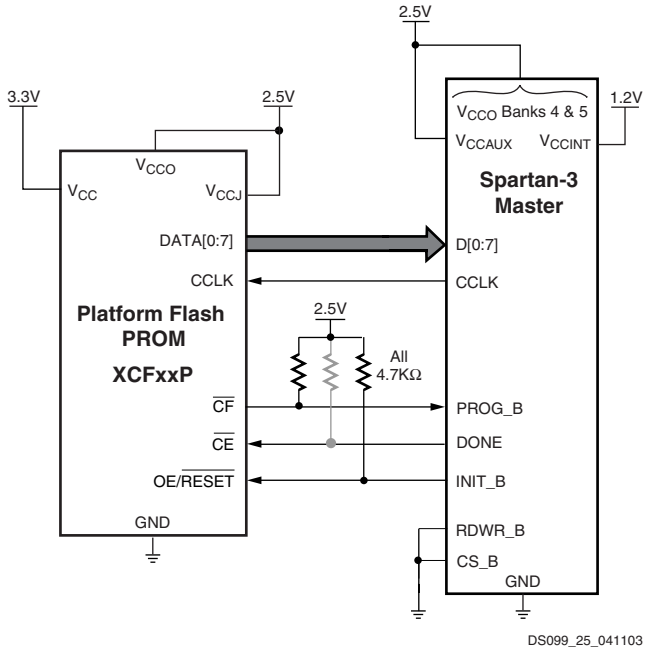
Multiple FPGAs can be configured using the Slave Parallel mode and can be made to start-up simultaneously. Figure 21 shows the device connections. To configure multiple devices in this way, wire the individual CCLK, Data, RDWR_B, and BUSY pins of all the devices in parallel. The individual devices are loaded separately by deasserting the CS_B pin of each device in turn and writing the appropriate data.




**Notes:**

1. There are two ways to use the DONE line. First, one may set the BitGen option DriveDone to "Yes" only for the last FPGA to be configured in the chain shown above (or for the single FPGA as may be the case). This enables the DONE pin to drive High; thus, no pull-up resistor is necessary. DriveDone is set to "No" for the remaining FPGAs in the chain. Second, DriveDone can be set to "No" for all FPGAs. Then all DONE lines are open-drain and require the pull-up resistor shown in grey. In most cases, a value between 3.3KΩ to 4.7KΩ is sufficient. However, when using DONE synchronously with a long chain of FPGAs, cumulative capacitance may necessitate lower resistor values (e.g. down to 330Ω) in order to ensure a rise time within one clock cycle.
2. If the FPGAs use different configuration data files, configure them in sequence by first asserting the CS_B of one FPGA then asserting the CS_B of the other FPGA.
3. For information on how to program the FPGA using 3.3V signals and power, see **3.3V-Tolerant Configuration Interface**.

*Figure 21:* **Connection Diagram for Slave Parallel Configuration**




**Notes:**

1. There are two ways to use the DONE line. First, one may set the BitGen option DriveDone to "Yes" only for the last FPGA to be configured in the chain shown above (or for the single FPGA as may be the case). This enables the DONE pin to drive High; thus, no pull-up resistor is necessary. DriveDone is set to "No" for the remaining FPGAs in the chain. Second, DriveDone can be set to "No" for all FPGAs. Then all DONE lines are open-drain and require the pull-up resistor shown in grey. In most cases, a value between 3.3KΩ to 4.7KΩ is sufficient. However, when using DONE synchronously with a long chain of FPGAs, cumulative capacitance may necessitate lower resistor values (e.g. down to 330Ω) in order to ensure a rise time within one clock cycle.

*Figure 22:* **Connection Diagram for Master Parallel Configuration**

### *Master Parallel Mode*

In this mode, the device is configured byte-wide on a CCLK supplied by the FPGA. Timing is similar to the Slave Parallel mode except that CCLK is supplied by the FPGA. The device connections are shown in Figure 22.

### *Boundary-Scan (JTAG) Mode*

In Boundary-Scan mode, dedicated pins are used for configuring the FPGA. The configuration is done entirely through the IEEE 1149.1 Test Access Port (TAP). FPGA configuration using the Boundary-Scan mode is compliant with the IEEE 1149.1-1993 standard and the new IEEE 1532 standard for In-System Configurable (ISC) devices.

Configuration through the boundary-scan port is always available, independent of the mode selection. Selecting the Boundary-Scan mode simply turns off the other modes.

### *Configuration Sequence*

The configuration of Spartan-3 devices is a three-stage process that occurs after Power-On Reset or the assertion of PROG_B. POR occurs after the $V_{CCINT}$, $V_{CCAUX}$, and $V_{CCO}$ Bank 4 supplies have reached their respective maximum input threshold levels (see **Table 6 in Module 3**). After POR, the three-stage process begins.

First, the configuration memory is cleared. Next, configuration data is loaded into the memory, and finally, the logic is activated by a start-up process. A flow diagram for the configuration sequence of the Serial and Parallel modes is shown in Figure 23. The flow diagram for the Boundary-Scan configuration sequence appears in Figure 24.

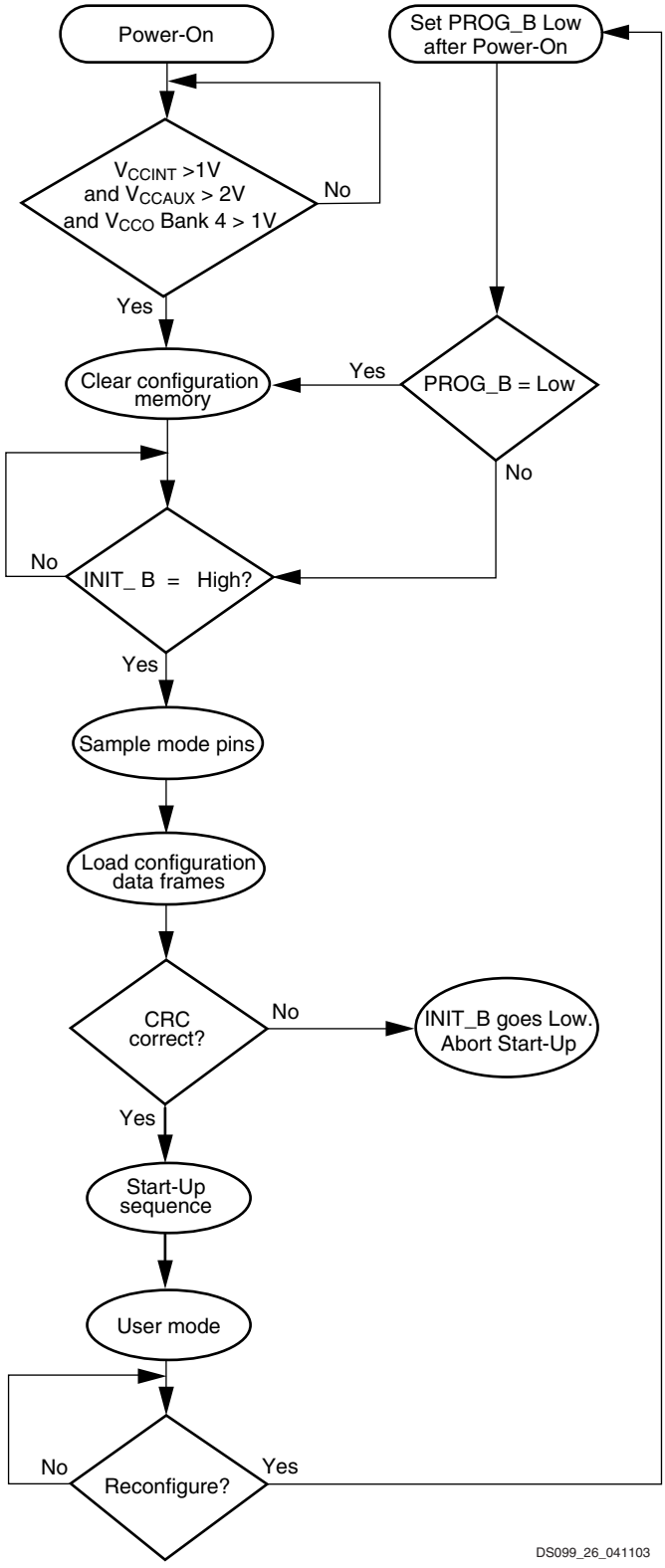



*Figure 23:* **Configuration Flow Diagram for the Serial and Parallel Modes**

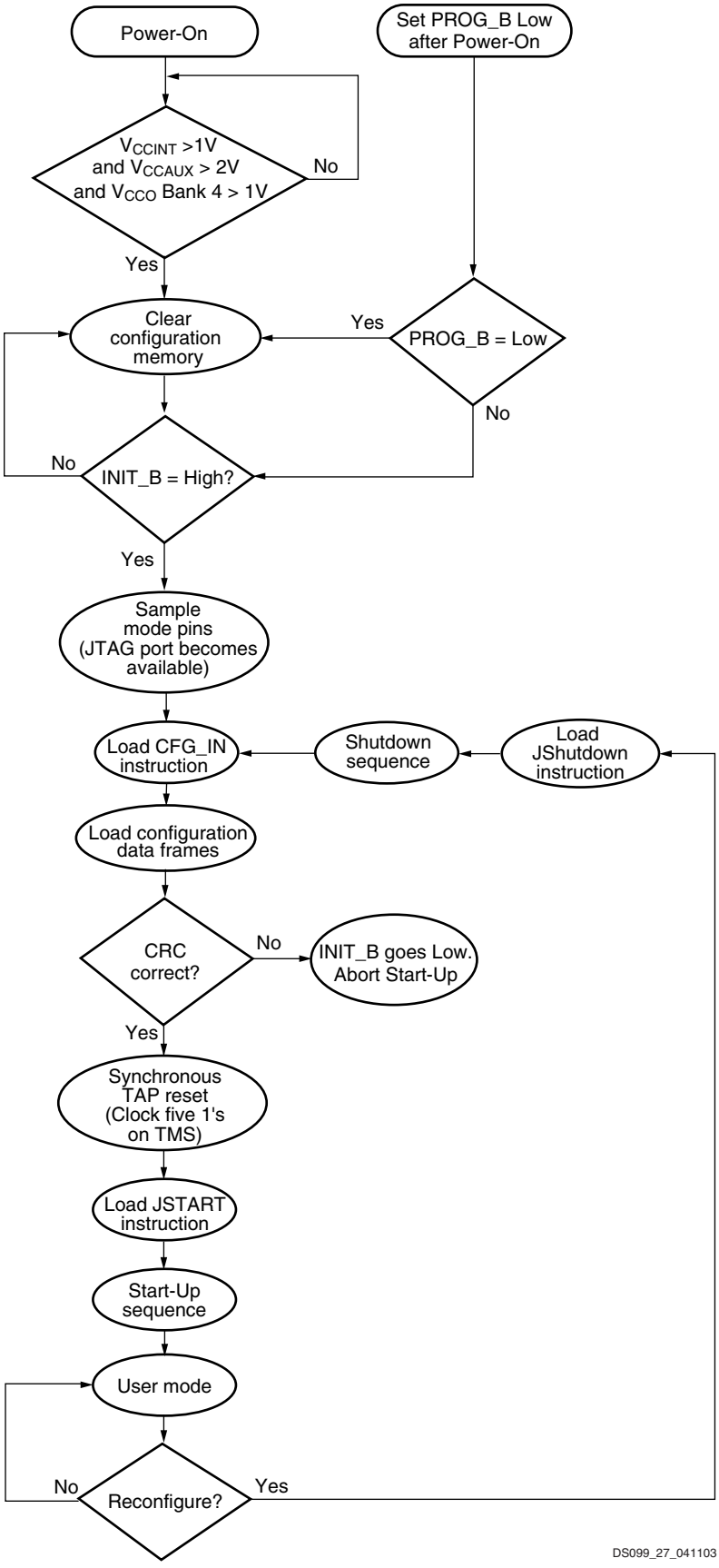
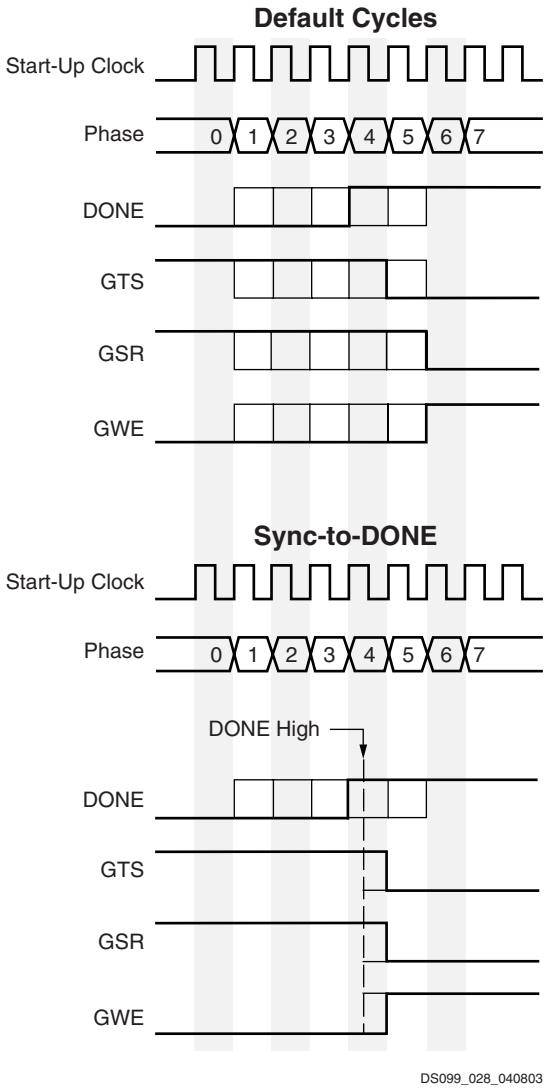



*Figure 24:* **Boundary-Scan Configuration Flow Diagram**

Configuration is automatically initiated after power-on unless it is delayed by the user. INIT_B is an open-drain line that the FPGA holds Low during the clearing of the configuration memory. Extending the time that the pin is Low causes the configuration sequencer to wait. Thus, configuration is delayed by preventing entry into the phase where data is loaded.

The configuration process can also be initiated by asserting the PROG_B pin. The end of the memory-clearing phase is signaled by the INIT_B pin going High. At this point, the configuration data is written to the FPGA. The FPGA holds the Global Set/Reset (GSR) signal active throughout configuration, keeping all flip-flops on the device in a reset state. The completion of the entire process is signaled by the DONE pin going High.




**Notes:**
1. The BitGen option StartupClk in the Xilinx development software selects the CCLK input, TCK input, or a user-designated global clock input (the GCLK0 - GCLK7 pins) for receiving the clock signal that synchronizes Start-Up.

*Figure 25:* **Default Start-Up Sequence**

The default start-up sequence, shown in Figure 25, serves as a transition to the User mode. The default start-up sequence is that one CCLK cycle after DONE goes High, the Global Three-State signal (GTS) is released. This permits device outputs to which signals have been assigned to become active. One CCLK cycle later, the Global Write Enable (GWE) signal is released. This permits the internal storage elements to begin changing state in response to the design logic and the user clock.

The relative timing of configuration events can be changed via the BitGen options in the Xilinx development software. In addition, the GTS and GWE events can be made dependent on the DONE pins of multiple devices all going High, forcing the devices to start synchronously. The sequence can also be paused at any stage, until lock has been achieved on any DCM.

### *Readback*

Using Slave Parallel mode, configuration data from the FPGA can be read back. Readback is supported only in the Slave Parallel and Boundary-Scan modes.

Along with the configuration data, it is possible to read back the contents of all registers, distributed SelectRAM, and block RAM resources. This capability is used for real-time debugging.

## Revision History

| Date | Version No. | Description |
|---|---|---|
| 04/11/03 | 1.0 | Initial Xilinx release |
| 05/19/03 | 1.1 | Added Block RAM column, DCMs, and multipliers to XC3S50 descriptions. |
| 07/11/03 | 1.2 | Explained the configuration port *Persist* option in **Slave Parallel Mode** section. Updated Figure 2 and **Double-Data-Rate Transmission** section to indicate that DDR clocking for the XC3S50 is the same as that for all other Spartan-3 devices. Updated description of I/O voltage tolerance in **ESD Protection** section. In Table 6, changed input termination type for DCI version of the LVCMOS standard to *None*. Added additional flexibility for making DLL connections in Figure 15 and accompanying text. In the **Configuration** section, inserted an explanation of how to choose power supplies for the configuration interface, including guidelines for achieving 3.3V-tolerance. |

## The Spartan-3 Family Data Sheet

**DS099-1**, ***Spartan-3 1.2V FPGA Family: Introduction and Ordering Information*** (Module 1)

**DS099-2, *Spartan-3 1.2V FPGA Family: Functional Description*** (Module 2)

**DS099-3, *Spartan-3 1.2V FPGA Family: DC and Switching Characteristics*** (Module 3)

**DS099-4**, ***Spartan-3 1.2V FPGA Family: Pinout Tables*** (Module 4)

## DC Electrical Characteristics

In this section, some specifications may be designated as Advance or Preliminary. These terms are defined as follows:

**Advance:** Initial estimates based on simulation and/or extrapolation from the characteristics of other families. Values are subject to change. Use as estimates, not for production.

**Preliminary:** Based on characterization. Further changes are not expected.

All specifications are representative of worst-case supply voltage and junction temperature conditions. All specifications are subject to change without notice.

DC and AC characteristics are specified using the same numbers for both commercial and industrial grades unless otherwise noted.

*Table 1:* **Absolute Maximum Ratings(1, 2)**

| Symbol | Description | | Min | Max | Units |
|---|---|---|---|---|---|
| $V_{CCINT}$ | Internal supply voltage | | –0.5 | 1.32 | V |
| $V_{CCAUX}$ | Auxiliary supply voltage | | –0.5 | 3.00 | V |
| $V_{CCO}$ | Output driver supply voltage | | –0.5 | 3.75 | V |
| $V_{REF}$ | Input reference voltage | | –0.5 | $V_{CCO}$ + 0.5 | V |
| $V_{IN}$ | Voltage applied to bidirectional I/O pins as well as unidirectional input and output pins.(3) If present, driver is put in a high-impedance state. | $V_{CCO} \leq$ 3.0V | –0.5 | $V_{CCO}$ + 0.5 | V |
| | | $V_{CCO}$ > 3.0V | –0.3 | 3.75 | V |
| $T_J$ | Operating junction temperature | $V_{CCO} \leq$ 3.0V | - | 125 | °C |
| | | $V_{CCO}$ > 3.0V | - | 105 | °C |
| $T_{SOL}$(5) | Soldering temperature | | - | 220 | °C |
| $T_{STG}$ | Storage temperature | | –65 | 150 | °C |

**Notes:**

1. Stresses beyond those listed under Absolute Maximum Ratings will cause permanent damage to the device. These are stress ratings only; functional operation of the device at these or any other conditions beyond those listed under the Recommended Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time adversely affects device reliability.
2. All parameters representing voltages are measured with respect to GND unless otherwise specified.
3. This specification applies to all User I/O, Multi-Function, and Dedicated pins.
4. For soldering guidelines, see the information on "Packaging and Thermal Characteristics" at www.xilinx.com.

*Table 2:* **Supply Voltage Thresholds for Power-On Reset**

| Symbol | Description | Min | Max | Units |
|---|---|---|---|---|
| $V_{CCINTT}$ | Threshold for the $V_{CCINT}$ supply | 0.4 | 1.0 | V |
| $V_{CCAUXT}$ | Threshold for the $V_{CCAUX}$ supply | 0.8 | 2.0 | V |
| $V_{CCO4T}$ | Threshold for the $V_{CCO}$ Bank 4 supply | 0.4 | 1.0 | V |

**Notes:**
1. $V_{CCINT}$, $V_{CCAUX}$, and $V_{CCO}$ supplies may be applied in any order.
2. During power-on, when the $V_{CCINT}$, $V_{CCO}$ Bank 4, and $V_{CCAUX}$ voltages are rising, none may dip at any point within their respective threshold-voltage ranges.
3. All parameters representing voltages are measured with respect to GND unless otherwise specified.

*Table 3:* **Power Voltage Levels Necessary for Preserving RAM Contents(1)**

| Symbol | Description | Min | Units |
|---|---|---|---|
| $V_{DRINT}$ | $V_{CCINT}$ level required to retain RAM data | 1.0 | V |
| $V_{DRAUX}$ | $V_{CCAUX}$ level required to retain RAM data | 2.0 | V |

**Notes:**
1. RAM contents include configuration data.
2. All parameters representing voltages are measured with respect to GND unless otherwise specified.
3. The level of the $V_{CCO}$ supply has no effect on data retention.

*Table 4:* **General Recommended Operating Conditions**

| Symbol | Description | | Min | Nom | Max | Units |
|---|---|---|---|---|---|---|
| $T_J$ | Junction temperature | Commercial | 0 | - | 85 | °C |
| | | Industrial | –40 | - | 100 | °C |
| $V_{CCINT}$ | Internal supply voltage | | 1.140 | 1.200 | 1.260 | V |
| $V_{CCO}$ (1) | Output driver supply voltage | | 1.140 | - | 3.450 | V |
| $V_{CCAUX}$ | Auxiliary supply voltage | | 2.375 | 2.500 | 2.625 | V |

**Notes:**
1. The $V_{CCO}$ range given here spans the lowest and highest operating voltages of all supported I/O standards. The recommended $V_{CCO}$ range specific to each of the single-ended I/O standards is given in Table 7, and that specific to the differential standards is given in Table 9.
2. All parameters representing voltages are measured with respect to GND unless otherwise specified.

*Table 5:* **General DC Characteristics of User I/O, Multi-Function, and Dedicated Pins**

| Symbol | Description | Test Conditions | | Min | Nom | Max | Units |
|---|---|---|---|---|---|---|---|
| $I_L$ | Leakage current at User I/O, Multi-Function, and Dedicated pins | Driver is in a high-impedance state | | –10 | - | +10 | µA |
| $I_{RPU}$ | Current through pull-up resistor at User I/O, Multi-Function, and Dedicated pins | $V_{IN}$ = 0V | $V_{CCO}$ = 3.3V | 500 | 1000 | 2000 | µA |
| | | | $V_{CCO}$ = 3.0V | 400 | 800 | 1600 | µA |
| | | | $V_{CCO}$ = 2.5V | 250 | 530 | 1100 | µA |
| | | | $V_{CCO}$ = 1.8V | 120 | 270 | 770 | µA |
| | | | $V_{CCO}$ = 1.5V | 70 | 180 | 440 | µA |
| | | | $V_{CCO}$ = 1.2V | 40 | 100 | 300 | µA |
| $I_{RPD}$ | Current through pull-down resistor at User I/O, Multi-Function, and Dedicated pins | $V_{IN}$ = $V_{CCO}$ = 3.3V | | 250 | 520 | 1100 | µA |
| | | $V_{IN}$ = $V_{CCO}$ = 3.0V | | 250 | 520 | 1100 | µA |
| | | $V_{IN}$ = $V_{CCO}$ = 2.5V | | 250 | 520 | 1100 | µA |
| | | $V_{IN}$ = $V_{CCO}$ = 1.8V | | 250 | 520 | 1100 | µA |
| | | $V_{IN}$ = $V_{CCO}$ = 1.5V | | 240 | 510 | 1100 | µA |
| | | $V_{IN}$ = $V_{CCO}$ = 1.2V | | 230 | 480 | 1000 | µA |
| $I_{REF}$ | $V_{REF}$ current per pin | | | –10 | - | +10 | µA |
| $C_{IN}$ | Input capacitance | | | 5 | - | 11 | pF |

**Notes:**
1. The numbers in this table are guaranteed over the conditions set forth in Table 4.

*Table 6:* **Quiescent Supply Current Characteristics**

| Symbol | Description | Device | Commercial | | Industrial | | Units |
|---|---|---|---|---|---|---|---|
| | | | Typ | Max | Typ | Max | |
| $I_{CCINTQ}$ | Quiescent $V_{CCINT}$ supply current | XC3S50 | 10 | | | | mA |
| | | XC3S200 | | | | | mA |
| | | XC3S400 | | | | | mA |
| | | XC3S1000 | 40 | | | | mA |
| | | XC3S1500 | | | | | mA |
| | | XC3S2000 | | | | | mA |
| | | XC3S4000 | | | | | mA |
| | | XC3S5000 | | | | | mA |
| $I_{CCOQ}$ | Quiescent $V_{CCO}$ supply current | XC3S50 | 1.5 | | | | mA |
| | | XC3S200 | | | | | mA |
| | | XC3S400 | | | | | mA |
| | | XC3S1000 | 1.5 | | | | mA |
| | | XC3S1500 | | | | | mA |
| | | XC3S2000 | | | | | mA |
| | | XC3S4000 | | | | | mA |
| | | XC3S5000 | | | | | mA |
| $I_{CCAUXQ}$ | Quiescent $V_{CCAUX}$ supply current | XC3S50 | 7.0 | | | | mA |
| | | XC3S200 | | | | | mA |
| | | XC3S400 | | | | | mA |
| | | XC3S1000 | 25.0 | | | | mA |
| | | XC3S1500 | | | | | mA |
| | | XC3S2000 | | | | | mA |
| | | XC3S4000 | | | | | mA |
| | | XC3S5000 | | | | | mA |

**Notes:**

1. Quiescent supply current is measured with all I/O drivers in a high-impedance state and with all pull-up/pull-down resistors at the I/O pads disabled. For typical values, the ambient temperature ($T_A$) is 85 °C with $V_{CCINT}$ = 1.2V, $V_{CCO}$ = 2.5V, and $V_{CCAUX}$ = 2.5V.
2. The numbers in this table are guaranteed over the conditions set forth in Table 4.

*Table 7:* **Recommended Operating Conditions for User I/Os Using Single-Ended Standards**

| Signal Standard | $V_{CCO}$ | | | $V_{REF}$ | | | $V_{IL}$ | $V_{IH}$ |
|---|---|---|---|---|---|---|---|---|
| | Min (V) | Nom (V) | Max (V) | Min (V) | Nom (V) | Max (V) | Max (V) | Min (V) |
| GTL | - | - | - | 0.74 | 0.8 | 0.86 | $V_{REF}$ - 0.05 | $V_{REF}$ + 0.05 |
| GTL_DCI(2) | - | 1.2 | - | 0.74 | 0.8 | 0.86 | $V_{REF}$ - 0.05 | $V_{REF}$ + 0.05 |
| GTLP | - | - | - | 0.88 | 1 | 1.12 | $V_{REF}$ - 0.1 | $V_{REF}$ + 0.1 |
| GTLP_DCI(2) | - | 1.5 | - | 0.88 | 1 | 1.12 | $V_{REF}$ - 0.1 | $V_{REF}$ + 0.1 |
| HSTL_I, HSTL_I_DCI | 1.4 | 1.5 | 1.6 | 0.68 | 0.75 | 0.9 | $V_{REF}$ - 0.1 | $V_{REF}$ + 0.1 |
| HSTL_III, HSTL_III_DCI | 1.4 | 1.5 | 1.6 | 0.68 | 0.9 | 0.9 | $V_{REF}$ - 0.1 | $V_{REF}$ + 0.1 |
| HSTL_I_18, HSTL_I_DCI_18 | 1.7 | 1.8 | 1.9 | - | 0.9 | - | $V_{REF}$ - 0.1 | $V_{REF}$ + 0.1 |
| HSTL_II_18, HSTL_II_DCI_18 | 1.7 | 1.8 | 1.9 | - | 0.9 | - | $V_{REF}$ - 0.1 | $V_{REF}$ + 0.1 |
| HSTL_III_18, HSTL_III_DCI_18 | 1.7 | 1.8 | 1.9 | - | 1.1 | - | $V_{REF}$ - 0.1 | $V_{REF}$ + 0.1 |
| LVCMOS12(3) | 1.14 | 1.2 | 1.3 | - | - | - | $0.20V_{CCO}$ | $0.70V_{CCO}$ |
| LVCMOS15, LVDCI_15(3) | 1.4 | 1.5 | 1.6 | - | - | - | $0.20V_{CCO}$ | $0.70V_{CCO}$ |
| LVCMOS18, LVDCI_18(3) | 1.7 | 1.8 | 1.9 | - | - | - | $0.20V_{CCO}$ | $0.70V_{CCO}$ |
| LVCMOS25(4), LVDCI_25(3) | 2.3 | 2.5 | 2.7 | - | - | - | 0.7 | 1.7 |
| LVCMOS33, LVDCI_33(3) | 3.0 | 3.3 | 3.45 | - | - | - | 0.8 | 2.0 |
| LVTTL | 3.0 | 3.3 | 3.45 | - | - | - | 0.8 | 2.0 |
| PCI33_3 | 3.0 | 3.0 | 3.0 | - | - | - | $0.30V_{CCO}$ | $0.50V_{CCO}$ |
| SSTL18_I | 1.65 | 1.8 | 1.95 | 0.825 | 0.9 | 0.975 | $V_{REF}$ - 0.125 | $V_{REF}$ + 0.125 |
| SSTL2_I, SSTL2_I_DCI | 2.3 | 2.5 | 2.7 | 1.15 | 1.25 | 1.35 | $V_{REF}$ - 0.15 | $V_{REF}$ + 0.15 |
| SSTL2_II, SSTL2_II_DCI | 2.3 | 2.5 | 2.7 | 1.15 | 1.25 | 1.35 | $V_{REF}$ - 0.15 | $V_{REF}$ + 0.15 |

**Notes:**

1. Descriptions of the symbols used in this table are as follows:
   $V_{CCO}$ -- the supply voltage for the output drivers.
   $V_{REF}$ -- the reference voltage for setting the input switching threshold.
   $V_{IL}$ -- the input voltage that indicates a Low logic level
   $V_{IH}$ -- the input voltage that indicates a High logic level
2. Because the GTL and GTLP standards employ open-drain output buffers, the $V_{CCO}$ supply does not provide drive current. Nevertheless, the $V_{CCO}$ level must always be at or above the termination voltage ($V_{TT}$) and I/O pad voltages.
3. There is approximately 100 mV of hysteresis on inputs using any LVCMOS standard.
4. In the standard case, all Dedicated pins (M0-M2, CCLK, PROG_B, DONE, HSWAP_EN, TCK, TDI, TDO, TMS) use the LVCMOS25 standard and are powered entirely by $V_{CCAUX}$. The Dual-Purpose configuration pins (DIN/D0, D1-D7, CS_B, RDWR_B, BUSY/DOUT, and INIT_B) use the LVCMOS25 standard during configuration. For information on how to program the FPGA using 3.3V signals and power, see the "3.3V-Tolerant Configuration Interface" section in **Module 2**. The recommended $V_{CCO}$ or $V_{CCAUX}$ levels must be applied to the Global Clock Inputs (GCLK0 - GCLK7) according to the signal standards assigned to them—the same as for User Inputs.
5. All parameters representing voltages are measured with respect to GND unless otherwise specified.

*Table 8:* **DC Characteristics of User I/Os Using Single-Ended Standards**

| Signal Standard | Test Conditions | | Logic Level Characteristics | |
|---|---|---|---|---|
| | $I_{OL}$ (mA) | $I_{OH}$ (mA) | $V_{OL}$ Max (V) | $V_{OH}$ Min (V) |
| GTL, GTL_DCI | 32 | - | 0.4 | - |
| GTLP, GTLP_DCI | 36 | - | 0.6 | - |
| HSTL_I, HSTL_I_DCI | 8 | –8 | 0.4 | $V_{CCO}$ - 0.4 |
| HSTL_III, HSTL_III_DCI | 24 | –8 | 0.4 | $V_{CCO}$ - 0.4 |
| HSTL_I_18, HSTL_I_DCI_18 | 8 | –8 | 0.4 | $V_{CCO}$ - 0.4 |
| HSTL_II_18, HSTL_II_DCI_18 | 16 | –16 | 0.4 | $V_{CCO}$ - 0.4 |
| HSTL_III_18, HSTL_III_DCI_18 | 24 | –8 | 0.4 | $V_{CCO}$ - 0.4 |
| LVCMOS12 | 6 | –6 | 0.4 | $V_{CCO}$ - 0.4 |
| LVCMOS15, LVDCI_15 | 12 | –12 | 0.4 | $V_{CCO}$ - 0.4 |
| LVCMOS18, LVDCI_18 | 16 | –16 | 0.4 | $V_{CCO}$ - 0.4 |
| LVCMOS25(2), LVDCI_25(3) | 24 | –24 | 0.4 | $V_{CCO}$ - 0.4 |
| LVCMOS33, LVDCI_33 | 24 | –24 | 0.4 | $V_{CCO}$ - 0.4 |
| LVTTL | 24 | –24 | 0.4 | 2.4 |
| PCI33_3 | Note 5 | Note 5 | $0.10V_{CCO}$ | $0.90V_{CCO}$ |
| SSTL18_I | 6.7 | –6.7 | $V_{TT}$ - 0.475 | $V_{TT}$ + 0.475 |
| SSTL2_I, SSTL2_I_DCI | 7.5 | –7.5 | $V_{TT}$ - 0.61 | $V_{TT}$ + 0.61 |
| SSTL2_II, SSTL2_II_DCI | 15 | –15 | $V_{TT}$ - 0.80 | $V_{TT}$ + 0.80 |

**Notes:**

1. Descriptions of the symbols used in this table are as follows:
   $I_{OL}$ -- the output current condition under which $V_{OL}$ is tested
   $I_{OH}$ -- the output current condition under which $V_{OH}$ is tested
   $V_{OL}$ -- the output voltage that indicates a Low logic level
   $V_{OH}$ -- the output voltage that indicates a High logic level
   $V_{IL}$ -- the input voltage that indicates a Low logic level
   $V_{IH}$ -- the input voltage that indicates a High logic level
   $V_{CCO}$ -- the supply voltage for the output drivers
   $V_{REF}$ -- the reference voltage for setting the input switching threshold
   $V_{TT}$ -- the termination voltage
2. In the standard case, all Dedicated pins (M0-M2, CCLK, PROG_B, DONE, HSWAP_EN, TCK, TDI, TDO, TMS) as well as the Dual-Purpose configuration pins (DIN/D0, D1-D7, CS_B, RDWR_B, BUSY/DOUT, and INIT_B) exhibit LVCMOS25 output characteristics. For information on how to program the FPGA using 3.3V signals and power, see the "3.3V-Tolerant Configuration Interface" section in **Module 2**.
3. All parameters representing voltages are measured with respect to GND unless otherwise specified.
4. The numbers in this table are guaranteed over the conditions set forth in Table 4 and Table 7.
5. Tested according to relevant PCI specifications.

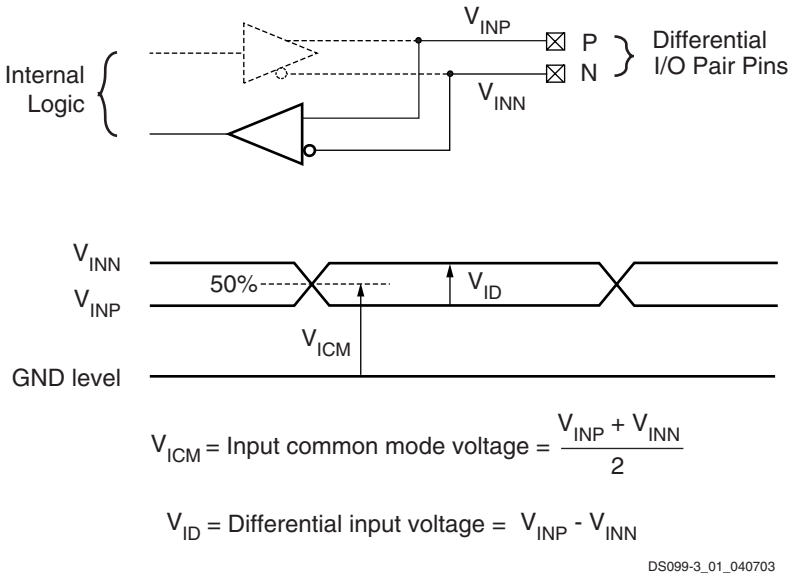



*Figure 1:* **Differential Input Voltages**

*Table 9:* **Recommended Operating Conditions for User I/Os Using Differential Signal Standards**

| | $V_{CCO}$ | | | $V_{ID}$ | | | $V_{ICM}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| **Signal Standard** | **Min (V)** | **Nom (V)** | **Max (V)** | **Min (mV)** | **Nom (mV)** | **Max (mV)** | **Min (V)** | **Nom (V)** | **Max (V)** |
| LDT_25 | 2.38 | 2.50 | 2.63 | 200 | 600 | 1000 | 0.44 | 0.60 | 0.78 |
| LVDS_25, LVDS_25_DCI | 2.38 | 2.50 | 2.63 | 100 | 350 | 600 | 0.30 | 1.25 | 2.20 |
| BLVDS_25 | 2.38 | 2.50 | 2.63 | - | 350 | - | - | 1.25 | - |
| LVDSEXT_25, LVDSEXT_25_DCI | 2.38 | 2.50 | 2.63 | 100 | 540 | 1000 | 0.30 | 1.20 | 2.20 |
| ULVDS_25 | 2.38 | 2.50 | 2.63 | 200 | 600 | 1000 | 0.44 | 0.60 | 0.78 |
| RSDS_25 | 2.38 | 2.50 | 2.63 | 100 | 200 | - | - | 1.30 | - |

**Notes:**
1. The $V_{REF}$ input is not used for any of the differential I/O standards.
2. Of the parameters shown, only $V_{CCO}$ represents a voltage measured with respect to GND. The remaining parameters are differential measurements. See Figure 1 for parameter definitions.

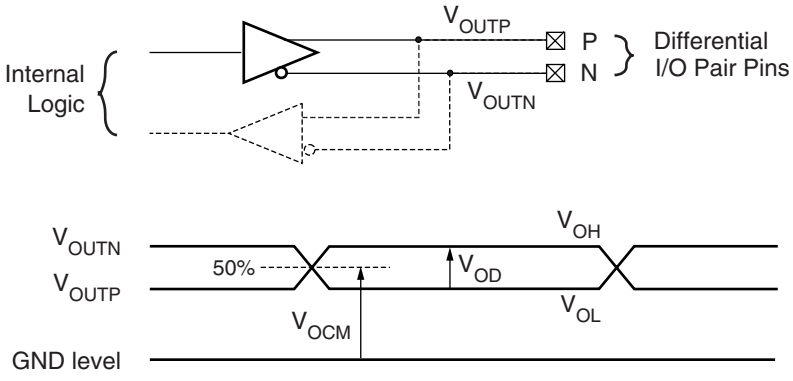



*Figure 2:* **Differential Output Voltages**

*Table 10:* **DC Characteristics of User I/Os Using Differential Signal Standards**

| Signal Standard | $V_{OD}$ | | | $\Delta V_{OD}$ | | $V_{OCM}$ | | | $\Delta V_{OCM}$ | | $V_{OH}$ | | $V_{OL}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Min (mV) | Typ (mV) | Max (mV) | Min (mV) | Max (mV) | Min (V) | Typ (V) | Max (V) | Min (mV) | Max (mV) | Typ (V) | Max (V) | Min (V) | Typ (V) |
| LDT_25 | 430 | 600 | 670 | –15 | 15 | 0.495 | 0.600 | 0.715 | -15 | 15 | - | - | - | - |
| LVDS_25, LVDS_25_DCI | 250 | 325 | 400 | - | - | 1.125 | 1.20 | 1.375 | - | - | - | 1.475 | 0.925 | - |
| BLVDS_25 | 250 | 350 | 450 | - | - | - | 1.20 | - | - | - | - | - | - | - |
| LVDSEXT_25, LVDSEXT_25_DCI | 330 | 540 | 700 | - | - | 1.125 | 1.20 | 1.375 | - | - | - | 1.700 | 0.705 | - |
| ULVDS_25 | 430 | 600 | 670 | - | - | 0.495 | 0.600 | 0.715 | - | - | - | - | - | - |
| RSDS_25 | 100 | 325 | 400 | - | - | 1.1 | 1.2 | 1.5 | - | - | - | - | - | - |

**Notes:**
1. Of the parameters shown, only $V_{OH}$, $V_{OL}$, and $V_{OCM}$ represent voltages measured with respect to GND. The remaining parameters are differential measurements. See Figure 2 for parameter definitions.
2. Output voltage measurements for all differential standards are made with a termination resistor ($R_T$) of 100Ω across the N and P pins of the differential signal pair.
3. The numbers in this table are guaranteed over the conditions set forth in Table 4 and Table 9.

## Switching Characteristics

All Spartan-3 devices are available in two speed grades: –4 and –5. Switching characteristics in this document may be designated as Advance, Preliminary, or Production. Each category is defined as follows:

**Advance**: These speed files are based on simulations only and are typically available soon after establishing FPGA specifications. Although speed grades with this designation are considered relatively stable and conservative, some under-reporting might still occur.

**Preliminary**: These speed files are based on complete ES (engineering sample) silicon characterization. Devices and speed grades with this designation are intended to give a better indication of the expected performance of production silicon. The probability of under-reporting preliminary delays is greatly reduced compared to Advance data.

**Production**: These speed files are released once enough production silicon of a particular device family member has been characterized to provide full correlation between speed files and devices over numerous production lots. There is no under-reporting of delays, and customers receive formal notification of any subsequent changes. Typically, the slowest speed grades transition to Production before faster speed grades.

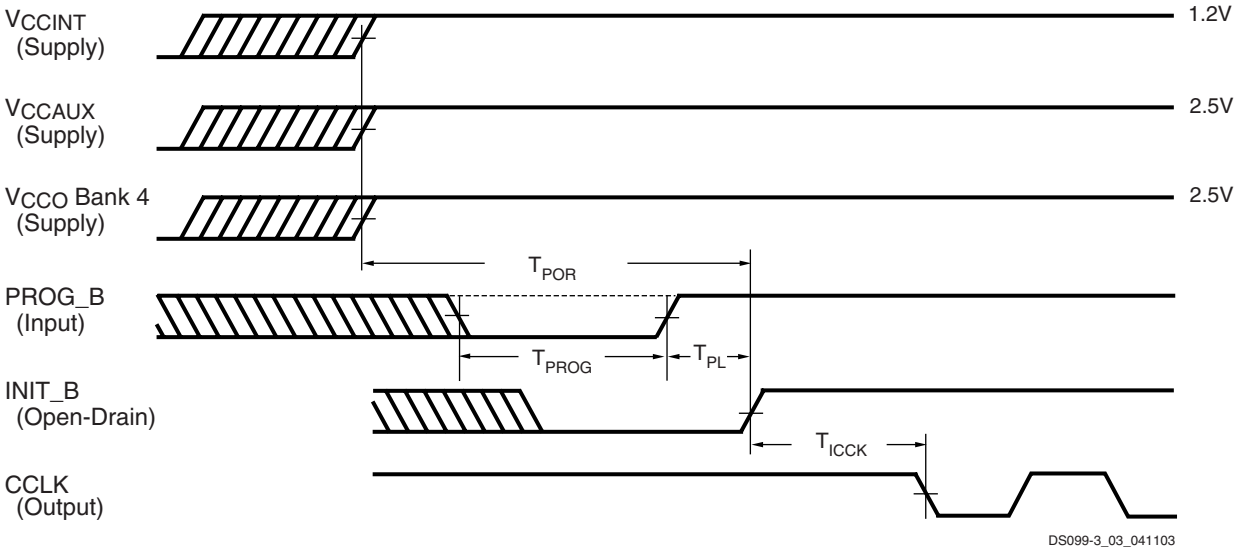
All specifications are representative of worst-case supply voltage and junction temperature conditions. Unless otherwise noted, values apply to all Spartan-3 devices.

Internal timing parameters are derived from measuring internal test patterns. Timing parameters and their representative values are selected for inclusion below either because they are important as general design requirements or they indicate fundamental device performance characteristics. For more complete, more precise, and worst-case guaranteed data, use the values reported by the Xilinx static timing analyzer (TRACE in the Xilinx development software) and back-annotate to the simulation net list.

*Table 11:* **DLL Timing**

| Symbol | Description | Frequency Mode | Speed Grade | | | | Units |
|---|---|---|---|---|---|---|---|
| | | | -5 | | -4 | | |
| | | | Min | Max | Min | Max | |
| **Clock Outputs** | | | | | | | |
| $F_{1XCO}$ | Frequency at the CLK0 and CLK180 pins | High | | | 48 | 326 | MHz |
| | | Low | | | 25 | 180 | MHz |
| **Clock Inputs** | | | | | | | |
| $F_{CLKIN}$ | Frequency at the CLKIN pin | High | | | 48 | 326 | MHz |
| | | Low | | | 25 | 180 | MHz |
| $T_{CLKINJ}$ | Allowable cycle-to-cycle jitter at the CLKIN pin | High | | | –100 | +100 | ps |
| | | Low | | | | | ps |

**Notes:**
1. For up-to-date information on DCM timing, see **http://www.xilinx.com/bvdocs/publications/ds099-3.pdf**.



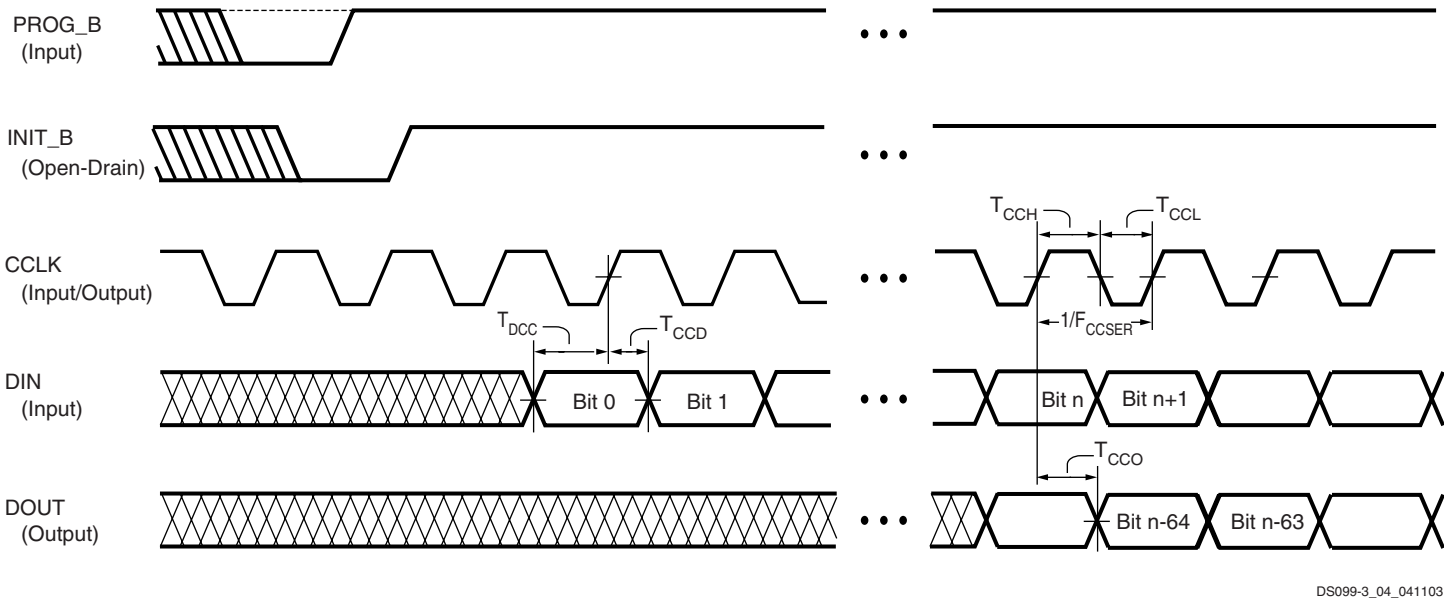

**Notes:**
1. The $V_{CCINT}$, $V_{CCAUX}$, and $V_{CCO}$ supplies may be applied in any order.
2. The Low-going pulse on PROG_B is optional after power-on but necessary for reconfiguration without a power cycle.
3. The rising edge of INIT_B samples the voltage levels applied to the mode pins (M0 - M2).

*Figure 3:* **Waveforms for Power-On and the Beginning of Configuration**

*Table 12:* **Power-On Timing and the Beginning of Configuration**

| Symbol | Description | Device | All Speed Grades | | Units |
|---|---|---|---|---|---|
| | | | Min | Max | |
| $T_{POR}$ | The time from the application of $V_{CCINT}$, $V_{CCAUX}$, and $V_{CCO}$ Bank 4 supply voltages (whichever occurs last) to the rising transition of the INIT_B pin(1) | XC3S50 | - | 5 | ms |
| | | XC3S200 | - | 5 | ms |
| | | XC3S400 | - | 5 | ms |
| | | XC3S1000 | - | 5 | ms |
| | | XC3S1500 | - | 7 | ms |
| | | XC3S2000 | - | 7 | ms |
| | | XC3S4000 | - | 7 | ms |
| | | XC3S5000 | - | 7 | ms |
| $T_{PROG}$ | The width of the low-going pulse on the PROG_B pin | All | 0.3 | - | µs |
| $T_{PL}$ | The time from the rising edge of the PROG_B pin to the rising transition on the INIT_B pin(1) | XC3S50 | - | 2 | ms |
| | | XC3S200 | - | 2 | ms |
| | | XC3S400 | - | 2 | ms |
| | | XC3S1000 | - | 2 | ms |
| | | XC3S1500 | - | 3 | ms |
| | | XC3S2000 | - | 3 | ms |
| | | XC3S4000 | - | 3 | ms |
| | | XC3S5000 | - | 3 | ms |
| $T_{ICCK}$ | The time from the rising edge of the INIT_B pin to the generation of the configuration clock signal at the CCLK output pin(2) | All | 0.5 | 4.0 | µs |

**Notes:**
1. Power-on reset and the clearing of configuration memory occurs during this period.
2. This specification applies only for the Master Serial and Master Parallel modes.

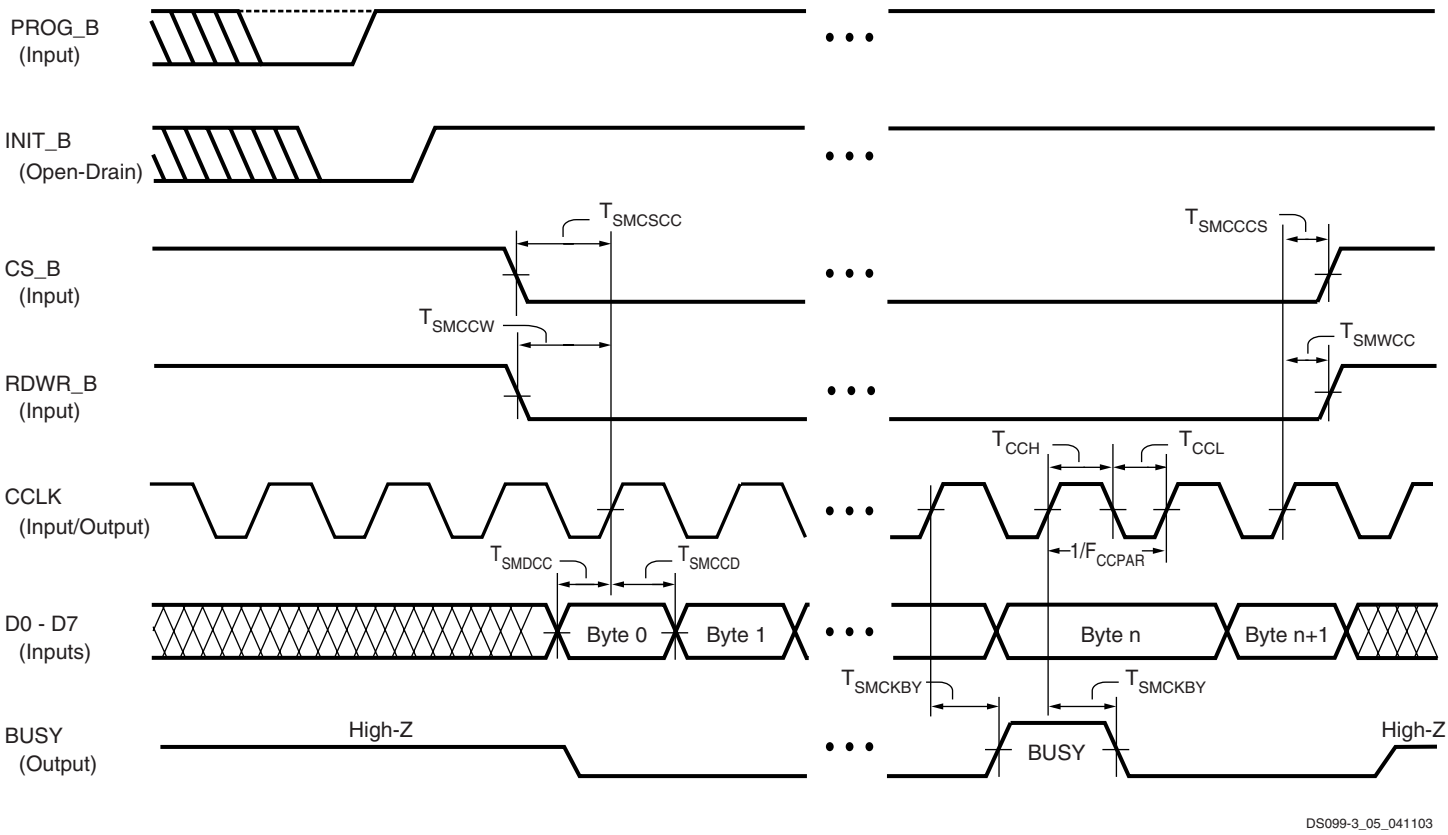



**Notes:**
1. The CS_B, WRITE_B, and BUSY signals are not used in the serial modes. Keep the CS_B and WRITE_B inputs inactive (i.e., both pins High).

*Figure 4:* **Waveforms for Master and Slave Serial Configuration**

*Table 13:* **Timing for the Master and Slave Serial Configuration Modes**

| Symbol | Description | Slave/Master | All Speed Grades | | Units |
|---|---|---|---|---|---|
| | | | Min | Max | |
| **Setup Times** | | | | | |
| $T_{DCC}$ | The time from the setup of data at the DIN pin to the rising transition at the CCLK pin | Both | - | 5.0 | ns |
| **Hold Times** | | | | | |
| $T_{CCD}$ | The time from the rising transition at the CCLK pin to the point when data is last held at the DIN pin | Both | - | 0 | ns |
| **Clock-to-Output Times** | | | | | |
| $T_{CCO}$ | The time from the rising transition on the CCLK pin to data appearing at the DOUT pin | Both | - | 12.0 | ns |
| **Clock Timing** | | | | | |
| $T_{CCH}$ | The High pulse width at the CCLK input pin | Slave | 5.0 | - | ns |
| $T_{CCL}$ | The Low pulse width at the CCLK input pin | | 5.0 | - | ns |
| $F_{CCSER}$ | Frequency of the clock signal at the CCLK input pin | | - | 66 | MHz |
| $\Delta F_{CCSER}$ | Variation from the generated CCLK frequency set using the ConfigRate BitGen option | Master | –50% | +50% | - |




**Notes:**
1. In a given CCLK cycle, when RDWR_B transitions High or Low while holding CS_B Low, the next rising edge on the CCLK pin will abort configuration.

*Figure 5:* **Waveforms for Master and Slave Parallel Configuration**

*Table 14:* **Timing for the Master and Slave Parallel Configuration Modes**

| Symbol | Description | Slave/Master | All Speed Grades | | Units |
|---|---|---|---|---|---|
| | | | Min | Max | |
| **Setup Times** | | | | | |
| $T_{SMDCC}$ | The time from the setup of data at the D0-D7 pins to the rising transition at the CCLK pin | Both | 5.0 | - | ns |
| $T_{SMCSCC}$ | The time from the setup of a logic level at the CS_B pin to the rising transition at the CCLK pin | | 7.0 | - | ns |
| $T_{SMCCW}$ | The time from the setup of a logic level at the RDWR_B pin to the rising transition at the CCLK pin(1) | | 7.0 | - | ns |
| **Hold Times** | | | | | |
| $T_{SMCCD}$ | The time from the rising transition at the CCLK pin to the point when data is last held at the D0-D7 pins | Both | 0 | - | ns |
| $T_{SMCCCS}$ | The time from the rising transition at the CCLK pin to the point when a logic level is last held at the CS_B pin | | 0 | - | ns |
| $T_{SMWCC}$ | The time from the rising transition at the CCLK pin(1) to the point when a logic level is last held at the RDWR_B pin(1) | | 0 | - | ns |

*Table 14:* **Timing for the Master and Slave Parallel Configuration Modes *(Continued)***

| Symbol | Description | | Slave/Master | All Speed Grades | | Units |
|---|---|---|---|---|---|---|
| | | | | Min | Max | |
| **Clock-to-Output Times** | | | | | | |
| $T_{SMCKBY}$ | The time from the rising transition on the CCLK pin to a signal transition at the BUSY pin | | Slave | - | 12.0 | ns |
| **Clock Timing** | | | | | | |
| $T_{CCH}$ | The High pulse width at the CCLK input pin | | Slave | 5 | - | ns |
| $T_{CCL}$ | The Low pulse width at the CCLK input pin | | | 5 | - | ns |
| $F_{CCPAR}$ | Frequency of the clock signal at the CCLK input pin | Not using the BUSY pin(2) | | - | 66 | MHz |
| | | Using the BUSY pin | | - | 100 | MHz |
| $\Delta F_{CCPAR}$ | Variation from the generated CCLK frequency set using the BitGen option ConfigRate | | Master | –50% | +50% | - |

**Notes:**
1. RDWR_B is synchronized to CCLK for the purpose of performing the Abort operation. The same pin asynchronously controls the driver impedance of the D0 - D7 pins. To avoid contention when writing configuration data to the D0 - D7 bus, do not bring RDWR_B High when CS_B is Low.
2. In the Slave Parallel mode, it is necessary to use the BUSY pin when the CCLK frequency exceeds this maximum specification.

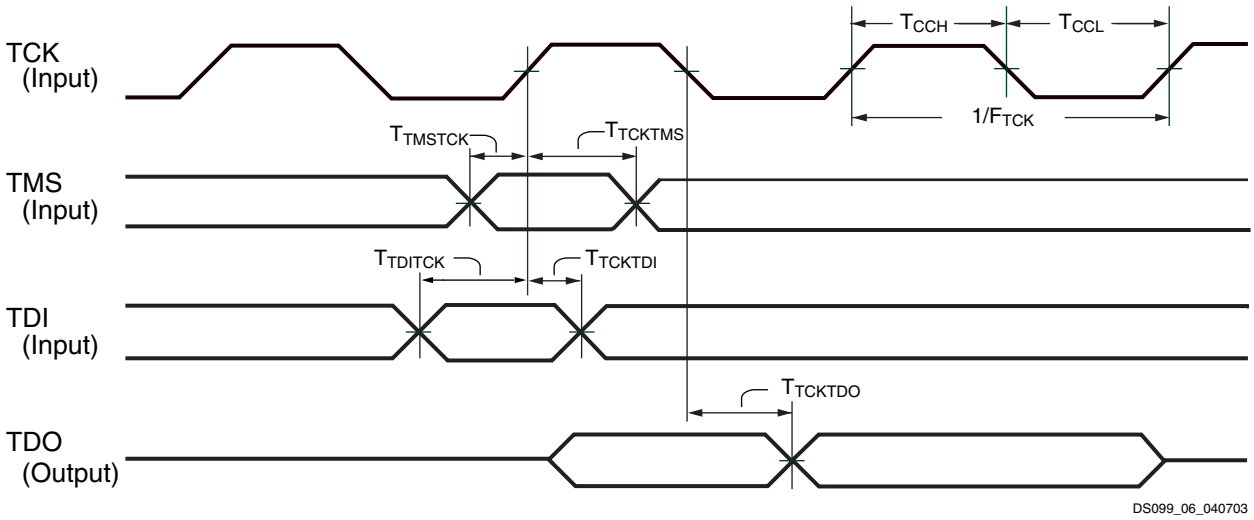



*Figure 6:* **JTAG Waveforms**

*Table 15:* **Timing for the JTAG Port**

| Symbol | Description | All Speed Grades | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| **Setup Times** | | | | |
| $T_{TDITCK}$ | The time from the setup of data at the TDI pin to the rising transition at the TCK pin | 4.0 | - | ns |
| $T_{TMSTCK}$ | The time from the setup of a logic level at the TMS pin to the rising transition at the TCK pin | 4.0 | - | ns |
| **Hold Times** | | | | |
| $T_{TCKTDI}$ | The time from the rising transition at the TCK pin to the point when data is last held at the TDI pin | 0 | - | ns |

*Table 15:* **Timing for the JTAG Port *(Continued)***

| Symbol | Description | All Speed Grades | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| $T_{TCKTMS}$ | The time from the rising transition at the TCK pin to the point when a logic level is last held at the TMS pin | 0 | - | ns |
| **Clock-to-Output Times** | | | | |
| $T_{TCKTDO}$ | The time from the falling transition on the TCK pin to data appearing at the TDO pin | - | 11.0 | ns |
| **Clock Timing** | | | | |
| $T_{CCH}$ | The High pulse width at the TCK pin | 5.0 | - | ns |
| $T_{CCL}$ | The Low pulse width at the TCK pin | 5.0 | - | ns |
| $F_{TCK}$ | Frequency of the clock signal at the TCK pin | - | 33 | MHz |

## Revision History

| Date | Version No. | Description |
|---|---|---|
| 04/11/03 | 1.0 | Initial Xilinx release. |
| 07/11/03 | 1.1 | Extended Absolute Maximum Rating for junction temperature in Table 1. Added numbers for typical quiescent supply current (Table 6) and DLL timing (Table 11) . |

## The Spartan-3 Family Data Sheet

**DS099-1**, ***Spartan-3 1.2V FPGA Family: Introduction and Ordering Information*** (Module 1)

**DS099-2, *Spartan-3 1.2V FPGA Family: Functional Description*** (Module 2)

**DS099-3, *Spartan-3 1.2V FPGA Family: DC and Switching Characteristics*** (Module 3)

**DS099-4**, ***Spartan-3 1.2V FPGA Family: Pinout Tables*** (Module 4)

## Introduction

This data sheet module describes the various pins on a Spartan™-3 FPGA and how they connect to the supported component packages.

- The **Pin Types** section categorizes all of the FPGA pins by their function type.
- The **Pin Definitions** section provides a top-level description for each pin on the device.
- The **Detailed, Functional Pin Descriptions** section offers significantly more detail about each pin, especially for the dual- or special-function pins used during device configuration.
- Some pins have associated optional behavior, controlled by settings in the configuration bitstream. These options are described in the **Bitstream Options** section.
- The **Package Overview** section describes the various packaging options available for Spartan-3 FPGAs. Detailed pin list tables and footprint diagrams are provided for each package solution.

## Pin Descriptions

### Pin Types

A majority of the pins on a Spartan-3 FPGA are general-purpose, user-defined I/O pins. There are, however, up to 12 different functional types of pins on Spartan-3 packages, as outlined in Table 1. In the package footprint drawings that follow, the individual pins are color-coded according to pin type as in the table.

*Table 1:* **Types of Pins on Spartan-3 FPGAs**

| Type/ Color Code | Description | Pin Name(s) in Type |
|---|---|---|
| I/O | Unrestricted, general-purpose user-I/O pin. Most pins can be paired together to form differential I/Os. | IO,<br>IO_Lxxy_# |
| DUAL | Dual-purpose pin used in some configuration modes during the configuration process and then usually available as a user I/O after configuration. If the pin is not used during configuration, this pin behaves as an I/O-type pin. There are 12 dual-purpose configuration pins on every package. | IO_Lxxy_#/DIN/D0, IO_Lxxy_#/D1,<br>IO_Lxxy_#/D2, IO_Lxxy_#/D3,<br>IO_Lxxy_#/D4, IO_Lxxy_#/D5,<br>IO_Lxxy_#/D6, IO_Lxxy_#/D7,<br>IO_Lxxy_#/CS_B, IO_Lxxy_#/RDWR_B,<br>IO_Lxxy_#/BUSY/DOUT,<br>IO_Lxxy_#/INIT_B |
| CONFIG | Dedicated configuration pin. Not available as a user-I/O pin. Every package has seven dedicated configuration pins. These pins are powered by VCCAUX. | CCLK, DONE, M2, M1, M0, PROG_B, HSWAP_EN |
| JTAG | Dedicated JTAG pin. Not available as a user-I/O pin. Every package has four dedicated JTAG pins. These pins are powered by VCCAUX. | TDI, TMS, TCK, TDO |
| DCI | Dual-purpose pin that is either a user-I/O pin or used to calibrate output buffer impedance for a specific bank using Digital Controlled Impedance (DCI). There are two DCI pins per I/O bank. | IO/VRN_#<br>IO_Lxxy_#/VRN_#<br>IO/VRP_#<br>IO_Lxxy_#/VRP_# |
| VREF | Dual-purpose pin that is either a user-I/O pin or, along with all other VREF pins in the same bank, provides a reference voltage input for certain I/O standamrds. If used for a reference voltage within a bank, all VREF pins within the bank must be connected. | IO/VREF_#<br>IO_Lxxy_#/VREF_# |

*Table 1:* **Types of Pins on Spartan-3 FPGAs *(Continued)***

| Type/ Color Code | Description | Pin Name(s) in Type |
|---|---|---|
| GND | Dedicated ground pin. The number of GND pins depends on the package used. All must be connected. | GND |
| VCCAUX | Dedicated auxiliary power supply pin. The number of VCCAUX pins depends on the package used. All must be connected to +2.5V. | VCCAUX |
| VCCINT | Dedicated internal core logic power supply pin. The number of VCCINT pins depends on the package used. All must be connected to +1.2V. | VCCINT |
| VCCO | Dedicated I/O bank, output buffer power supply pin. Along with other VCCO pins in the same bank, this pin supplies power to the output buffers within the I/O bank and sets the input threshold voltage for some I/O standards. | VCCO_#<br>***TQ144 Package Only:***<br>VCCO_LEFT, VCCO_TOP, VCCO_RIGHT, VCCO_BOTTOM |
| GCLK | Dual-purpose pin that is either a user-I/O pin or an input to a specific global buffer input. Every package has eight dedicated GCLK pins. | IO_Lxxy_#/GCLK0, IO_Lxxy_#/GCLK1, IO_Lxxy_#/GCLK2, IO_Lxxy_#/GCLK3, IO_Lxxy_#/GCLK4, IO_Lxxy_#/GCLK5, IO_Lxxy_#/GCLK6, IO_Lxxy_#/GCLK7 |
| N.C. | This package pin is not connected in this specific device/package combination but may be connected in larger devices in the same package. | N.C. |

**Notes:**
1. # = I/O bank number, an integer between 0 and 7.

I/Os with Lxxy_# are part of a differential output pair. 'L' indicates differential output capability. The "xx" field is a two-digit integer, unique to each bank that identifies a differential pin-pair. The 'y' field is either 'P' for the true signal or 'N' for the inverted signal in the differential pair. The '#' field is the I/O bank number.

## Pin Definitions

Table 2 provides a brief description of each pin listed in the Spartan-3 pinout tables and package footprint diagrams. Pins are categorized by their pin type, as listed in Table 1. See **Detailed, Functional Pin Descriptions** for more information.

*Table 2:* **Spartan-3 Pin Definitions**

| Pin Name | Direction | Description |
|---|---|---|
| I/O: General-purpose I/O pins | | |
| I/O | User-defined as input, output, bidirectional, three-state output, open-drain output, open-source output | **User I/O:**<br>Unrestricted single-ended user-I/O pin. Supports all I/O standards except the differential standards. |
| I/O_Lxxy_# | User-defined as input, output, bidirectional, three-state output, open-drain output, open-source output | **User I/O, Half of Differential Pair:**<br>Unrestricted single-ended user-I/O pin or half of a differential pair. Supports all I/O standards including the differential standards. |
| **DUAL:** Dual-purpose configuration pins | | |
| IO_Lxxy_#/DIN/D0, IO_Lxxy_#/D1, IO_Lxxy_#/D2, IO_Lxxy_#/D3, IO_Lxxy_#/D4, IO_Lxxy_#/D5, IO_Lxxy_#/D6, IO_Lxxy_#/D7 | Input during configuration<br>Possible bidirectional I/O after configuration if SelectMap port is retained.<br>Otherwise, user I/O after configuration | **Configuration Data Port:**<br>In Parallel (SelectMAP) modes, D0-D7 are byte-wide configuration data pins. These pins become user I/Os after configuration unless the SelectMAP port is retained via the Persist bitstream option.<br>In Serial modes, DIN (D0) serves as the single configuration data input. This pin becomes a user I/O after configuration unless retained by the Persist bitstream option. |
| IO_Lxxy_#/CS_B | Input during Parallel mode configuration<br>Possible input after configuration if SelectMap port is retained.<br>Otherwise, user I/O after configuration | **Chip Select for Parallel Mode Configuration:**<br>In Parallel (SelectMAP) modes, this is the active-Low Chip Select signal. This pin becomes a user I/O after configuration unless the SelectMAP port is retained via the Persist bitstream option. |
| IO_Lxxy_#/RDWR_B | Input during Parallel mode configuration<br>Possible input after configuration if SelectMap port is retained.<br>Otherwise, user I/O after configuration | **Read/Write Control for Parallel Mode Configuration:**<br>In Parallel (SelectMAP) modes, this is the active-Low Write Enable, active-High Read Enable signal. This pin becomes a user I/O after configuration unless the SelectMAP port is retained via the Persist bitstream option. |
| IO_Lxxy_#/ BUSY/DOUT | Output during configuration<br>Possible output after configuration if SelectMap port is retained.<br>Otherwise, user I/O after configuration | **Configuration Data Rate Control for Parallel Mode, Serial Data Output for Serial Mode:**<br>In Parallel (SelectMAP) modes, BUSY throttles the rate at which configuration data is loaded. This pin becomes a user I/O after configuration unless the SelectMAP port is retained via the Persist bitstream option.<br>In Serial modes, DOUT provides preamble and configuration data to downstream devices in a multi-FPGA daisy-chain. This pin becomes a user I/O after configuration. |

*Table 2:* **Spartan-3 Pin Definitions *(Continued)***

| Pin Name | Direction | Description |
|---|---|---|
| IO_Lxxy_#/INIT_B | Bidirectional (open-drain) during configuration<br>User I/O after configuration | **Initializing Configuration Memory/Detected Configuration Error:**<br>When Low, this pin indicates that configuration memory is being cleared. When held Low, this pin delays the start of configuration. After this pin is released or configuration memory is cleared, the pin goes High. During configuration, a Low on this output indicates that a configuration data error occurred. This pin becomes a user I/O after configuration. |
| **DCI:** Digitally Controlled Impedance reference resistor input pins | | |
| IO_Lxxy_#/VRN_# or IO/VRN_# | Input when using DCI<br>Otherwise, same as I/O | **DCI Reference Resistor for NMOS I/O Transistor (per bank):**<br>If using DCI, a 1% precision impedance-matching resistor is connected between this pin and the VCCO supply for this bank. Otherwise, this pin is a user I/O. |
| IO_Lxxy_#/VRP_# or IO/VRP_# | Input when using DCI<br>Otherwise, same as I/O | **DCI Reference Resistor for PMOS I/O Transistor (per bank):**<br>If using DCI, a 1% precision impedance-matching resistor is connected between this pin and the ground supply. Otherwise, this pin is a user I/O. |
| **GCLK:** Global clock buffer inputs | | |
| IO_Lxxy_#/GCLK0, IO_Lxxy_#/GCLK1, IO_Lxxy_#/GCLK2, IO_Lxxy_#/GCLK3, IO_Lxxy_#/GCLK4, IO_Lxxy_#/GCLK5, IO_Lxxy_#/GCLK6, IO_Lxxy_#/GCLK7 | Input if connected to global clock buffers<br>Otherwise, same as I/O | **Global Buffer Input:**<br>Direct input to a low-skew global clock buffer. If not connected to a global clock buffer, this pin is a user I/O. |
| **VREF:** I/O bank input reference voltage pins | | |
| IO_Lxxy_#/VREF_# or IO/VREF_# | Voltage supply input when VREF pins are used within a bank.<br>Otherwise, same as I/O | **Input Buffer Reference Voltage for Special I/O Standards (per bank):**<br>If required to support special I/O standards, all the VREF pins within a bank connect to a input threshold voltage source.<br>If not used as input reference voltage pins, these pins are available as individual user-I/O pins. |
| **CONFIG:** Dedicated configuration pins | | |
| CCLK | Input in Slave configuration modes<br>Output in Master configuration modes | **Configuration Clock:**<br>The configuration clock signal synchronizes configuration data. |
| PROG_B | Input | **Program/Configure Device:**<br>Active Low asynchronous reset to configuration logic. Asserting PROG_B Low for an extended period delays the configuration process. This pin has an internal weak pull-up resistor during configuration. |

*Table 2:* **Spartan-3 Pin Definitions *(Continued)***

| Pin Name | Direction | Description |
|---|---|---|
| DONE | Bidirectional with open-drain or totem-pole Output | **Configuration Done, Delay Start-up Sequence:**<br>A Low-to-High output transition on this bidirectional pin signals the end of the configuration process.<br>The FPGA produces a Low-to-High transition on this pin to indicate that the configuration process is complete. The DriveDone bitstream generation option defines whether this pin functions as a totem-pole output that actively drives High or as an open-drain output. An open-drain output requires a pull-up resistor to produce a High logic level. The open-drain option permits the DONE lines of multiple FPGAs to be tied together, so that the common node transitions High only after all of the FPGAs have completed configuration. Externally holding the open-drain output Low delays the start-up sequence, which marks the transition to user mode. |
| M0, M1, M2 | Input | **Configuration Mode Selection:**<br>These inputs select the configuration mode. The logic levels applied to the mode pins are sampled on the rising edge of INIT_B. See Table 7. |
| HSWAP_EN | Input | **Disable Weak Pull-up Resistors During Configuration:**<br>A Low on this pin enables weak pull-up resistors on all pins that are not actively involved in the configuration process. A High value disables all pull-ups, allowing the non-configuration pins to float. |
| **JTAG:** JTAG interface pins | | |
| TCK | Input | **JTAG Test Clock:**<br>The TCK clock signal synchronizes all JTAG port operations. |
| TDI | Input | **JTAG Test Data Input:**<br>TDI is the serial data input for all JTAG instruction and data registers. |
| TMS | Input | **JTAG Test Mode Select:**<br>The serial TMS input controls the operation of the JTAG port. |
| TDO | Output | **JTAG Test Data Output:**<br>TDO is the serial data output for all JTAG instruction and data registers. |
| **VCCO:** I/O bank output voltage supply pins | | |
| VCCO_# | Supply | **Power Supply for Output Buffer Drivers (per bank):**<br>These pins power the output drivers within a specific I/O bank. |
| **VCCAUX:** Auxiliary voltage supply pins | | |
| VCCAUX | Supply | **Power Supply for Auxiliary Circuits:**<br>+2.5V power pins for auxiliary circuits, including the Digital Clock Managers (DCMs), the dedicated configuration pins (CONFIG), and the dedicated JTAG pins. All VCCAUX pins must be connected. |
| **VCCINT:** Internal core voltage supply pins | | |
| VCCINT | Supply | **Power Supply for Internal Core Logic:**<br>+1.2V power pins for the internal logic. All pins must be connected. |

*Table 2:* **Spartan-3 Pin Definitions *(Continued)***

| Pin Name | Direction | Description |
|---|---|---|
| **GND:** Ground supply pins | | |
| GND | Supply | **Ground:** Ground pins, which are connected to the power supply's return path. All pins must be connected. |
| **N.C.:** Unconnected package pins | | |
| N.C. | | **Unconnected Package Pin:** These package pins are unconnected. |

**Notes:**
1. All unused inputs and bidirectional pins must be tied either High or Low. For unused enable inputs, apply the level that disables the associated function. One common approach is to activate internal pull-up or pull-down resistors. An alternative approach is to externally connect the pin to either VCCO or GND.
2. All outputs are of the totem-pole type — i.e., they can drive High as well as Low logic levels — except for the cases where "Open Drain" is indicated. The latter can only drive a Low logic level and require a pull-up resistor to produce a High logic level.

# Detailed, Functional Pin Descriptions

## I/O Type: Unrestricted, General-purpose I/O Pins

After configuration, I/O-type pins are inputs, outputs, bidirectional I/O, three-state outputs, open-drain outputs, or open-source outputs, as defined in the application

Pins labeled "IO" support all SelectIO™ signal standards except differential standards. A given device at most only has a few of these pins.

A majority of the general-purpose I/O pins are labeled in the format "IO_Lxxy_#". These pins support all SelectIO signal standards, including the differential standards such as LVDS, ULVDS, BLVDS, RSDS, or LDT.

For additional information, see the **"IOB" section under Functional Description (Module 2 of the Spartan-3 data sheet)**.

### *Differential Pair Labeling*

A pin supports differential standards if the pin is labeled in the format "Lxxy_#". The pin name suffix has the following significance. Figure 1 provides a specific example showing a differential input to and a differential output from Bank 2.

- 'L' indicates differential capability.
- "xx" is a two-digit integer, unique for each bank, that identifies a differential pin-pair.
- 'y' is replaced by 'P' for the true signal or 'N' for the inverted. These two pins form one differential pin-pair.
- '#' is an integer, 0 through 7, indicating the associated I/O bank.

If unused, these pins are in a high impedance state. The Bitstream generator option UnusedPin enables a weak pull-up or pull-down resistor on all unused I/O pins.

### *Behavior from Power-On through End of Configuration*

During the configuration process, all pins that are not actively involved in the configuration process are in a high-impedance state. The HSWAP_EN input determines whether or not weak pull-up resistors are enabled during configuration. HSWAP_EN = 0 enables the weak pull-up resistors. HSWAP_EN = 1 disables the pull-up resistors allowing the pins to float, which is the desired state for hot-swap applications.

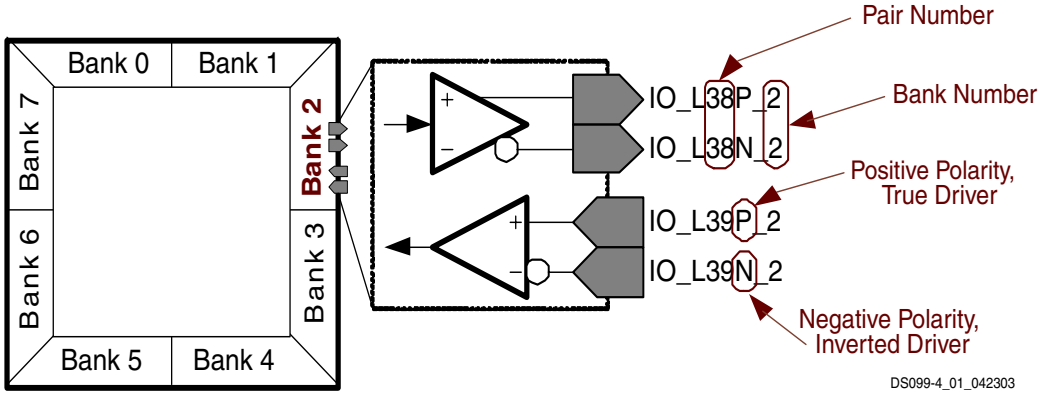



*Figure 1:* **Differential Pair Labelling**

## DUAL Type: Dual-Purpose Configuration and I/O Pins

These pins serve dual purposes. The user-I/O pins are temporarily borrowed during the configuration process to load configuration data into the FPGA. After configuration, these pins are then usually available as a user I/O in the application. If a pin is not applicable to the specific configuration mode—controlled by the mode select pins M2, M1, and M0—then the pin behaves as an I/O-type pin.

There are 12 dual-purpose configuration pins on every package, six of which are part of I/O Bank 4, the other six part of I/O Bank 5. Only a few of the pins in Bank 4 are used in the Serial configuration modes.

See **"Configuration" under Functional Description (Module 2 of the Spartan-3 data sheet)**.

See "**Pin Behavior During Configuration**, page 85".

### *Serial Configuration Modes*

This section describes the dual-purpose pins used during either Master or Slave Serial mode. See Table 7 for Mode Select pin settings required for Serial modes. All such pins are in Bank 4 and powered by VCCO_4.

In both the Master and Slave Serial modes, DIN is the serial configuration data input. The D1-D7 inputs are unused in serial mode and behave like general-purpose I/O pins.

In all the cases, the configuration data is synchronized to the rising edge of the CCLK clock signal.

The DIN, DOUT, and INIT_B pins can be retained in the application to support reconfiguration by setting the Persist bitstream generation option. However, the serial modes do not support device readback.

*Table 3:* **Dual-Purpose Pins Used in Master or Slave Serial Mode**

| Pin Name | Direction | Description |
|---|---|---|
| DIN | Input | **Serial Data Input:**<br>During the Master or Slave Serial configuration modes, DIN is the serial configuration data input, and all data is synchronized to the rising CCLK edge. After configuration, this pin is available as a user I/O.<br>This signal is located in Bank 4 and its output voltage determined by VCCO_4.<br>The BitGen option Persist permits this pin to retain its configuration function in the User mode. |
| DOUT | Output | **Serial Data Output:**<br>In a multi-FPGA design where all the FPGAs use serial mode, connect the DOUT output of one FPGA—in either Master or Slave Serial mode—to the DIN input of the next FPGA—in Slave Serial mode—so that configuration data passes from one to the next, in daisy-chain fashion. This "daisy chain" permits sequential configuration of multiple FPGAs.<br>This signal is located in Bank 4 and its output voltage determined by VCCO_4.<br>The BitGen option Persist permits this pin to retain its configuration function in the User mode. |
| INIT_B | Bidirectional (open-drain) | **Initializing Configuration Memory/Configuration Error:**<br>Just after power is applied, the FPGA produces a Low-to-High transition on this pin indicating that initialization (*i.e.*, clearing) of the configuration memory has finished. Before entering the User mode, this pin functions as an open-drain output, which requires a pull-up resistor in order to produce a High logic level. In a multi-FPGA design, tie (wire AND) the INIT_B pins from all FPGAs together so that the common node transitions High only after all of the FPGAs have been successfully initialized.<br>Externally holding this pin Low beyond the initialization phase delays the start of configuration. This action stalls the FPGA at the configuration step just before the mode select pins are sampled.<br>During configuration, the FPGA indicates the occurrence of a data (i.e., CRC) error by asserting INIT_B Low.<br>This signal is located in Bank 4 and its output voltage determined by VCCO_4.<br>The BitGen option Persist permits this pin to retain its configuration function in the User mode. |

| | I/O Bank 4 (VCCO_4) High Nibble | | | | I/O Bank 5 (VCCO_5) Low Nibble | | | |
|---|---|---|---|---|---|---|---|---|
| **Configuration Data Byte** | **D0** | **D1** | **D2** | **D3** | **D4** | **D5** | **D6** | **D7** |
| 0xA5 = | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |

*Figure 2:* **Configuration Data Byte Mapping to D0-D7 Bits**

### *Parallel Configuration Modes (SelectMAP)*

This section describes the dual-purpose configuration pins used during the Master and Slave Parallel configuration modes, sometimes also called the SelectMAP modes. In both Master and Slave Parallel configuration modes, D0-D7 form the byte-wide configuration data input. See Table 7 for Mode Select pin settings required for Parallel modes.

As shown in Figure 2, D0 is the most-significant bit while D7 is the least-significant bit. Bits D0-D3 form the high nibble of the byte and bits D4-D7 form the low nibble.

In the Parallel configuration modes, both the VCCO_4 and VCCO_5 voltage supplies are required and must both equal the voltage of the attached configuration device, typically either 2.5V or 3.3V.

Assert Low both the chip-select pin, CS_B, and the read/write control pin, RDWR_B, to write the configuration data byte presented on the D0-D7 pins to the FPGA on a rising-edge of the configuration clock, CCLK. The order of CS_B and RDWR_B does not matter, although RDWR_B must be asserted throughout the configuration process. If RDWR_B is de-asserted during configuration, the FPGA aborts the configuration operation.

After configuration, these pins are available as general-purpose user I/O. However, the SelectMAP configuration interface is optionally available for debugging and dynamic reconfiguration. To use these SelectMAP pins after configuration, set the Persist bitstream generation option.

The Readback debugging option, for example, requires the Persist bitstream generation option. During Readback mode, assert CS_B Low, along with RDWR_B High, to read a configuration data byte from the FPGA to the D0-D7 bus on a rising CCLK edge. During Readback mode, D0-D7 are output pins.

In all the cases, the configuration data and control signals are synchronized to the rising edge of the CCLK clock signal.

*Table 4:* **Dual-Purpose Configuration Pins for Parallel (SelectMAP) Configuration Modes**

| Pin Name | Direction | Description |
|---|---|---|
| D0, D1, D2, D3 | Input during configuration<br><br>Output during readback | **Configuration Data Port (high nibble):**<br>Collectively, the D0-D7 pins are the byte-wide configuration data port for the Parallel (SelectMAP) configuration modes. Configuration data is synchronized to the rising edge of CCLK clock signal.<br>The D0-D3 pins are the high nibble of the configuration data byte and located in Bank 4 and powered by VCCO_4.<br>The BitGen option Persist permits this pin to retain its configuration function in the User mode. |
| D4, D5, D6, D7 | Input during configuration<br><br>Output during readback | **Configuration Data Port (low nibble):**<br>The D4-D7 pins are the low nibble of the configuration data byte. However, these signals are located in Bank 5 and powered by VCCO_5.<br>The BitGen option Persist permits this pin to retain its configuration function in the User mode. |
| CS_B | Input | **Chip Select for Parallel Mode Configuration:**<br>Assert this pin Low, together with RDWR_B to write a configuration data byte from the D0-D7 bus to the FPGA on a rising CCLK edge.<br>During Readback, assert this pin Low, along with RDWR_B High, to read a configuration data byte from the FPGA to the D0-D7 bus on a rising CCLK edge.<br>This signal is located in Bank 5 and powered by VCCO_5.<br>The BitGen option Persist permits this pin to retain its configuration function in the User mode. |

| CS_B | Function |
|---|---|
| 0 | FPGA selected. SelectMAP inputs are valid on the next rising edge of CCLK. |
| 1 | FPGA deselected. All SelectMAP inputs are ignored. |

*Table 4:* **Dual-Purpose Configuration Pins for Parallel (SelectMAP) Configuration Modes *(Continued)***

<table>
<tr><th>Pin Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>RDWR_B</td><td>Input</td><td>Read/Write Control for Parallel Mode Configuration:<br>In Master and Slave Parallel modes, assert this pin Low together with CS_B to write a configuration data byte from the D0-D7 bus to the FPGA on a rising CCLK edge. Once asserted during configuration, RDWR_B must remain asserted until configuration is complete.<br>During Readback, assert this pin High with CS_B Low to read a configuration data byte from the FPGA to the D0-D7 bus on a rising CCLK edge.<br>This signal is located in Bank 5 and powered by VCCO_5.<br>The BitGen option Persist permits this pin to retain its configuration function in the User mode.
<table>
<tr><th>RDWR_B</th><th>Function</th></tr>
<tr><td>0</td><td>If CS_B is Low, then load (write) configuration data to the FPGA.</td></tr>
<tr><td>1</td><td>This option is valid only if the Persist bitstream option is set to Yes. If CS_B is Low, then read configuration data from the FPGA.</td></tr>
</table></td></tr>
<tr><td>BUSY</td><td>Output</td><td>Configuration Data Rate Control for Parallel Mode:<br>In the Slave and Master Parallel modes, BUSY throttles the rate at which configuration data is loaded. BUSY is only necessary if CCLK operates at greater than 50 MHz. Ignore BUSY for frequencies of 50 MHz and below.<br>When BUSY is Low, the FPGA accepts the next configuration data byte on the next rising CCLK edge for which CS_B and RDWR_B are Low. When BUSY is High, the FPGA ignores the next configuration data byte. The next configuration data value must be held or reloaded until the next rising CCLK edge when BUSY is Low. When CS_B is High, BUSY is in a high impedance state.
<table>
<tr><th>BUSY</th><th>Function</th></tr>
<tr><td>0</td><td>The FPGA is ready to accept the next configuration data byte.</td></tr>
<tr><td>1</td><td>The FPGA is busy processing the current configuration data byte and is not ready to accept the next byte.</td></tr>
<tr><td>Hi-Z</td><td>If CS_B is High, then BUSY is high impedance.</td></tr>
</table>
This signal is located in Bank 4 and its output voltage is determined by VCCO_4. The BitGen option Persist permits this pin to retain its configuration function in the User mode.</td></tr>
<tr><td>INIT_B</td><td>Bidirectional (open-drain)</td><td>Initializing Configuration Memory/Configuration Error (active-Low):<br>See description under Serial Configuration Modes, page 77.</td></tr>
</table>

### *JTAG Configuration Mode*

In the JTAG configuration mode all dual-purpose configuration pins are unused and behave exactly like user-I/O pins, as shown in Table 10. See Table 7 for Mode Select pin settings required for JTAG mode.

### *Dual-Purpose Pin I/O Standard During Configuration*

During configuration, the dual-purpose pins default to CMOS input and output levels for the associated VCCO voltage supply pins. For example, in the Parallel configuration modes, both VCCO_4 and VCCO_5 are required. If connected to +2.5V, then the associated pins conform to the LVCMOS25 I/O standard. If connected to +3.3V, then the pins drive LVCMOS output levels and accept either LVTTL or LVCMOS input levels.

### *Dual-Purpose Pin Behavior After Configuration*

After the configuration process completes, these pins, if they were borrowed during configuration, become user-I/O pins available to the application. If a dual-purpose configuration pin is not used during the configuration process—*i.e.*, the parallel configuration pins when using serial mode—then the pin behaves exactly like a general-purpose I/O. See **I/O Type: Unrestricted, General-purpose I/O Pins** section above.

## DCI: User I/O or Digitally Controlled Impedance Resistor Reference Input

These pins are individual user-I/O pins unless one of the I/O standards used in the bank requires the Digitally Controlled Impedance (DCI) feature. If DCI is used, then 1% precision resistors connected to the VRP_# and VRN_# pins match the impedance on the input or output buffers of the I/O standards that use DCI within the bank.

The '#' character in the pin name indicates the associated I/O bank and is an integer, 0 through 7.

There are two DCI pins per I/O bank, except in the TQ144 package, which does not have any DCI inputs for Bank 5.

### *VRP and VRN Impedance Resistor Reference Inputs*

The 1% precision impedance-matching resistor attached to the VRP_# pin controls the pull-up impedance of PMOS transistor in the input or output buffer. Consequently, the VRP_# pin must connect to ground. The 'P' character in "VRP" indicates that this pin controls the I/O buffer's PMOS transistor impedance. The VRP_# pin is used for both single and split termination.

The 1% precision impedance-matching resistor attached to the VRN_# pin controls the pull-down impedance of NMOS transistor in the input or output buffer. Consequently, the VRN_# pin must connect to VCCO. The 'N' character in "VRN" indicates that this pin controls the I/O buffer's NMOS transistor impedance. The VRN_# pin is only used for split termination.

Each VRN or VRP reference input requires its own resistor. A single resistor cannot be shared between VRN or VRP pins associated with different banks.

During configuration, these pins behave exactly like user-I/O pins. The associated DCI behavior is not active or valid until after configuration completes.

See **"Digitally Controlled Impedance (DCI)" under Functional Description (Module 2 of the Spartan-3 data sheet)**.

### *DCI Termination Types*

If the I/O in an I/O bank do not use the DCI feature, then no external resistors are required and both the VRP_# and VRN_# pins are available for user I/O, as shown in Figure 3a.

If the I/O standards within the associated I/O bank require single termination—such as GTL_DCI, GTLP_DCI, or HSTL_III_DCI—then only the VRP_# signal connects to a 1% precision impedance-matching resistor, as shown in Figure 3b. The VRN_# pin is available for user I/O.

Finally, if the I/O standards with the associated I/O bank require split termination—such as HSTL_I_DCI, SSTL2_I_DCI, SSTL2_II_DCI, or LVDS_25_DCI and LVDSEXT_25_DCI receivers—then both the VRP_# and VRN_# pins connect to separate 1% precision impedance-matching resistors, as shown in Figure 3c. Neither pin is available for user I/O.

## GCLK: Global Clock Buffer Inputs or General-Purpose I/O Pins

These pins are user-I/O pins unless they specifically connect to one of the eight low-skew global clock buffers on the device, specified using the IBUFG primitive.

There are eight GCLK pins per device and two each appear in the top-edge banks, Bank 0 and 1, and the bottom-edge banks, Banks 4 and 5. See Figure 1 for a picture of bank labeling.

During configuration, these pins behave exactly like user-I/O pins.

## CONFIG: Dedicated Configuration Pins

The dedicated configuration pins control the configuration process and are not available as user-I/O pins. Every package has seven dedicated configuration pins. All CONFIG-type pins are powered by the +2.5V VCCAUX supply.

See **"Configuration" under Functional Description (Module 2 of the Spartan-3 data sheet)**.

### *CCLK: Configuration Clock*

The configuration clock signal on this pin synchronizes the reading or writing of configuration data. This pin is an input for the Slave configuration modes, both parallel and serial.

After configuration, the CCLK pin is in a high-impedance, floating state. By default, CCLK optionally is pulled High to VCCAUX as defined by the CclkPin bitstream selection. Any clocks applied to CCLK after configuration are ignored unless the bitstream option Persist is set to Yes, which retains the configuration interface. Persist is set to No by default. However, if Persist is set to Yes, then all clock edges are potentially active events, depending on the other configuration control signals.

The bitstream generator option ConfigRate determines the frequency of the internally-generated CCLK oscillator required for the Master configuration modes. The actual frequency is approximate due to the characteristics of the silicon oscillator and varies by up to 30% over the temperature and voltage range. By default, CCLK operates at approximately 6 MHz. Via the ConfigRate option, the oscillator frequency can be approximately 3, 6, 12, 25, or 50 MHz. At power-on, CCLK always starts operation at its lowest frequency. The device does not start operating at the higher frequency until the ConfigRate control bits are loaded during the configuration process.

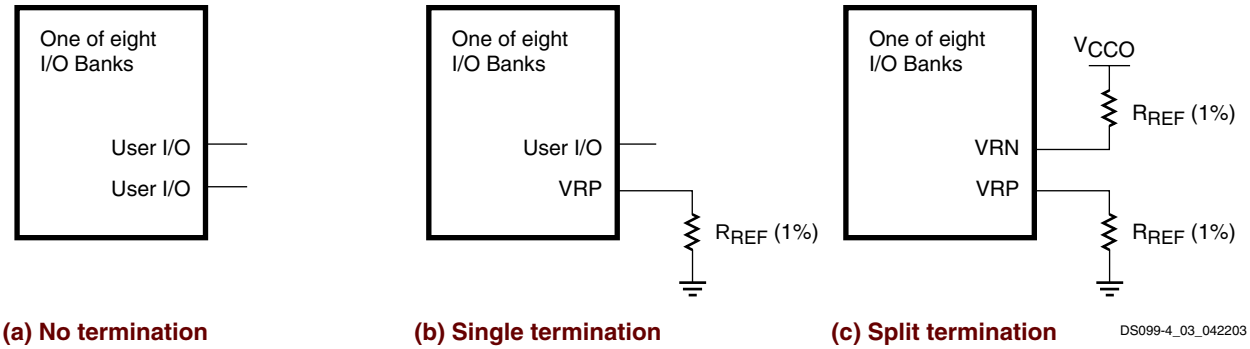



*Figure 3:* **DCI Termination Types**

### *PROG_B: Program/Configure Device*

This asynchronous pin initiates the configuration or re-configuration processes. A Low-going pulse resets the configuration logic, initializing the configuration memory. This initialization process cannot finish until PROG_B returns High. Asserting PROG_B Low for an extended period delays the configuration process. At power-up, there is always a weak pull-up resistor to VCCAUX on this pin. After configuration, the bitstream generator option ProgPin determines whether or not the weak pull-up resistor is present. By default, the ProgPin option retains the weak pull-up resistor.

After configuration, hold the PROG_B input High. Any Low-going pulse on PROG_B restarts the configuration process.

*Table 5:* **PROG_B Operation**

| PROG_B Input | Response |
|---|---|
| **Power-up** | Automatically initiates configuration process. |
| **Low-going pulse**  | Initiate (re-)configuration process and continue to completion. |
| **Extended Low**  | Initiate (re-)configuration process and stall process at step where configuration memory is cleared. Process is stalled until PROG_B returns High. |
| **1** | If the configuration process is started, continue to completion. If configuration process is complete, stay in User mode. |

### *DONE: Configuration Done, Delay Start-Up Sequence*

The FPGA produces a Low-to-High transition on this pin indicating that the configuration process is complete. The bitstream generator option DriveDone determines whether this pin functions as a totem-pole output that can drive High or as an open-drain output. If configured as an open-drain output—which is the default behavior—then a pull-up resistor is required to produce a High logic level. There is a bitstream option that provides an internal weak pull-up resistor, otherwise an external pull-up resistor is required.

The open-drain option permits the DONE lines of multiple FPGAs to be tied together, so that the common node transitions High only after all of the FPGAs have completed configuration. Externally holding the open-drain DONE pin Low delays the start-up sequence, which marks the transition to user mode.

Once the FPGA enters User mode after completing configuration, the DONE pin no longer drives the DONE pin Low. The bitstream generator option DonePin determines whether or not a weak pull-up resistor is present on the DONE pin to pull the pin to VCCAUX. If the weak pull-up resistor is eliminated, then the DONE pin must be pulled High using an external pull-up resistor or one of the FPGAs in the design must actively drive the DONE pin High via the DriveDone bitstream generator option.

The bitstream generator option DriveDone causes the FPGA to actively drive the DONE output High after configuration. This option should only be used in single-FPGA designs or on the last FPGA in a multi-FPGA daisy-chain.

By default, the bitstream generator software retains the weak pull-up resistor and does not actively drive the DONE pin as highlighted in Table 6. Table 6 shows the interaction of these bitstream options in single- and multi-FPGA designs.

*Table 6:* **DonePin and DriveDone Bitstream Option Interaction**

| DonePin | DriveDone | Single- or Multi-FPGA Design | Comments |
|---|---|---|---|
| Pullnone | No | Single | External pull-up resistor, with value between 330Ω to 3.3kΩ, required on DONE. |
| Pullnone | No | Multi | External pull-up resistor, with value between 330Ω to 3.3kΩ, required on common node connecting to all DONE pins. |
| Pullnone | Yes | Single | OK, no external requirements. |
| Pullnone | Yes | Multi | DriveDone on last device in daisy-chain only. No external requirements. |
| Pullup | No | Single | OK, but weak pull-up on DONE pin has slow rise time. May require 330 Ω pull-up resistor for high CCLK frequencies. |
| Pullup | No | Multi | External pull-up resistor, with value between 330Ω to 3.3kΩ, required on common node connecting to all DONE pins. |
| Pullup | Yes | Single | OK, no external requirements. |
| Pullup | Yes | Multi | DriveDone on last device in daisy-chain only. No external requirements. |

### *M2, M1, M0: Configuration Mode Selection*

These inputs select the mode to configure the FPGA. The logic levels applied to the mode pins are sampled on the rising edge of INIT_B.

*Table 7:* **Spartan-3 Configuration Mode Select Settings**

| Configuration Mode | M2 | M1 | M0 |
|---|---|---|---|
| Master Serial | 0 | 0 | 0 |
| Slave Serial | 1 | 1 | 1 |
| Master Parallel | 0 | 1 | 1 |
| Slave Parallel | 1 | 1 | 0 |
| JTAG | 1 | 0 | 1 |
| Reserved | 0 | 0 | 1 |
| Reserved | 0 | 1 | 0 |
| Reserved | 1 | 0 | 0 |
| After Configuration | X | X | X |

**Notes:**
1. X = don't care, either 0 or 1.

In user mode, after configuration successfully completes, any levels applied to these input are ignored. Each of the bitstream generator options M0Pin, M1Pin, and M2Pin determines whether a weak pull-up resistor, weak pull-down resistor, or no resistor is present on its respective mode pin, M0, M1, or M2.

### *HSWAP_EN: Disable Weak Pull-up Resistors During Configuration*

A Low on this asynchronous pin enables weak pull-up resistors on all user I/Os, although only until device configuration completes. A High disables the weak pull-up resistors (during configuration, which is the desired state for some applications.

*Table 8:* **HSWAP_EN Encoding**

| HSWAP_EN | Function |
|---|---|
| **During Configuration** | |
| 0 | Enable weak pull-up resistors on all pins not actively involved in the configuration process. Pull-ups are only active until configuration completes. See Table 10. |
| 1 | No pull-up resistors during configuration. |
| **After Configuration, User Mode** | |
| X | This pin has no function except during device configuration. |

**Notes:**
1. X = don't care, either 0 or 1.

After configuration, HSWAP_EN essentially becomes a "don't care" input and any pull-up resistors previously enabled by HSWAP_EN are disabled. If a user I/O in the application requires a weak pull-up resistor after configuration, place a PULLUP primitive on the associated I/O pin.

The Bitstream generator option HswapenPin determines whether a weak pull-up resistor to VCCAUX, a weak pull-down resistor, or no resistor is present on HSWAP_EN after configuration.

## JTAG: Dedicated JTAG Port Pins

These pins are dedicated connections to the four-wire IEEE 1532/IEEE 1149.1 JTAG port, shown in Figure 4. The JTAG

port is used for boundary-scan testing, device configuration, application debugging, and possibly an additional serial port for the application. These pins are dedicated and are not available as user-I/O pins. Every package has four dedicated JTAG pins and these pins are powered by the +2.5V VCCAUX supply.

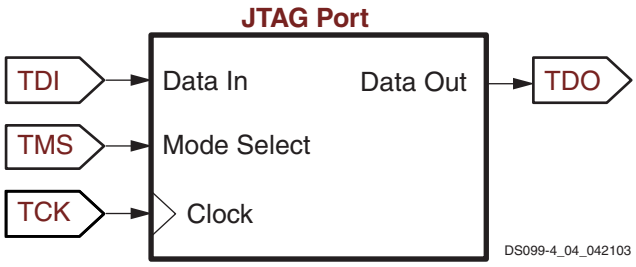



*Figure 4:* **JTAG Port**

### *Using JTAG Port After Configuration*

By default, the JTAG port is disabled after the configuration process completes. To continue using the JTAG port, after configuration, place a BSCAN_SPARTAN3 primitive in the design.

Furthermore, the contents of the User ID register within the JTAG port can be specified as a Bitstream Generation option. By default, the 32-bit User ID register contains 0xFFFFFFFF.

### *Precautions When Using the JTAG Port in 3.3V Environments*

The JTAG port is powered by the +2.5V VCCAUX power supply. The JTAG pins can tolerate 3.3V interface signals but inputs will be clamped by the electro-static discharge (ESD) protection diodes to VCCAUX. Similarly, the TDO pin is a CMOS output powered from +2.5V. Driving a 3.3V input from TDO may require a pull-up resistor to +3.3V.

The following interface precautions are recommended when connecting the JTAG port to a 3.3V interface.

1. Set any inactive JTAG signals, including TCK, Low when not actively used.
2. Drive the JTAG input pins with no more than 10 mA drivers.

## VREF: User I/O or Input Buffer Reference Voltage for Special Interface Standards

These pins are individual user-I/O pins unless collectively they supply an input reference voltage, VREF_#, for any SSTL, HSTL, GTL, or GTLP I/Os implemented in the associated I/O bank.

The '#' character in the pin name represents an integer, 0 through 7, that indicates the associated I/O bank.

The VREF function becomes active for this pin whenever a signal standard requiring a reference voltage is used in the associated bank.

If used as a user I/O, then each pin behaves as an independent I/O described in the I/O type section. If used for a reference voltage within a bank, then *all* VREF pins within the bank must be connected to the same reference voltage.

Spartan-3 devices are designed and characterized to support certain I/O standards when VREF is connected to +1.25V, +1.10V, +1.00V, +0.90V, +0.80V, and +0.75V.

During configuration, these pins behave exactly like user-I/O pins.

*Table 9:* **JTAG Pin Descriptions**

| Pin Name | Direction | Description | Bitstream Generation Option |
|---|---|---|---|
| TCK | Input | **Test Clock:** The TCK clock signal synchronizes all boundary scan operations on its rising edge. | The BitGen option TckPin determines whether a weak pull-up resistor, weak pull-down resistor or no resistor is present. |
| TDI | Input | **Test Data Input:** TDI is the serial data input for all JTAG instruction and data registers. This input is sampled on the rising edge of TCK. | The BitGen option TdiPin determines whether a weak pull-up resistor, weak pull-down resistor or no resistor is present. |
| TMS | Input | **Test Mode Select:** The TMS input controls the sequence of states through which the JTAG TAP state machine passes. This input is sampled on the rising edge of TCK. | The BitGen option TmsPin determines whether a weak pull-up resistor, weak pull-down resistor or no resistor is present. |
| TDO | Output | **Test Data Output:** The TDO pin is the data output for all JTAG instruction and data registers. This output is sampled on the rising edge of TCK. | The BitGen option TdoPin determines whether a weak pull-up resistor, weak pull-down resistor or no resistor is present. |

If designing for footprint compatibility across the range of devices in a specific package, and if the VREF_# pins within a bank connect to an input reference voltage, then also connect any N.C. (not connected) pins on the smaller devices in that package to the input reference voltage. More details are provided later for each package type

### N.C. Type: Unconnected Package Pins

Pins marked as "N.C." are unconnected for the specific device/package combination. For other devices in this same package, this pin may be used as an I/O or VREF connection. In both the pinout tables and the footprint diagrams, unconnected pins are noted with either a black diamond symbol (◆) or a black square symbol (■).

If designing for footprint compatibility across multiple device densities, check the pin types of the other Spartan-3 devices available in the same footprint. If the N.C. pin matches to VREF pins in other devices, and the VREF pins are used in the associated I/O bank, then connect the N.C. to the VREF voltage source.

### VCCO Type: Output Voltage Supply for I/O Bank

Each I/O bank has its own set of voltage supply pins that determines the output voltage for the output buffers in the I/O bank. Furthermore, for some I/O standards such as LVCMOS, LVCMOS25, LVTTL, etc., VCCO sets the input threshold voltage on the associated input buffers.

Spartan-3 devices are designed and characterized to support various I/O standards for VCCO values of +1.2V, +1.5V, +1.8V, +2.5V, and +3.3V.

Most VCCO pins are labeled as VCCO_# where the '#' symbol represents the associated I/O bank number, an integer ranging from 0 to 7. In the 144-pin TQFP package (TQ144) however, the VCCO pins along an edge of the device are combined into a single VCCO input. For example, the VCCO inputs for Bank 0 and Bank 1 along the top edge of the package are combined and relabeled VCCO_TOP. The bottom, left, and right edges are similarly combined.

In Serial configuration mode, VCCO_4 must be at a level compatible with the attached configuration memory or data source. In Parallel configuration mode, both VCCO_4 and VCCO_5 must be at the same compatible voltage level.

All VCCO inputs to a bank must be connected together and to the voltage supply. Furthermore, there must be sufficient supply decoupling to guarantee problem-free operation, as described in **XAPP623**: *Power Distribution System (PDS) Design: Using Bypass/Decoupling Capacitors*.

### VCCINT Type: Voltage Supply for Internal Core Logic

Internal core logic circuits such as the configurable logic blocks (CLBs) and programmable interconnect operate from the VCCINT voltage supply inputs. VCCINT must be +1.2V.

All VCCINT inputs must be connected together and to the +1.2V voltage supply. Furthermore, there must be sufficient supply decoupling to guarantee problem-free operation, as described in **XAPP623**: *Power Distribution System (PDS) Design: Using Bypass/Decoupling Capacitors*.

### VCCAUX Type: Voltage Supply for Auxiliary Logic

The VCCAUX pins supply power to various auxiliary circuits, such as to the Digital Clock Managers (DCMs), the JTAG pins, and to the dedicated configuration pins (CONFIG type). VCCAUX must be +2.5V.

All VCCAUX inputs must be connected together and to the +2.5V voltage supply. Furthermore, there must be sufficient supply decoupling to guarantee problem-free operation, as described in **XAPP623**: *Power Distribution System (PDS) Design: Using Bypass/Decoupling Capacitors*.

Because VCCAUX connects to the DCMs and the DCMs are sensitive to voltage changes, be sure that the VCCAUX supply and the ground return paths are designed for low noise and low voltage drop, especially that caused by a large number of simultaneous switching I/Os.

### GND Type: Ground

All GND pins must be connected and have a low resistance path back to the various VCCO, VCCINT, and VCCAUX supplies.

## Pin Behavior During Configuration

Table 10 shows how various pins behave during the FPGA configuration process. The actual behavior depends on the values applied to the M2, M1, and M0 mode select pins and the HSWAP_EN pin. The mode select pins determine which of the DUAL type pins are active during configuration. In JTAG configuration mode, none of the DUAL-type pins are used for configuration and all behave as user-I/O pins.

All DUAL-type pins not actively used during configuration and all I/O-type, DCI-type, VREF-type, GCLK-type pins are high impedance (floating, three-stated, Hi-Z) during the configuration process. These pins are indicated in Table 10 as shaded table entries or cells. These pins have a weak pull-up resistor to their associated VCCO if the HSWAP_EN pin is Low.

After configuration completes, some pins have optional behavior controlled by the configuration bitstream loaded into the part. For example, via the bitstream, all unused I/O pins can collectively be configured to have a weak pull-up resistor, a weak pull-down resistor, or be left in a high-impedance state.

*Table 10:* **Pin Behavior After Power-Up, During Configuration**

| Pin Name | Configuration Mode Settings <M2:M1:M0> | | | | | Bitstream Configuration Option |
|---|---|---|---|---|---|---|
| | Serial Modes | | SelectMap Parallel Modes | | JTAG Mode <1:0:1> | |
| | Master <0:0:0> | Slave <1:1:1> | Master <0:1:1> | Slave <1:1:0> | | |
| **I/O:** General-purpose I/O pins | | | | | | |
| IO | | | | | | UnusedPin |
| IO_Lxxy_# | | | | | | UnusedPin |
| **DUAL:** Dual-purpose configuration pins | | | | | | |
| IO_Lxxy_#/ DIN/D0 | DIN (I) | DIN (I) | D0 (I/O) | D0 (I/O) | | Persist UnusedPin |
| IO_Lxxy_#/ D1 | | | D1 (I/O) | D1 (I/O) | | Persist UnusedPin |
| IO_Lxxy_#/ D2 | | | D2 (I/O) | D2 (I/O) | | Persist UnusedPin |
| IO_Lxxy_#/ D3 | | | D3 (I/O) | D3 (I/O) | | Persist UnusedPin |
| IO_Lxxy_#/ D4 | | | D4 (I/O) | D4 (I/O) | | Persist UnusedPin |
| IO_Lxxy_#/ D5 | | | D5 (I/O) | D5 (I/O) | | Persist UnusedPin |
| IO_Lxxy_#/ D6 | | | D6 (I/O) | D6 (I/O) | | Persist UnusedPin |
| IO_Lxxy_#/ D7 | | | D7 (I/O) | D7 (I/O) | | Persist UnusedPin |
| IO_Lxxy_#/ CS_B | | | CS_B (I) | CS_B (I) | | Persist UnusedPin |
| IO_Lxxy_#/ RDWR_B | | | RDWR_B (I) | RDWR_B (I) | | Persist UnusedPin |
| IO_Lxxy_#/ BUSY/DOUT | DOUT (O) | DOUT (O) | BUSY (O) | BUSY (O) | | Persist UnusedPin |
| IO_Lxxy_#/ INIT_B | INIT_B (I/OD) | INIT_B (I/OD) | INIT_B (I/OD) | INIT_B (I/OD) | | UnusedPin |
| **DCI:** Digitally Controlled Impedance reference resistor input pins | | | | | | |
| IO_Lxxy_#/ VRN_# | | | | | | UnusedPin |
| IO/VRN_# | | | | | | UnusedPin |
| IO_Lxxy_#/ VRP_# | | | | | | UnusedPin |
| IO/VRP_# | | | | | | UnusedPin |

*Table 10:* **Pin Behavior After Power-Up, During Configuration *(Continued)***

| Pin Name | Configuration Mode Settings <M2:M1:M0> | | | | | Bitstream Configuration Option |
|---|---|---|---|---|---|---|
| | Serial Modes | | SelectMap Parallel Modes | | JTAG Mode <1:0:1> | |
| | Master <0:0:0> | Slave <1:1:1> | Master <0:1:1> | Slave <1:1:0> | | |
| **GCLK:** Global clock buffer inputs | | | | | | |
| IO_Lxxy_#/ GCLK0 through GCLK7 | | | | | | UnusedPin |
| **VREF:** I/O bank input reference voltage pins | | | | | | |
| IO_Lxxy_#/ VREF_# | | | | | | UnusedPin |
| IO/VREF_# | | | | | | UnusedPin |
| **CONFIG:** Dedicated configuration pins | | | | | | |
| CCLK | CCLK (O) | CCLK (I) | CCLK (O) | CCLK (I) | | CclkPin ConfigRate |
| PROG_B | PROG_B (I) (pull-up) | PROG_B (I) (pull-up) | PROG_B (I) (pull-up) | PROG_B (I) (pull-up) | PROG_B (I), Via JPROG_B instruction | ProgPin |
| DONE | DONE (I/OD) | DONE (I/OD) | DONE (I/OD) | DONE (I/OD) | DONE (I/OD) | DriveDone DonePin DonePipe |
| M2 | M2=0 (I) | M2=1 (I) | M2=0 (I) | M2=1 (I) | M2=1 (I) | M2Pin |
| M1 | M1=0 (I) | M1=1 (I) | M1=1 (I) | M1=1 (I) | M1=0 (I) | M1Pin |
| M0 | M0=0 (I) | M0=1 (I) | M0=1 (I) | M0=0 (I) | M0=1 (I) | M0Pin |
| HSWAP_EN | HSWAP_EN (I) | HSWAP_EN (I) | HSWAP_EN (I) | HSWAP_EN (I) | HSWAP_EN (I) | HswapenPin |
| **JTAG:** JTAG interface pins | | | | | | |
| TDI | TDI (I) | TDI (I) | TDI (I) | TDI (I) | TDI (I) | TdiPin |
| TMS | TMS (I) | TMS (I) | TMS (I) | TMS (I) | TMS (I) | TmsPin |
| TCK | TCK (I) | TCK (I) | TCK (I) | TCK (I) | TCK (I) | TckPin |
| TDO | TDO (O) | TDO (O) | TDO (O) | TDO (O) | TDO (O) | TdoPin |
| **VCCO:** I/O bank output voltage supply pins | | | | | | |
| VCCO_4 (for DUAL pins) | Same voltage as external interface | Same voltage as external interface | Same voltage as external interface | Same voltage as external interface | VCCO_4 | |
| VCCO_5 (for DUAL pins) | VCCO_5 | VCCO_5 | Same voltage as external interface | Same voltage as external interface | VCCO_5 | |
| VCCO_# | VCCO_# | VCCO_# | VCCO_# | VCCO_# | VCCO_# | |
| **VCCAUX:** Auxiliary voltage supply pins | | | | | | |
| VCCAUX | +2.5V | +2.5V | +2.5V | +2.5V | +2.5V | |

*Table 10:* **Pin Behavior After Power-Up, During Configuration *(Continued)***

| Pin Name | Configuration Mode Settings <M2:M1:M0> | | | | | Bitstream Configuration Option |
|---|---|---|---|---|---|---|
| | Serial Modes | | SelectMap Parallel Modes | | JTAG Mode <1:0:1> | |
| | Master <0:0:0> | Slave <1:1:1> | Master <0:1:1> | Slave <1:1:0> | | |
| **VCCINT:** Internal core voltage supply pins | | | | | | |
| VCCINT | +1.2V | +1.2V | +1.2V | +1.2V | +1.2V | |
| **GND:** Ground supply pins | | | | | | |
| GND | GND | GND | GND | GND | GND | |

**Notes:**
1. #= I/O bank number, an integer from 0 to 7.
2. (I) = input, (O) = output, (OD) = open-drain output, (I/O) = bidirectional, (I/OD) = bidirectional with open-drain output. Open-drain output requires pull-up to create logic High level.
3. Shaded cell indicates that the pin is high-impedance during configuration. To enable a soft pull-up resistor during configuration, drive or tie HSWAP_EN Low.

## Bitstream Options

Table 11 lists the various bitstream options that affect pins on a Spartan-3 FPGA. The table shows the names of the affected pins, describes the function of the bitstream option, the name of the bitstream generator option variable, and the legal values for each variable. The default option setting for each variable is indicated with bold, underlined text.

*Table 11:* **Bitstream Options Affecting Spartan-3 Pins**

| Affected Pin Name(s) | Bitstream Generation Function | Option Variable Name | Values (default value) |
|---|---|---|---|
| All unused I/O pins of type I/O, DUAL, GCLK, DCI, VREF | For all I/O pins that are unused after configuration, this option defines whether the I/Os are individually tied to VCCO via a weak pull-up resistor, tied ground via a weak pull-down resistor, or left floating. If left floating, the unused pins should be connected to a defined logic level, either from a source internal to the FPGA or external. | UnusedPin | • **Pulldown**<br>• Pullup<br>• Pullnone |
| IO_Lxxy_#/DIN, IO_Lxxy_#/DOUT, IO_Lxxy_#/INIT_B | Serial configuration mode: If set to Yes, then these pins retain their functionality after configuration completes, allowing for device (re-)configuration. Readback is not supported in with serial mode. | Persist | • **No**<br>• Yes |
| IO_Lxxy_#/D0, IO_Lxxy_#/D1, IO_Lxxy_#/D2, IO_Lxxy_#/D3, IO_Lxxy_#/D4, IO_Lxxy_#/D5, IO_Lxxy_#/D6, IO_Lxxy_#/D7, IO_Lxxy_#/CS_B, IO_Lxxy_#/RDWR_B, IO_Lxxy_#/BUSY, IO_Lxxy_#/INIT_B | Parallel configuration mode (also called SelectMAP): If set to Yes, then these pins retain their SelectMAP functionality after configuration completes, allowing for device readback and for partial or complete (re-)configuration. | Persist | • **No**<br>• Yes |
| CCLK | After configuration, this bitstream option either pulls CCLK to VCCAUX via a weak pull-up resistor, or allows CCLK to float. | CclkPin | • **Pullup**<br>• Pullnone |
| CCLK | For Master configuration modes, this option sets the approximate frequency, in MHz, for the internal silicon oscillator. | ConfigRate | 3, **6**, 12, 25, 50 |

*Table 11:* **Bitstream Options Affecting Spartan-3 Pins *(Continued)***

| Affected Pin Name(s) | Bitstream Generation Function | Option Variable Name | Values (default value) |
|---|---|---|---|
| PROG_B | A weak pull-up resistor to VCCAUX exists on PROG_B during configuration. After configuration, this bitstream option either pulls DONE to VCCAUX via a weak pull-up resistor, or allows DONE to float. | ProgPin | • **Pullup**<br>• Pullnone |
| DONE | After configuration, this bitstream option either pulls DONE to VCCAUX via a weak pull-up resistor, or allows DONE to float. See also DriveDone option. | DonePin | • **Pullup**<br>• Pullnone |
| DONE | If set to Yes, this option allows the FPGA's DONE pin to drive High when configuration completes. By default, the DONE is an open-drain output and can only drive Low. Only single FPGAs and the last FPGA in a multi-FPGA daisy-chain should use this option. | DriveDone | • **No**<br>• Yes |
| M2 | After configuration, this bitstream option either pulls M2 to VCCAUX via a weak pull-up resistor, to ground via a weak pull-down resistor, or allows M2 to float. | M2Pin | • **Pullup**<br>• Pulldown<br>• Pullnone |
| M1 | After configuration, this bitstream option either pulls M1 to VCCAUX via a weak pull-up resistor, to ground via a weak pull-down resistor, or allows M1 to float. | M1Pin | • **Pullup**<br>• Pulldown<br>• Pullnone |
| M0 | After configuration, this bitstream option either pulls M0 to VCCAUX via a weak pull-up resistor, to ground via a weak pull-down resistor, or allows M0 to float. | M0Pin | • **Pullup**<br>• Pulldown<br>• Pullnone |
| HSWAP_EN | After configuration, this bitstream option either pulls HSWAP_EN to VCCAUX via a weak pull-up resistor, to ground via a weak pull-down resistor, or allows HSWAP_EN to float. | HswapenPin | • **Pullup**<br>• Pulldown<br>• Pullnone |
| TDI | After configuration, this bitstream option either pulls TDI to VCCAUX via a weak pull-up resistor, to ground via a weak pull-down resistor, or allows TDI to float. | TdiPin | • **Pullup**<br>• Pulldown<br>• Pullnone |
| TMS | After configuration, this bitstream option either pulls TMS to VCCAUX via a weak pull-up resistor, to ground via a weak pull-down resistor, or allows TMS to float. | TmsPin | • **Pullup**<br>• Pulldown<br>• Pullnone |
| TCK | After configuration, this bitstream option either pulls TCK to VCCAUX via a weak pull-up resistor, to ground via a weak pull-down resistor, or allows TCK to float. | TckPin | • **Pullup**<br>• Pulldown<br>• Pullnone |
| TDO | After configuration, this bitstream option either pulls TDO to VCCAUX via a weak pull-up resistor, to ground via a weak pull-down resistor, or allows TDO to float. | TdoPin | • **Pullup**<br>• Pulldown<br>• Pullnone |

## Setting Options via BitGen Command-Line Program

To set one or more bitstream generator options using the BitGen command-line program, enter

```
bitgen -g <variable_name>:<value>
[<variable_name>:<value> …]
```

where **<variable_name>** is one of the entries from Table 11 and **<value>** is one of the possible values for the specified variable. Multiple bitstream options may be entered in this manner.

For a complete listing of all BitGen options, their possible settings, and their default settings, enter the following command.

```
bitgen -help spartan3
```

## Setting Options in Project Navigator

To set the bitstream generation options in Xilinx ISE Project Navigator, right-click on the **Generate Programming File** step in the Process View and click **Properties**, as shown in Figure 5.

Click the **Configuration options** tab and modify the available options as required by the application, as shown in Figure 6.

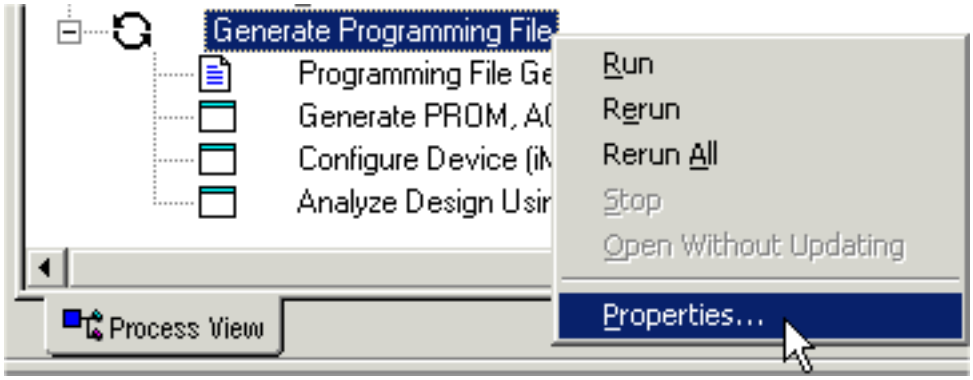



*Figure 5:* **Setting Properties for Generate Programming File Step**

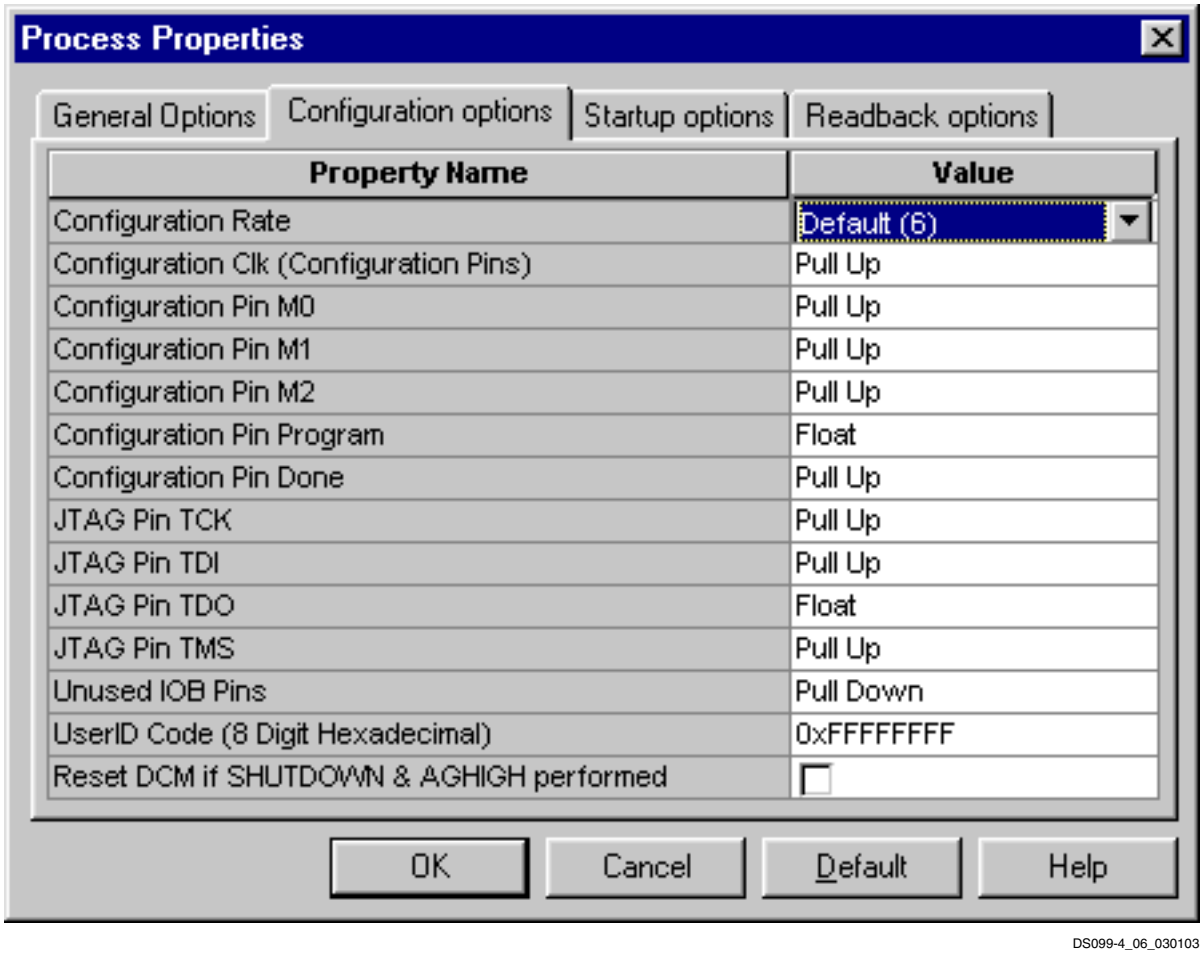



*Figure 6:* **Configuration Option Settings**

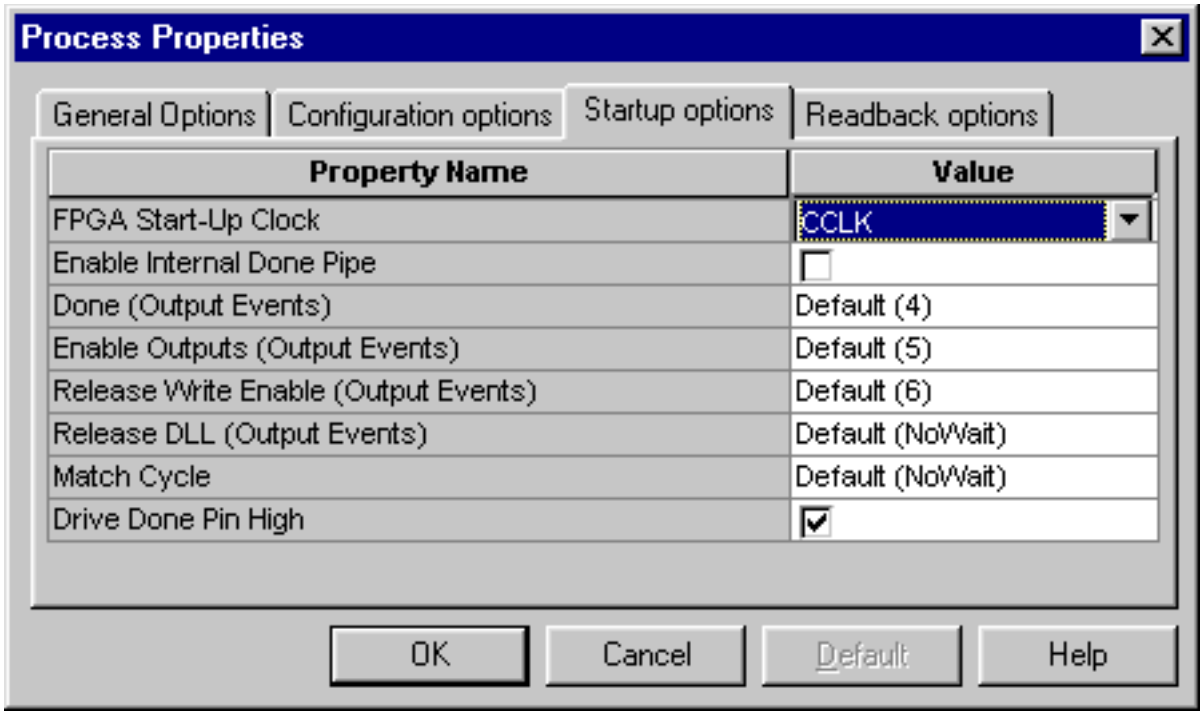



*Figure 7:* **Setting to Drive DONE Pin High after Configuration**

To have the DONE pin drive High after successful configuration, click the **Startup options** tab and check the **Drive Done Pin High** box, as shown in Figure 7.

Click **OK** when finished.

Again, right-click on the **Generate Programming File** step in the Process View. This time, choose **Run** or **Rerun** to execute the changes.

## Package Overview

Table 12 shows the eight, low-cost, space-saving production packages for the Spartan-3 family. Not all Spartan-3 densities are available in all packages. However, for a specific package there is a common footprint for that supports the various devices available in that package. See the footprint diagrams that follow.

*Table 12:* **Spartan-3 Family Package Options**

| Package | Leads | Type | Maximum I/O | Pitch (mm) | Area (mm) | Height (mm) |
|---|---|---|---|---|---|---|
| VQ100 | 100 | Very-thin Quad Flat Pack | 63 | 0.5 | 16 x 16 | 1.20 |
| TQ144 | 144 | Thin Quad Flat Pack | 97 | 0.5 | 22 x 22 | 1.60 |
| PQ208 | 208 | Quad Flat Pack | 141 | 0.5 | 30.6 x 30.6 | 4.10 |
| FT256 | 256 | Fine-pitch, Thin Ball Grid Array | 173 | 1.0 | 17 x 17 | 1.55 |
| FG456 | 456 | Fine-pitch Ball Grid Array | 333 | 1.0 | 23 x 23 | 2.60 |
| FG676 | 676 | Fine-pitch Ball Grid Array | 405 | 1.0 | 27 x 27 | 2.60 |
| FG900 | 900 | Fine-pitch Ball Grid Array | 549 | 1.0 | 31 x 31 | 2.60 |
| FG1156 | 1156 | Fine-pitch Ball Grid Array | 692 | 1.0 | 35 x 35 | 2.60 |

Detailed mechanical drawings for each package type are available from the Xilinx website at the specified location in Table 13.

*Table 13:* **Xilinx Package Mechanical Drawings**

| Package | Web Link (URL) |
|---|---|
| VQ100 | http://www.xilinx.com/bvdocs/packages/vq100.pdf |
| TQ144 | http://www.xilinx.com/bvdocs/packages/tq144.pdf |
| PQ208 | http://www.xilinx.com/bvdocs/packages/pq208.pdf |
| FT256 | http://www.xilinx.com/bvdocs/packages/ft256.pdf |
| FG456 | http://www.xilinx.com/bvdocs/packages/fg456.pdf |
| FG676 | http://www.xilinx.com/bvdocs/packages/fg676.pdf |
| FG900 | http://www.xilinx.com/bvdocs/packages/fg900.pdf |
| FG1156 | http://www.xilinx.com/bvdocs/packages/fg1156.pdf |

Each package has three separate voltage supply inputs—VCCINT, VCCAUX, and VCCO—and a common ground return, GND. The numbers of pins dedicated to these functions varies by package, as shown in Table 14.

*Table 14:* **Power and Ground Supply Pins by Package**

| Package | VCCINT | VCCAUX | VCCO | GND |
|---|---|---|---|---|
| VQ100 | 4 | 4 | 8 | 10 |
| TQ144 | 4 | 4 | 12 | 16 |
| PQ208 | 4 | 8 | 12 | 28 |
| FT256 | 8 | 8 | 24 | 32 |
| FG456 | 12 | 8 | 40 | 52 |
| FG676 | 20 | 16 | 64 | 76 |
| FG900 | 32 | 24 | 80 | 120 |
| FG1156 | 40 | 32 | 104 | 184 |

A majority of package pins are user-defined I/O pins. However, the numbers and characteristics of these I/O depends on the device type and the package in which it is available, as shown in Table 15. The table shows the maximum number of single-ended I/O pins available, assuming that all I/O-, DUAL-, DCI-, VREF-, and GCLK-type pins are used as general-purpose I/O. Likewise, the table shows the maximum number of differential pin-pairs available on the package. Finally, the table shows how the total maximum user I/Os are distributed by pin type, including the number of unconnected—i.e., N.C.—pins on the device.

*Table 15:* **Maximum User I/Os by Package**

| Device | Package | Maximum User I/Os | Maximum Differential Pairs | All Possible I/O Pins by Type | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | I/O | DUAL | DCI | VREF | GCLK | N.C. |
| XC3S50 | VQ100 | 63 | 29 | 22 | 12 | 14 | 7 | 8 | 0 |
| XC3S200 | VQ100 | 63 | 29 | 22 | 12 | 14 | 7 | 8 | 0 |
| XC3S50 | TQ144 | 97 | 46 | 51 | 12 | 14 | 12 | 8 | 0 |
| XC3S200 | TQ144 | 97 | 46 | 51 | 12 | 14 | 12 | 8 | 0 |
| XC3S400 | TQ144 | 97 | 46 | 51 | 12 | 14 | 12 | 8 | 0 |
| XC3S50 | PQ208 | 124 | 56 | 72 | 12 | 16 | 16 | 8 | 17 |
| XC3S200 | PQ208 | 141 | 62 | 83 | 12 | 16 | 22 | 8 | 0 |
| XC3S400 | PQ208 | 141 | 62 | 83 | 12 | 16 | 22 | 8 | 0 |

*Table 15:* **Maximum User I/Os by Package *(Continued)***

| Device | Package | Maximum User I/Os | Maximum Differential Pairs | All Possible I/O Pins by Type | | | | | N.C. |
|---|---|---|---|---|---|---|---|---|---|
| | | | | I/O | DUAL | DCI | VREF | GCLK | |
| XC3S200 | FT256 | 173 | 76 | 113 | 12 | 16 | 24 | 8 | 0 |
| XC3S400 | FT256 | 173 | 76 | 113 | 12 | 16 | 24 | 8 | 0 |
| XC3S1000 | FT256 | 173 | 76 | 113 | 12 | 16 | 24 | 8 | 0 |
| XC3S400 | FG456 | 264 | 116 | 196 | 12 | 16 | 32 | 8 | 69 |
| XC3S1000 | FG456 | 333 | 149 | 261 | 12 | 16 | 36 | 8 | 0 |
| XC3S1500 | FG456 | 333 | 149 | 261 | 12 | 16 | 36 | 8 | 0 |
| XC3S1000 | FG676 | 391 | 175 | 315 | 12 | 16 | 40 | 8 | 98 |
| XC3S1500 | FG676 | 487 | 221 | 403 | 12 | 16 | 48 | 8 | 2 |
| XC3S2000 | FG676 | 489 | 221 | 405 | 12 | 16 | 48 | 8 | 0 |
| XC3S2000 | FG900 | 565 | 270 | 481 | 12 | 16 | 48 | 8 | 68 |
| XC3S4000 | FG900 | 633 | 300 | 549 | 12 | 16 | 48 | 8 | 0 |
| XC3S5000 | FG900 | 633 | 300 | 549 | 12 | 16 | 48 | 8 | 0 |
| XC3S4000 | FG1156 | 712 | 312 | 621 | 12 | 16 | 55 | 8 | 73 |
| XC3S5000 | FG1156 | 784 | 344 | 692 | 12 | 16 | 56 | 8 | 1 |

Electronic versions of the package pinout tables and footprints are available for download from the Xilinx website. Using a spreadsheet program, the data can be sorted and reformatted according to any specific needs. Similarly, the ASCII-text file is easily parsed by most scripting programs. Download the files from the following location:

http://www.xilinx.com/bvdocs/publications/s3_pin.zip

## VQ100 Footprint

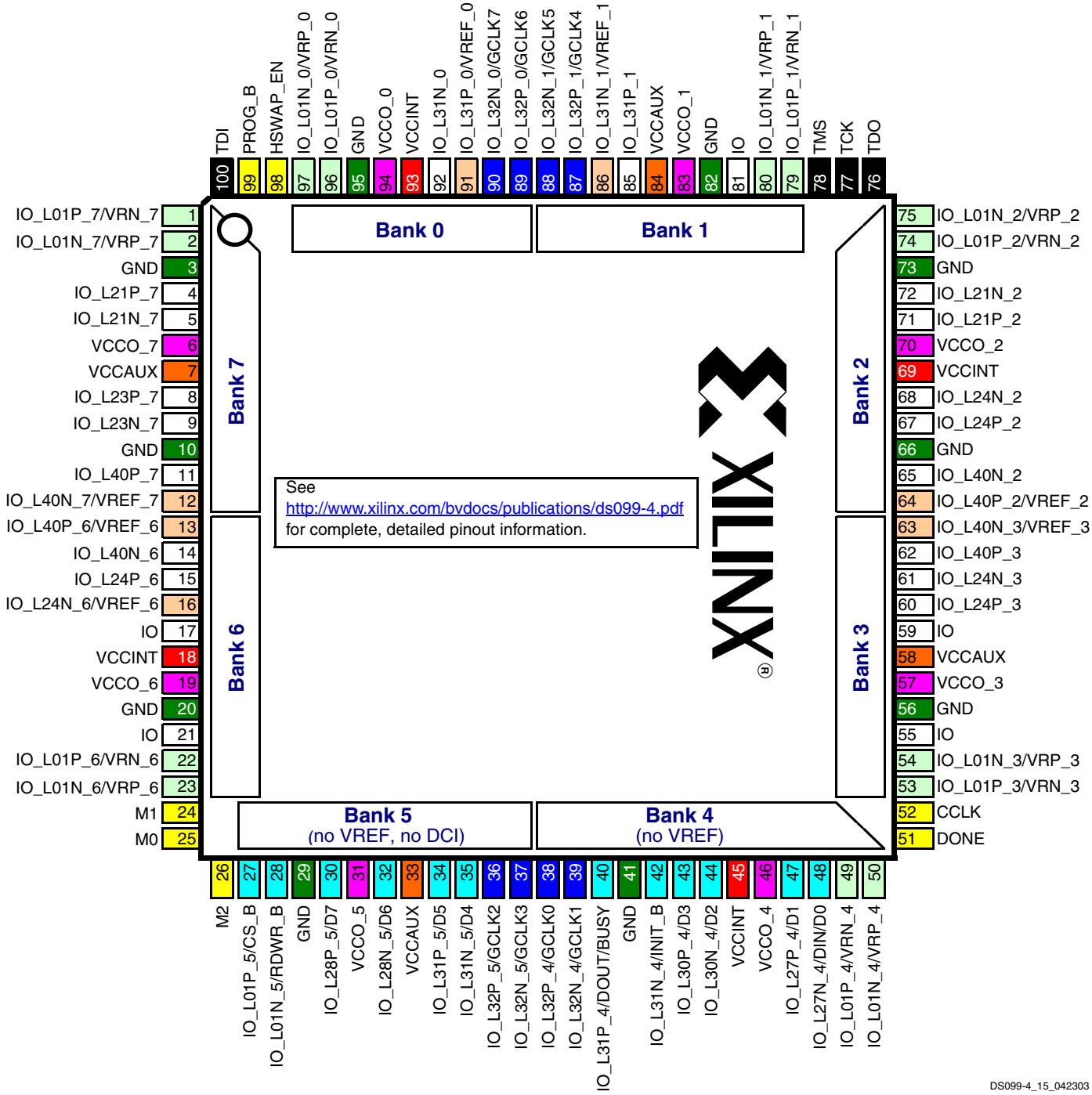



*Figure 8:* **VQ100 Package Footprint (top view). Note pin 1 indicator in top-left corner and logo orientation.**

| | | |
|---|---|---|
| 22 **I/O:** Unrestricted, general-purpose user I/O | 12 **DUAL:** Configuration pin, then possible user I/O | 7 **VREF:** User I/O or input voltage reference for bank |
| 14 **DCI:** User I/O or reference resistor input for bank | 8 **GCLK:** User I/O or global clock buffer input | 8 **VCCO:** Output voltage supply for bank |
| 7 **CONFIG:** Dedicated configuration pins | 4 **JTAG:** Dedicated JTAG port pins | 4 **VCCINT:** Internal core voltage supply (+1.2V) |
| 0 **N.C.:** No unconnected pins in this package | 10 **GND:** Ground | 4 **VCCAUX:** Auxiliary voltage supply (+2.5V) |

## TQ144 Footprint

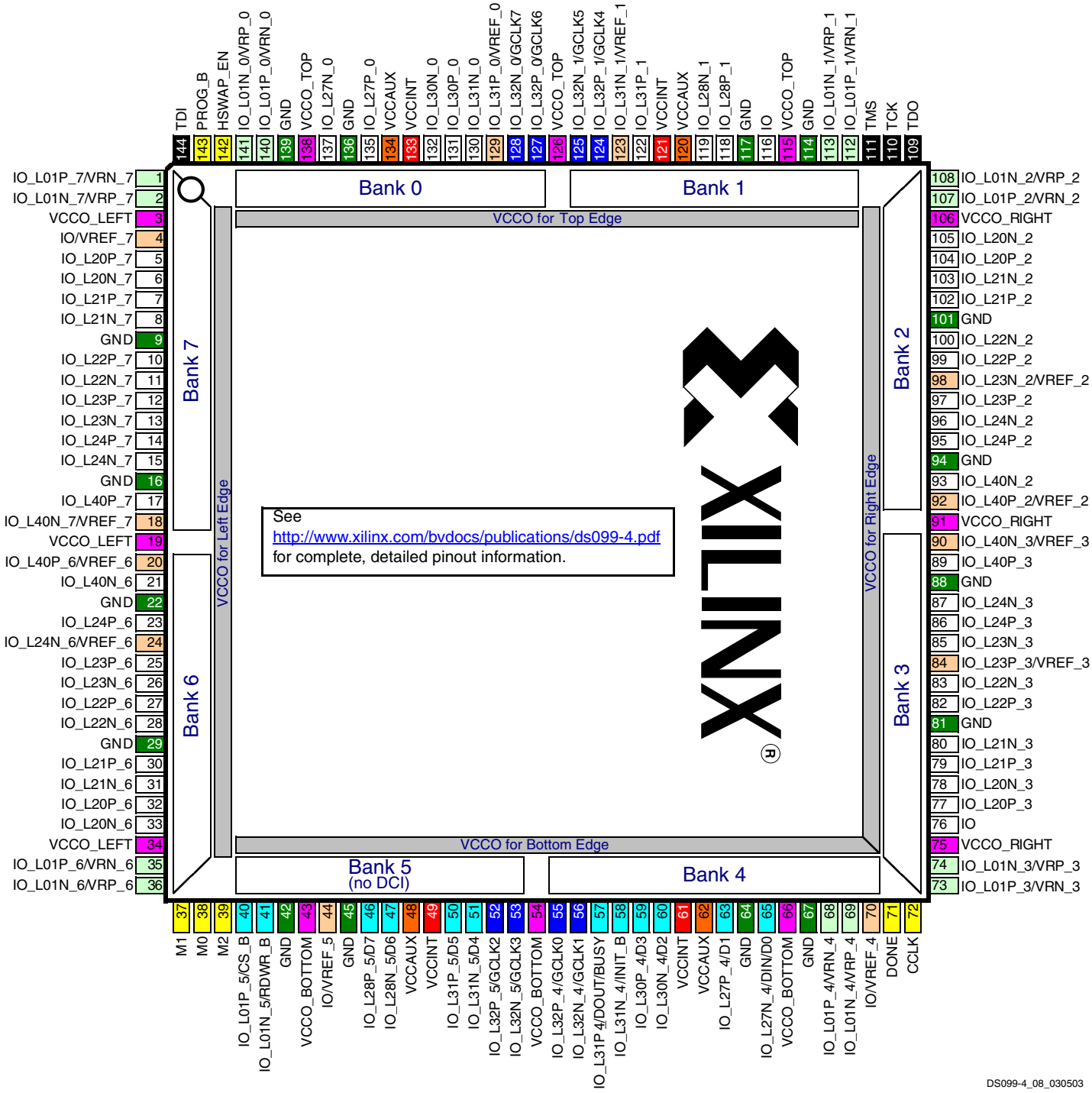



*Figure 9:* **TQ144 Package Footprint (top view). Note pin 1 indicator in top-left corner and logo orientation.**

| | | |
|---|---|---|
| 51 **I/O:** Unrestricted, general-purpose user I/O | 12 **DUAL:** Configuration pin, then possible user I/O | 12 **VREF:** User I/O or input voltage reference for bank |
| 14 **DCI:** User I/O or reference resistor input for bank | 8 **GCLK:** User I/O or global clock buffer input | 12 **VCCO:** Output voltage supply for bank |
| 7 **CONFIG:** Dedicated configuration pins | 4 **JTAG:** Dedicated JTAG port pins | 4 **VCCINT:** Internal core voltage supply (+1.2V) |
| 0 **N.C.:** No unconnected pins in this package | 16 **GND:** Ground | 4 **VCCAUX:** Auxiliary voltage supply (+2.5V) |

## PQ208 Footprint

### *Left Half of Package (top view)*

**XC3S50**
**(124 max. user I/O)**

72 **I/O:** Unrestricted, general-purpose user I/O

16 **VREF:** User I/O or input voltage reference for bank

17 **N.C.:** Unconnected pins for XC3S50 (◆)

**XC3S200, XC3S400**
**(141 max user I/O)**

83 **I/O:** Unrestricted, general-purpose user I/O

22 **VREF:** User I/O or input voltage reference for bank

0 **N.C.:** No unconnected pins in this package

**All devices**

12 **DUAL:** Configuration pin, then possible user I/O

8 **GCLK:** User I/O or global clock buffer input

16 **DCI:** User I/O or reference resistor input for bank

7 **CONFIG:** Dedicated configuration pins

4 **JTAG:** Dedicated JTAG port pins

4 **VCCINT:** Internal core voltage supply (+1.2V)

12 **VCCO:** Output voltage supply for bank

8 **VCCAUX:** Auxiliary voltage supply (+2.5V)

28 **GND:** Ground

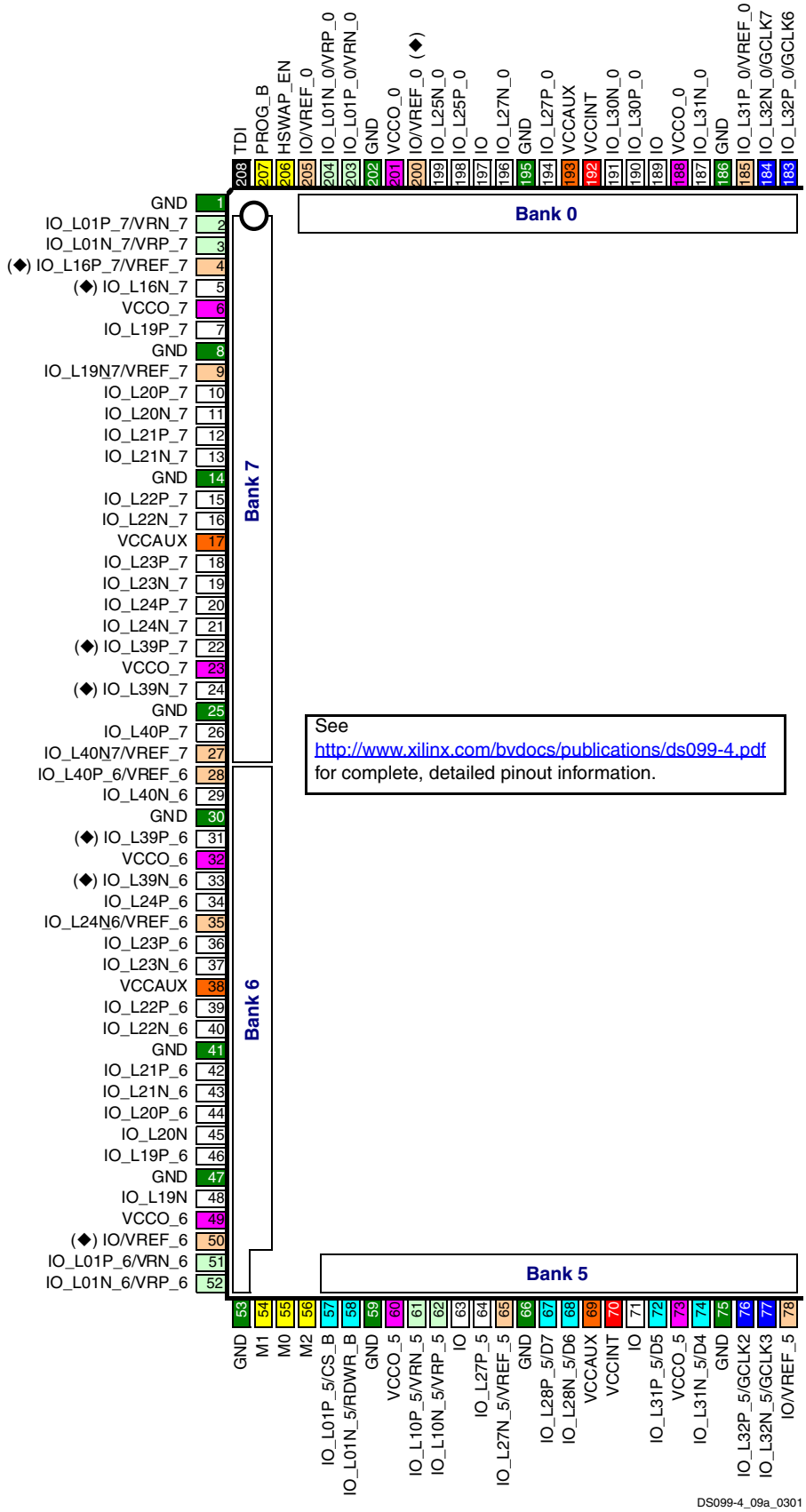
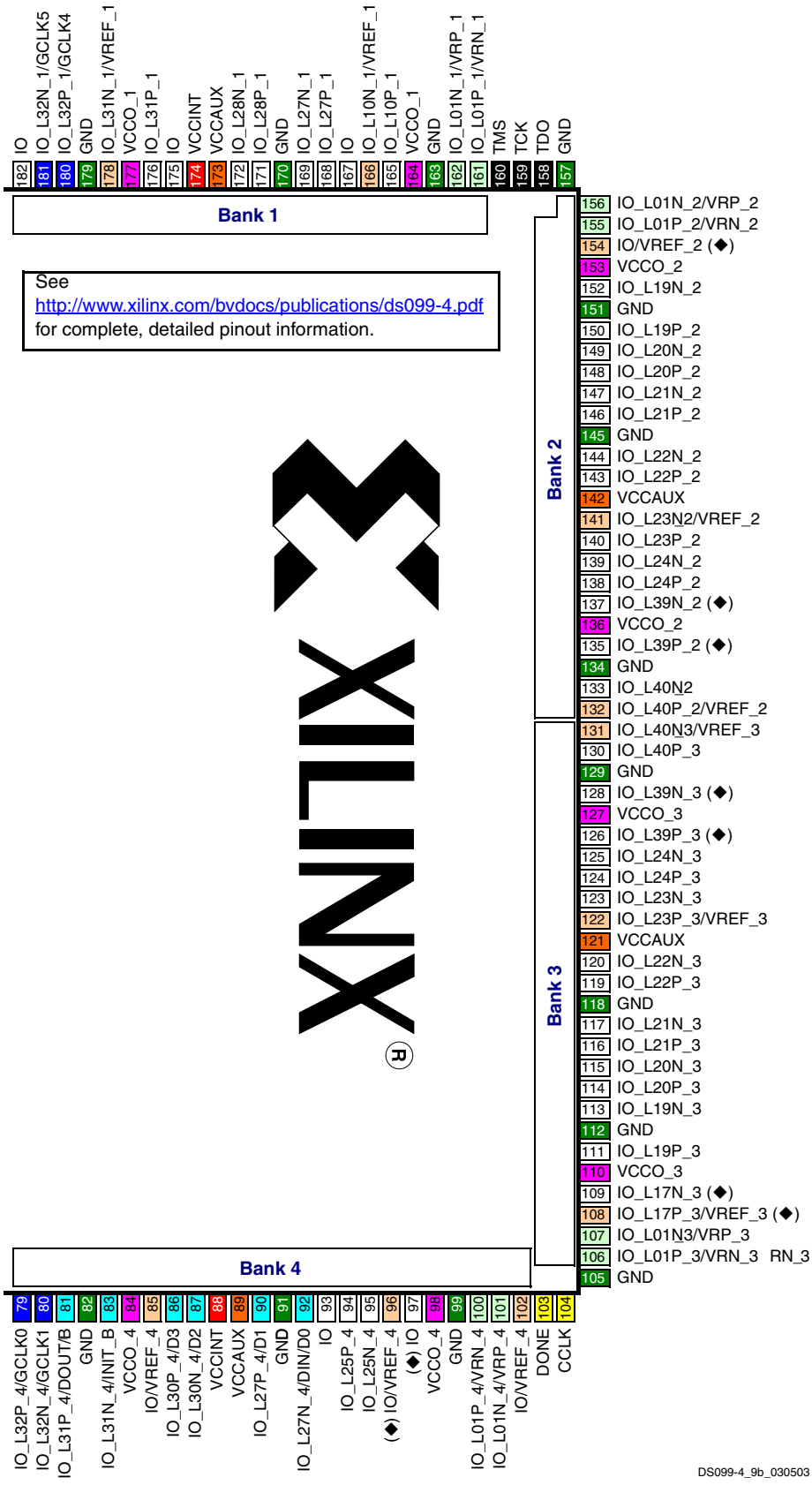



*Figure 10:* **PQ208 Package Footprint (top view). Note pin 1 indicator in top-left corner and logo orientation.**

***Right Half of Package (top view)***




See http://www.xilinx.com/bvdocs/publications/ds099-4.pdf for complete, detailed pinout information.

## FT256 Footprint

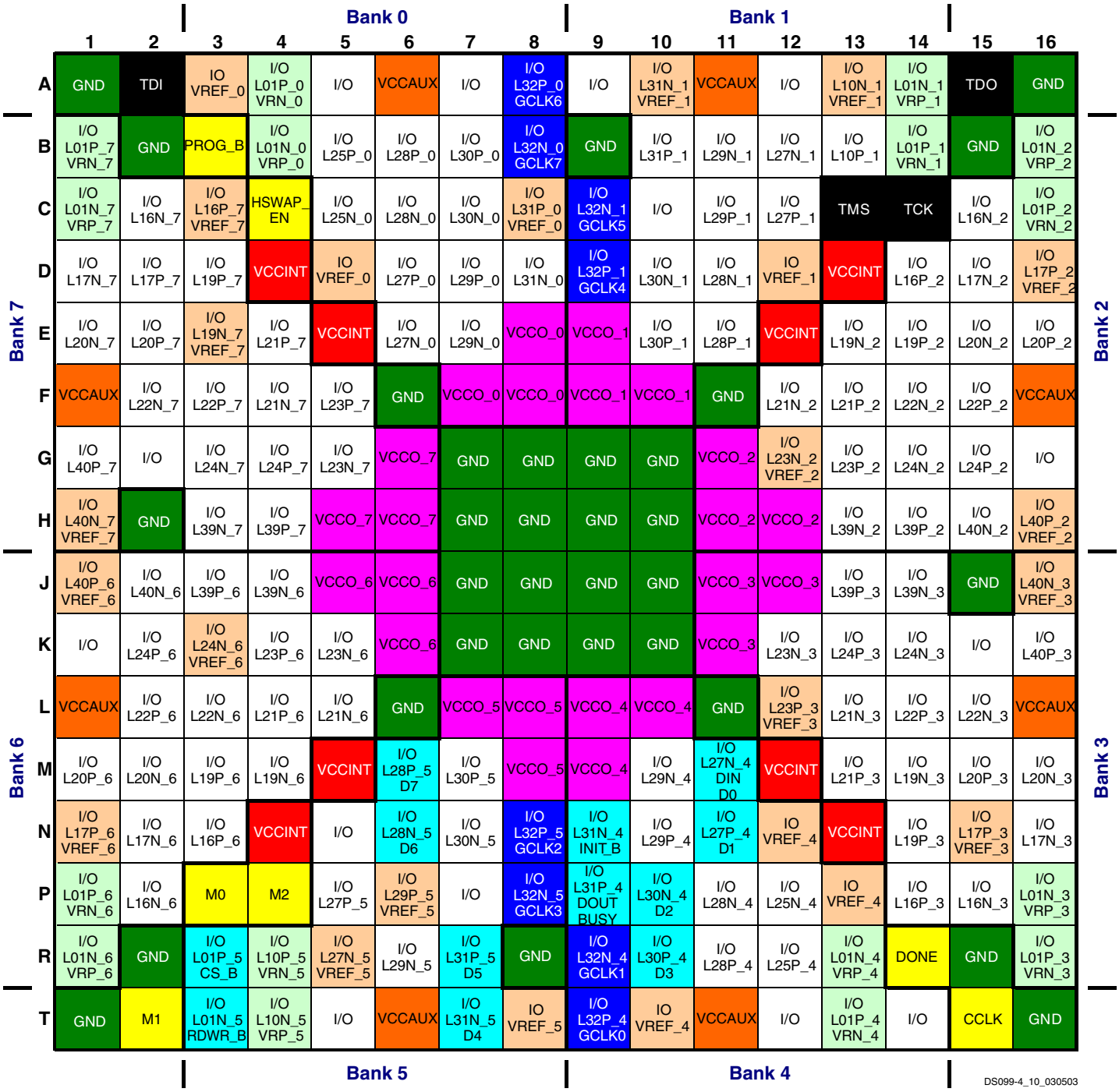



*Figure 11:* **FT256 Package Footprint (top view)**

| | | |
|---|---|---|
| 113 **I/O:** Unrestricted, general-purpose user I/O | 12 **DUAL:** Configuration pin, then possible user I/O | 24 **VREF:** User I/O or input voltage reference for bank |
| 16 **DCI:** User I/O or reference resistor input for bank | 8 **GCLK:** User I/O or global clock buffer input | 24 **VCCO:** Output voltage supply for bank |
| 7 **CONFIG:** Dedicated configuration pins | 4 **JTAG:** Dedicated JTAG port pins | 8 **VCCINT:** Internal core voltage supply (+1.2V) |
| 0 **N.C.:** No unconnected pins in this package | 32 **GND:** Ground | 8 **VCCAUX:** Auxiliary voltage supply (+2.5V) |

## FG456 Footprint

### *Left Half of Package (top view)*

**XC3S400**
**(264 max. user I/O)**

196 **I/O:** Unrestricted, general-purpose user I/O

32 **VREF:** User I/O or input voltage reference for bank

69 **N.C.:** Unconnected pins for XC3S400 (◆)

**XC3S1000, XC3S1500**
**(333 max user I/O)**

261 **I/O:** Unrestricted, general-purpose user I/O

36 **VREF:** User I/O or input voltage reference for bank

0 **N.C.:** No unconnected pins in this package

**All devices**

12 **DUAL:** Configuration pin, then possible user I/O

8 **GCLK:** User I/O or global clock buffer input

16 **DCI:** User I/O or reference resistor input for bank

7 **CONFIG:** Dedicated configuration pins

4 **JTAG:** Dedicated JTAG port pins

12 **VCCINT:** Internal core voltage supply (+1.2V)

40 **VCCO:** Output voltage supply for bank

8 **VCCAUX:** Auxiliary voltage supply (+2.5V)

52 **GND:** Ground

See http://www.xilinx.com/bvdocs/publications/ds099-4.pdf for complete, detailed pinout information.

| | Bank 0 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| A | GND | PROG_B | IO VREF_0 | I/O L01P_0 VRN_0 | I/O L09P_0 | VCCAUX | I/O L19P_0 ◆ | I/O L24P_0 | I/O L27P_0 | I/O | I/O L32P_0 GCLK6 |
| B | TDI | GND | HSWAP_EN | I/O L01N_0 VRP_0 | I/O L09N_0 | I/O L15P_0 | I/O L19N_0 ◆ | I/O L24N_0 | I/O L27N_0 | I/O L29P_0 | I/O L32N_0 GCLK7 |
| C | I/O L16P_7 VREF_7 | I/O | I/O L01N_7 VRP_7 | I/O L01P_7 VRN_7 | I/O L06P_0 | I/O L15N_0 | IO VREF_0 | VCCO_0 | GND | I/O L29N_0 | I/O L31P_0 VREF_0 |
| D | I/O L16N_7 | I/O L19P_7 | I/O L19N_7 VREF_7 | I/O L17P_7 | I/O L06N_0 | I/O L10P_0 | I/O L16P_0 | I/O L22P_0 ◆ | I/O | I/O | I/O L31N_0 |
| E | I/O L21N_7 | I/O L21P_7 | I/O L20P_7 | I/O L17N_7 | IO VREF_0 ◆ | I/O L10N_0 | I/O L16N_0 | I/O L22N_0 ◆ | I/O L25P_0 | I/O L28P_0 | I/O L30P_0 |
| F | VCCAUX | I/O L23N_7 | I/O L23P_7 | I/O L20N_7 | I/O L22P_7 | I/O | IO VREF_0 | VCCO_0 | I/O L25N_0 | I/O L28N_0 | I/O L30N_0 |
| G | I/O L27N_7 | I/O L27P_7 VREF_7 | I/O L26N_7 ◆ | I/O L26P_7 ◆ | I/O L24P_7 | I/O L22N_7 | VCCINT | VCCINT | VCCO_0 | VCCO_0 | VCCO_0 |
| H | I/O L28N_7 ◆ | I/O L28P_7 ◆ | VCCO_7 | I/O L29P_7 ◆ | I/O L24N_7 | VCCO_7 | VCCINT | | | | |
| J | I/O L32N_7 ◆ | I/O L32P_7 ◆ | GND | I/O L29N_7 ◆ | I/O L31N_7 ◆ | I/O L31P_7 ◆ | VCCO_7 | | GND | GND | GND |
| K | I/O L35N_7 | I/O L35P_7 | I/O L34N_7 | I/O L34P_7 | I/O L33N_7 ◆ | I/O L33P_7 ◆ | VCCO_7 | | GND | GND | GND |
| L | I/O L40N_7 VREF_7 | I/O L40P_7 | I/O L39N_7 | I/O L39P_7 | I/O L38N_7 | I/O L38P_7 | VCCO_7 | | GND | GND | GND |
| M | I/O L40P_6 VREF_6 | I/O L40N_6 | I/O L39P_6 | I/O L39N_6 | I/O L38P_6 | I/O L38N_6 | VCCO_6 | | GND | GND | GND |
| N | I/O L35P_6 | I/O L35N_6 | I/O L34P_6 | I/O L34N_6 VREF_6 | I/O L33P_6 ◆ | I/O L33N_6 ◆ | VCCO_6 | | GND | GND | GND |
| P | I/O L32P_6 ◆ | I/O L32N_6 ◆ | GND | I/O L31P_6 ◆ | I/O L31N_6 ◆ | I/O L28P_6 ◆ | VCCO_6 | | GND | GND | GND |
| R | I/O L29P_6 ◆ | I/O L29N_6 ◆ | VCCO_6 | I/O L26P_6 ◆ | I/O L28N_6 ◆ | VCCO_6 | VCCINT | | | | |
| T | I/O L27P_6 | I/O L27N_6 | I/O L26N_6 ◆ | I/O L23P_6 | I/O L22P_6 | I/O L22N_6 | VCCINT | VCCINT | VCCO_5 | VCCO_5 | VCCO_5 |
| U | VCCAUX | I/O L24P_6 | I/O L24N_6 VREF_6 | I/O L23N_6 | I/O L19P_6 | IO VREF_5 | I/O | VCCO_5 | I/O ◆ | I/O | I/O |
| V | I/O L21P_6 | I/O L21N_6 | I/O L20P_6 | I/O L20N_6 | I/O L19N_6 | I/O L15P_5 | I/O | I/O L24P_5 | I/O L27P_5 | I/O | I/O L31P_5 D5 |
| W | I/O L17P_6 VREF_6 | I/O L17N_6 | I/O L16P_6 | I/O L16N_6 | I/O L09P_5 | I/O L15N_5 | I/O L19P_5 VREF_5 ◆ | I/O L24N_5 | I/O L27N_5 VREF_5 | I/O L29P_5 VREF_5 | I/O L31N_5 D4 |
| Y | I/O | I/O L01P_6 VRN_6 | I/O L01N_6 VRP_6 | I/O L01N_5 RDWR_B | I/O L09N_5 | I/O L16P_5 | I/O L19N_5 ◆ | VCCO_5 | GND | I/O L29N_5 | I/O L32P_5 GCLK2 |
| AA | M1 | GND | I/O L01P_5 CS_B | I/O L06P_5 | I/O L10P_5 VRN_5 | I/O L16N_5 | I/O L22P_5 ◆ | I/O L25P_5 | I/O L28P_5 D7 | I/O L30P_5 | I/O L32N_5 GCLK3 |
| AB | GND | M0 | M2 | I/O L06N_5 | I/O L10N_5 VRP_5 | VCCAUX | I/O L22N_5 ◆ | I/O L25N_5 | I/O L28N_5 D6 | I/O L30N_5 | IO VREF_5 |

Bank 7 (rows C–L) · Bank 6 (rows M–T) · Bank 5 (rows U–AB)

DS099-4_11a_030203

*Figure 12:* **FG456 Package Footprint (top view)**

***Right Half of Package (top view)***

| | Bank 1 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| A | I/O | I/O L30N_1 | I/O L28N_1 | I/O L25P_1 | I/O L22N_1 ◆ | VCCAUX | I/O L10N_1 VREF_1 | I/O L06N_1 VREF_1 | TMS | TCK | GND |
| B | I/O L32N_1 GCLK5 | I/O L30P_1 | I/O L28P_1 | I/O L25N_1 | I/O L22P_1 ◆ | I/O L16N_1 | I/O L10P_1 | I/O L06P_1 | I/O L01P_1 VRN_1 | GND | TDO |
| C | I/O L32P_1 GCLK4 | I/O L29N_1 | GND | VCCO_1 | I/O L19N_1 ◆ | I/O L16P_1 | I/O L09N_1 | I/O L01N_1 VRP_1 | I/O L01N_2 VRP_2 | I/O L01P_2 VRN_2 | I/O |
| D | I/O L31N_1 VREF_1 | I/O L29P_1 | I/O L27N_1 | I/O L24N_1 | I/O L19P_1 ◆ | I/O L15N_1 | I/O L09P_1 | I/O L16P_2 | I/O L16N_2 | I/O L17N_2 | I/O L17P_2 VREF_2 |
| E | I/O L31P_1 | IO VREF_1 | I/O L27P_1 | I/O L24P_1 | I/O | I/O L15P_1 | I/O L19N_2 | I/O L20N_2 | I/O L20P_2 | I/O L21N_2 | I/O L21P_2 |
| F | I/O | I/O | IO VREF_1 ◆ | VCCO_1 | I/O | I/O | I/O L19P_2 | I/O L23N_2 VREF_2 | I/O L24N_2 | I/O L24P_2 | VCCAUX |
| G | VCCO_1 | VCCO_1 | VCCO_1 | VCCINT | VCCINT | I/O L22N_2 | I/O L22P_2 | I/O L23P_2 | I/O L26N_2 ◆ | I/O L27N_2 | I/O L27P_2 |
| H | | | | | VCCINT | VCCO_2 | I/O L28N_2 ◆ | I/O L26P_2 ◆ | VCCO_2 | I/O L29N_2 ◆ | I/O L29P_2 ◆ |
| J | GND | GND | GND | | VCCO_2 | I/O L28P_2 ◆ | I/O L31N_2 ◆ | I/O L31P_2 ◆ | GND | I/O L32N_2 ◆ | I/O L32P_2 ◆ |
| K | GND | GND | GND | | VCCO_2 | I/O L33N_2 ◆ | I/O L33P_2 ◆ | I/O L34N_2 VREF_2 | I/O L34P_2 | I/O L35N_2 | I/O L35P_2 |
| L | GND | GND | GND | | VCCO_2 | I/O L38N_2 | I/O L38P_2 | I/O L39N_2 | I/O L39P_2 | I/O L40N_2 | I/O L40P_2 VREF_2 |
| M | GND | GND | GND | | VCCO_3 | I/O L38P_3 | I/O L38N_3 | I/O L39P_3 | I/O L39N_3 | I/O L40P_3 | I/O L40N_3 VREF_3 |
| N | GND | GND | GND | | VCCO_3 | I/O L33P_3 ◆ | I/O L33N_3 ◆ | I/O L34P_3 VREF_3 | I/O L34N_3 | I/O L35P_3 | I/O L35N_3 |
| P | GND | GND | GND | | VCCO_3 | I/O L31P_3 ◆ | I/O L31N_3 ◆ | I/O L29N_3 ◆ | GND | I/O L32P_3 ◆ | I/O L32N_3 ◆ |
| R | | | | | VCCINT | VCCO_3 | I/O L24N_3 | I/O L29P_3 ◆ | VCCO_3 | I/O L28P_3 ◆ | I/O L28N_3 ◆ |
| T | VCCO_4 | VCCO_4 | VCCO_4 | VCCINT | VCCINT | I/O L22N_3 | I/O L24P_3 | I/O L26P_3 ◆ | I/O L26N_3 ◆ | I/O L27P_3 | I/O L27N_3 |
| U | I/O L30N_4 D2 | I/O L28N_4 | I/O L25N_4 | VCCO_4 | I/O | I/O | I/O L22P_3 | I/O L20N_3 | I/O L23P_3 VREF_3 | I/O L23N_3 | VCCAUX |
| V | I/O L30P_4 D3 | I/O L28P_4 | I/O L25P_4 | I/O L22N_4 VREF_4 ◆ | I/O L16N_4 | I/O L10N_4 | IO VREF_4 | I/O L17N_3 | I/O L20P_3 | I/O L21P_3 | I/O L21N_3 |
| W | I/O L31N_4 INIT_B | I/O | I/O | I/O L22P_4 ◆ | I/O L16P_4 | I/O L10P_4 | I/O L06N_4 VREF_4 | I/O L17P_3 VREF_3 | I/O L19P_3 | I/O L19N_3 | I/O L16N_3 |
| Y | I/O L31P_4 DOUT BUSY | I/O L29N_4 | GND | VCCO_4 | IO VREF_4 | I/O L15N_4 | I/O L06P_4 | I/O L01P_3 VRN_3 | I/O L01N_3 VRP_3 | I/O | I/O L16P_3 |
| AA | I/O L32N_4 GCLK1 | I/O L29P_4 | I/O L27N_4 DN D0 | I/O L24N_4 | I/O L19N_4 ◆ | I/O L15P_4 | I/O L09N_4 | I/O L05N_4 ◆ | I/O L01N_4 VRP_4 | GND | CCLK |
| AB | I/O L32P_4 GCLK0 | IO VREF_4 | I/O L27P_4 D1 | I/O L24P_4 | I/O L19P_4 ◆ | VCCAUX | I/O L09P_4 | I/O L05P_4 ◆ | I/O L01P_4 VRN_4 | DONE | GND |

Bank 2 (rows G–L) | Bank 3 (rows M–AB)

Bank 4

DS099-4_11b_030503

See http://www.xilinx.com/bvdocs/publications/ds099-4.pdf for complete, detailed pinout information.

## FG676 Footprint

### *Left Half of Package (top view)*

**XC3S1000**
**(391 max. user I/O)**

315 **I/O:** Unrestricted, general-purpose user I/O

40 **VREF:** User I/O or input voltage reference for bank

98 **N.C.:** Unconnected pins for XC3S1000 (◆)

**XC3S1500**
**(487 max user I/O)**

403 **I/O:** Unrestricted, general-purpose user I/O

48 **VREF:** User I/O or input voltage reference for bank

2 **N.C.:** Unconnected pins for XC3S1500 (■)

**XC3S2000**
**(489 max user I/O)**

405 **I/O:** Unrestricted, general-purpose user I/O

48 **VREF:** User I/O or input voltage reference for bank

0 **N.C.:** No unconnected pins

**All devices**

12 **DUAL:** Configuration pin, then possible user I/O

8 **GCLK:** User I/O or global clock buffer input

16 **DCI:** User I/O or reference resistor input for bank

7 **CONFIG:** Dedicated configuration pins

4 **JTAG:** Dedicated JTAG port pins

20 **VCCINT:** Internal core voltage supply (+1.2V)

64 **VCCO:** Output voltage supply for bank

16 **VCCAUX:** Auxiliary voltage supply (+2.5V)

76 **GND:** Ground

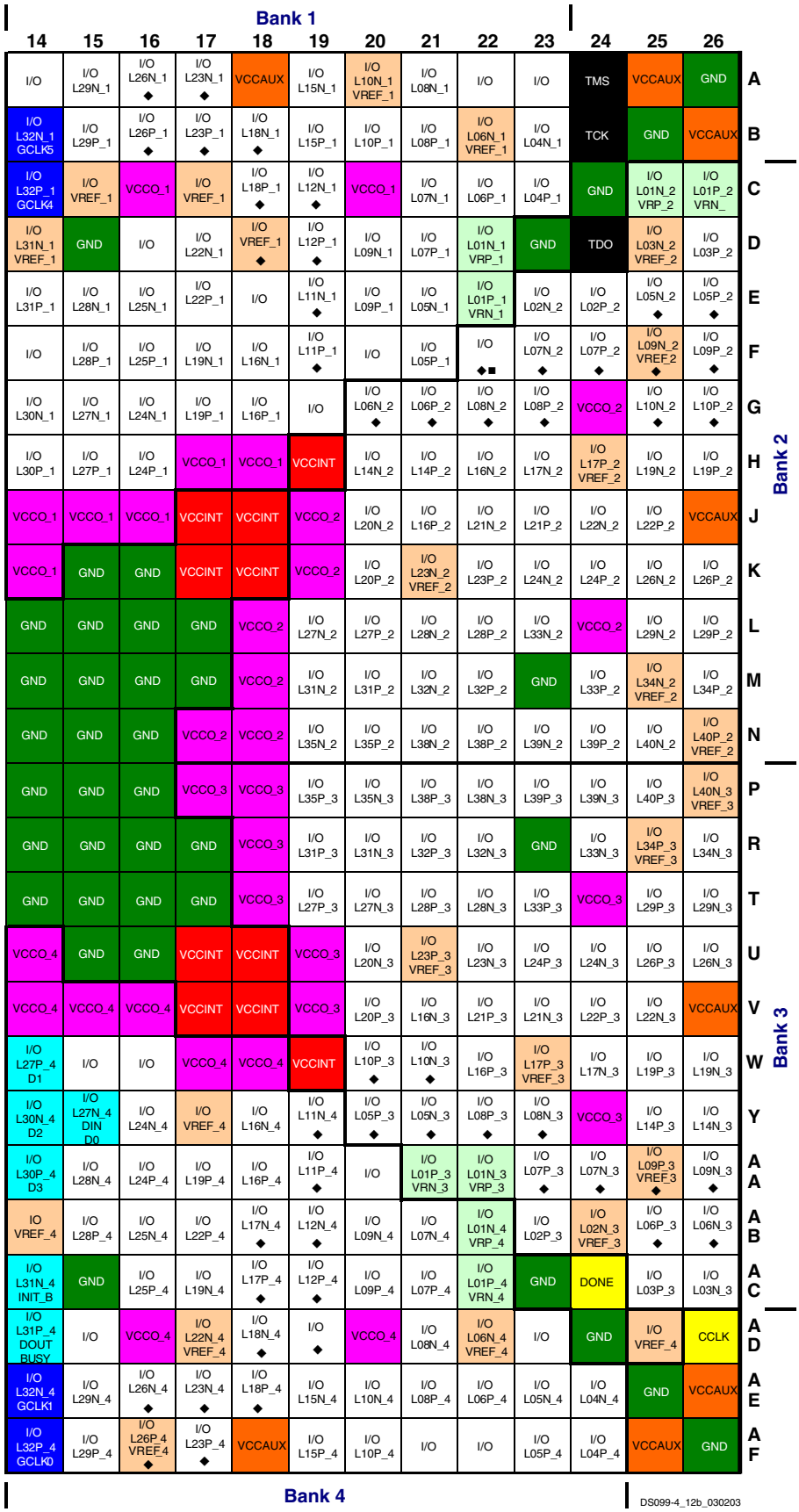
See http://www.xilinx.com/bvdocs/publications/ds099-4.pdf for complete, detailed pinout information.

| | Bank 0 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| A | GND | VCCAUX | I/O | I/O L05P_0 VREF_0 | I/O | I/O | I/O L10P_0 | I/O L15P_0 | VCCAUX | I/O L23P_0 ◆ | I/O L26P_0 VREF_0 ◆ | I/O L29P_0 | I/O L32P_0 GCLK6 |
| B | VCCAUX | GND | I/O VREF_0 | I/O L05N_0 | I/O L06P_0 | I/O L08P_0 | I/O L10N_0 | I/O L15N_0 | I/O L18P_0 ◆ | I/O L23N_0 ◆ | I/O L26N_0 ◆ | I/O L29N_0 | I/O L32N_0 GCLK7 |
| C | TDI | HSWAP_EN | GND | I/O | I/O L06N_0 | I/O L08N_0 | VCCO_0 | I/O ◆ | I/O L18N_0 ◆ | I/O L22P_0 | VCCO_0 | I/O | I/O L31P_0 VREF_0 |
| D | I/O L03N_7 VREF_7 | I/O L03P_7 | PROG_B | GND | I/O L01P_0 VRN_0 | I/O L07P_0 | I/O L09P_0 | I/O L12P_0 ◆ | I/O L17P_0 ◆ | I/O L22N_0 | I/O L25P_0 | GND | I/O L31N_0 |
| E | I/O L06N_7 ◆ | I/O L06P_7 ◆ | I/O L02N_7 | I/O L02P_7 | I/O L01N_0 VRP_0 | I/O L07N_0 | I/O L09N_0 | I/O L12N_0 ◆ | I/O L17N_0 ◆ | I/O L19P_0 | I/O L25N_0 | I/O L28P_0 | I/O |
| F | I/O L09N_7 ◆ | I/O L09P_7 ◆ | I/O L07N_7 ◆ | I/O L07P_7 ◆ | I/O L01N_7 VRP_7 | I/O L01P_7 VRN_7 | I/O VREF_0 | I/O L11P_0 ◆ | I/O L16P_0 | I/O L19N_0 | I/O L24P_0 | I/O L28N_0 | I/O L30P_0 |
| G | I/O L14N_7 | I/O L14P_7 | VCCO_7 | I/O L08N_7 ◆ | I/O L08P_7 ◆ | I/O L05N_7 ◆ | I/O L05P_7 ◆ | I/O L11N_0 ◆ | I/O L16N_0 | I/O VREF_0 | I/O L24N_0 | I/O L27N_0 | I/O L30N_0 |
| H | I/O L19N_7 VREF_7 | I/O L19P_7 | I/O L17N_7 | I/O L17P_7 | I/O L16P_7 VREF_7 | I/O L10N_7 ◆ | I/O L10P_7 VREF_7 ◆ | VCCINT | VCCO_0 | VCCO_0 | I/O | I/O | I/O L27P_0 |
| J | VCCAUX | I/O L22N_7 | I/O L22P_7 | I/O L21N_7 | I/O L21P_7 | I/O L16N_7 | I/O L20P_7 | VCCO_7 | VCCINT | VCCINT | VCCO_0 | VCCO_0 | VCCO_0 |
| K | I/O L26N_7 | I/O L26P_7 | I/O L24N_7 | I/O L24P_7 | I/O L23N_7 | I/O L23P_7 | I/O L20N_7 | VCCO_7 | VCCINT | VCCINT | GND | GND | VCCO_0 |
| L | I/O L29N_7 | I/O L29P_7 | VCCO_7 | I/O L33P_7 | I/O L28N_7 | I/O L28P_7 | I/O L27N_7 | I/O L27P_7 VREF_7 | VCCO_7 | GND | GND | GND | GND |
| M | I/O L34N_7 | I/O L34P_7 | I/O L33N_7 | GND | I/O L32P_7 | I/O L32N_7 | I/O L31N_7 | I/O L31P_7 | VCCO_7 | GND | GND | GND | GND |
| N | I/O L40N_7 VREF_7 | I/O L40P_7 | I/O L39N_7 | I/O L39P_7 | I/O L38N_7 | I/O L38P_7 | I/O L35N_7 | I/O L35P_7 | VCCO_7 | VCCO_7 | GND | GND | GND |
| P | I/O L40P_6 VREF_6 | I/O L40N_6 | I/O L39P_6 | I/O L39N_6 | I/O L38P_6 | I/O L38N_6 | I/O L35P_6 | I/O L35N_6 | VCCO_6 | VCCO_6 | GND | GND | GND |
| R | I/O L34P_6 | I/O L34N_6 VREF_6 | I/O L33P_6 | GND | I/O L32P_6 | I/O L32N_6 | I/O L31P_6 | I/O L31N_6 | VCCO_6 | GND | GND | GND | GND |
| T | I/O L29P_6 | I/O L29N_6 | VCCO_6 | I/O L33N_6 | I/O L28P_6 | I/O L28N_6 | I/O L27P_6 | I/O L27N_6 | VCCO_6 | GND | GND | GND | GND |
| U | I/O L26P_6 | I/O L26N_6 | I/O L24P_6 | I/O L24N_6 VREF_6 | I/O L23P_6 | I/O L23N_6 | I/O L20P_6 | VCCO_6 | VCCINT | VCCINT | GND | GND | VCCO_5 |
| V | VCCAUX | I/O L22P_6 | I/O L22N_6 | I/O L21P_6 | I/O L21N_6 | I/O L16N_6 | I/O L20N_6 | VCCO_6 | VCCINT | VCCINT | VCCO_5 | VCCO_5 | VCCO_5 |
| W | I/O L19P_6 | I/O L19N_6 | I/O L17P_6 VREF_6 | I/O L17N_6 | I/O L16P_6 | I/O L14P_6 | I/O L14N_6 | VCCINT | VCCO_5 | VCCO_5 | I/O L24P_5 | I/O L27P_5 | I/O L30P_5 |
| Y | I/O L10P_6 ◆ | I/O L10N_6 ◆ | VCCO_6 | I/O L08P_6 ◆ | I/O L08N_6 ◆ | I/O L06P_6 ◆ | I/O L06N_6 ◆ | I/O | I/O L16P_5 | I/O L19P_5 VREF_5 | I/O L24N_5 | I/O L27N_5 VREF_5 | I/O L30N_5 |
| AA | I/O L09P_6 ◆ | I/O L09N_6 VREF_6 ◆ | I/O L07P_6 ◆ | I/O L07N_6 ◆ | I/O ◆■ | I/O L05P_5 | I/O | I/O L11P_5 ◆ | I/O L16N_5 | I/O L19N_5 | I/O L25P_5 | I/O L28P_5 D7 | I/O |
| AB | I/O L05P_6 ◆ | I/O L05N_6 ◆ | I/O L02P_6 | I/O L02N_6 | I/O L01P_5 CS_B | I/O L05N_5 | I/O L09P_5 | I/O L11N_5 VREF_5 ◆ | I/O | I/O L22P_5 | I/O L25N_5 | I/O L28N_5 D6 | I/O L31P_5 D5 |
| AC | I/O L03P_6 | I/O L03N_6 VREF_6 | M1 | GND | I/O L01N_5 RDWR_B | I/O L07P_5 | I/O L09N_5 | I/O L12P_5 ◆ | I/O ◆ | I/O L22N_5 | I/O | GND | I/O L31N_5 D4 |
| AD | I/O L01P_6 VRN_6 | I/O L01N_6 VRP_6 | GND | I/O L04P_5 | I/O L06P_5 | I/O L07N_5 | VCCO_5 | I/O L12N_5 ◆ | I/O L18P_5 ◆ | I/O | VCCO_5 | I/O | I/O L32P_5 GCLK2 |
| AE | VCCAUX | GND | M0 | I/O L04N_5 | I/O L06N_5 | I/O L08P_5 | I/O L10P_5 VRN_5 | I/O L15P_5 | I/O L18N_5 ◆ | I/O L23P_5 ◆ | I/O L26P_5 ◆ | I/O L29P_5 VREF_5 | I/O L32N_5 GCLK3 |
| AF | GND | VCCAUX | M2 | I/O | I/O VREF_5 | I/O L08N_5 | I/O L10N_5 VRP_5 | I/O L15N_5 | VCCAUX | I/O L23N_5 ◆ | I/O L26N_5 ◆ | I/O L29N_5 | I/O VREF_5 |

Bank 7 (rows H–N) Bank 6 (rows P–Y) Bank 5 (bottom)

DS099-4_12a_030203

*Figure 13:* **FG676 Package Footprint (top view)**




## *Right Half of Package (top view)*

See http://www.xilinx.com/bvdocs/publications/ds099-4.pdf for complete, detailed pinout information.

## FG900 Footprint

### *Left Half of Package (top view)*

**XC3S2000**
**(565 max. user I/O)**

481 **I/O:** Unrestricted, general-purpose user I/O

48 **VREF:** User I/O or input voltage reference for bank

68 **N.C.:** Unconnected pins for XC3S2000 (◆)

**XC3S4000, XC3S5000**
**(633 max user I/O)**

549 **I/O:** Unrestricted, general-purpose user I/O

48 **VREF:** User I/O or input voltage reference for bank

0 **N.C.:** No unconnected pins in this package

**All devices**

12 **DUAL:** Configuration pin, then possible user I/O

8 **GCLK:** User I/O or global clock buffer input

16 **DCI:** User I/O or reference resistor input for bank

7 **CONFIG:** Dedicated configuration pins

4 **JTAG:** Dedicated JTAG port pins

32 **VCCINT:** Internal core voltage supply (+1.2V)

80 **VCCO:** Output voltage supply for bank

24 **VCCAUX:** Auxiliary voltage supply (+2.5V)

120 **GND:** Ground

See http://www.xilinx.com/bvdocs/publications/ds099-4.pdf for complete, detailed pinout information.

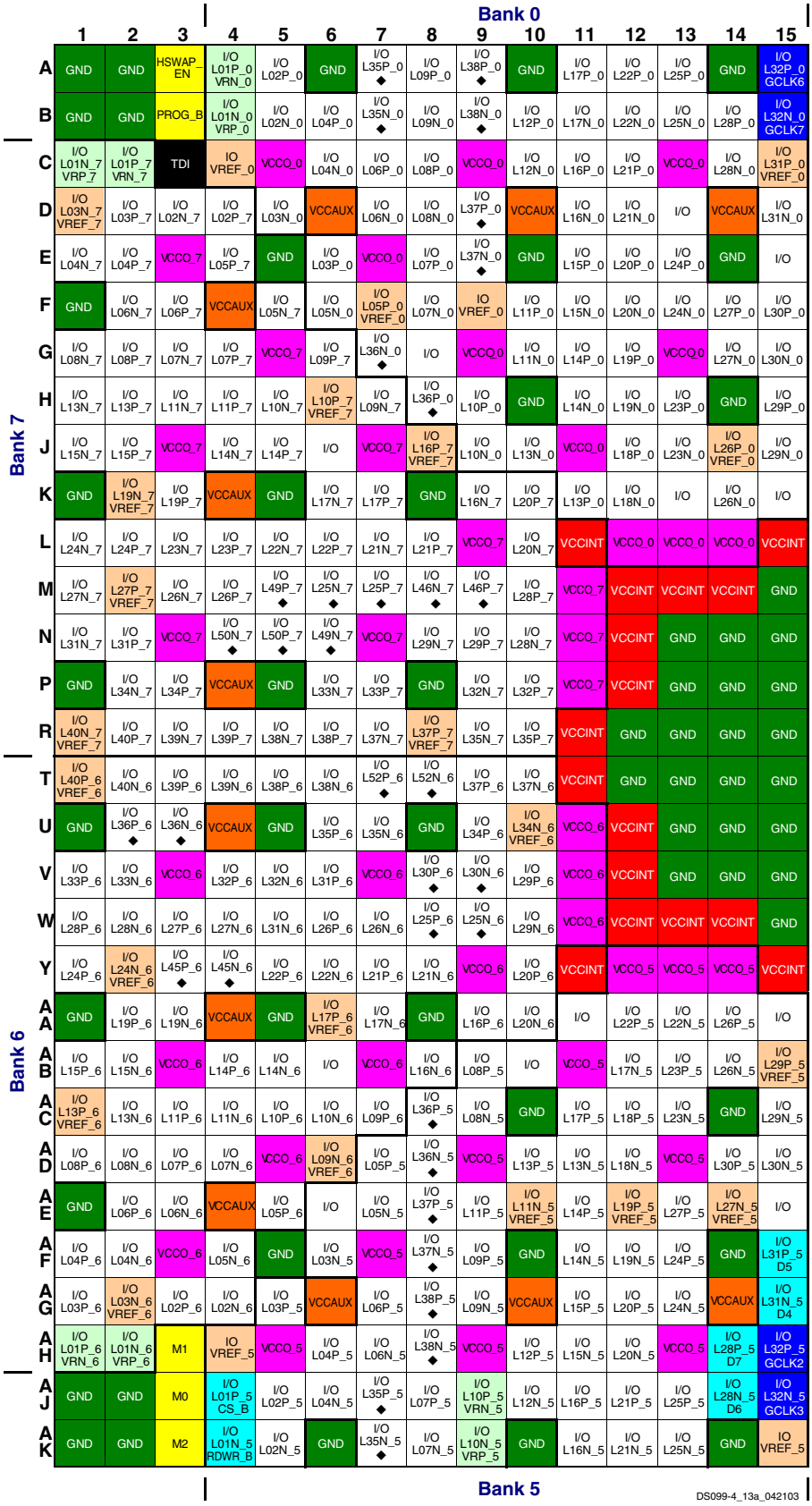
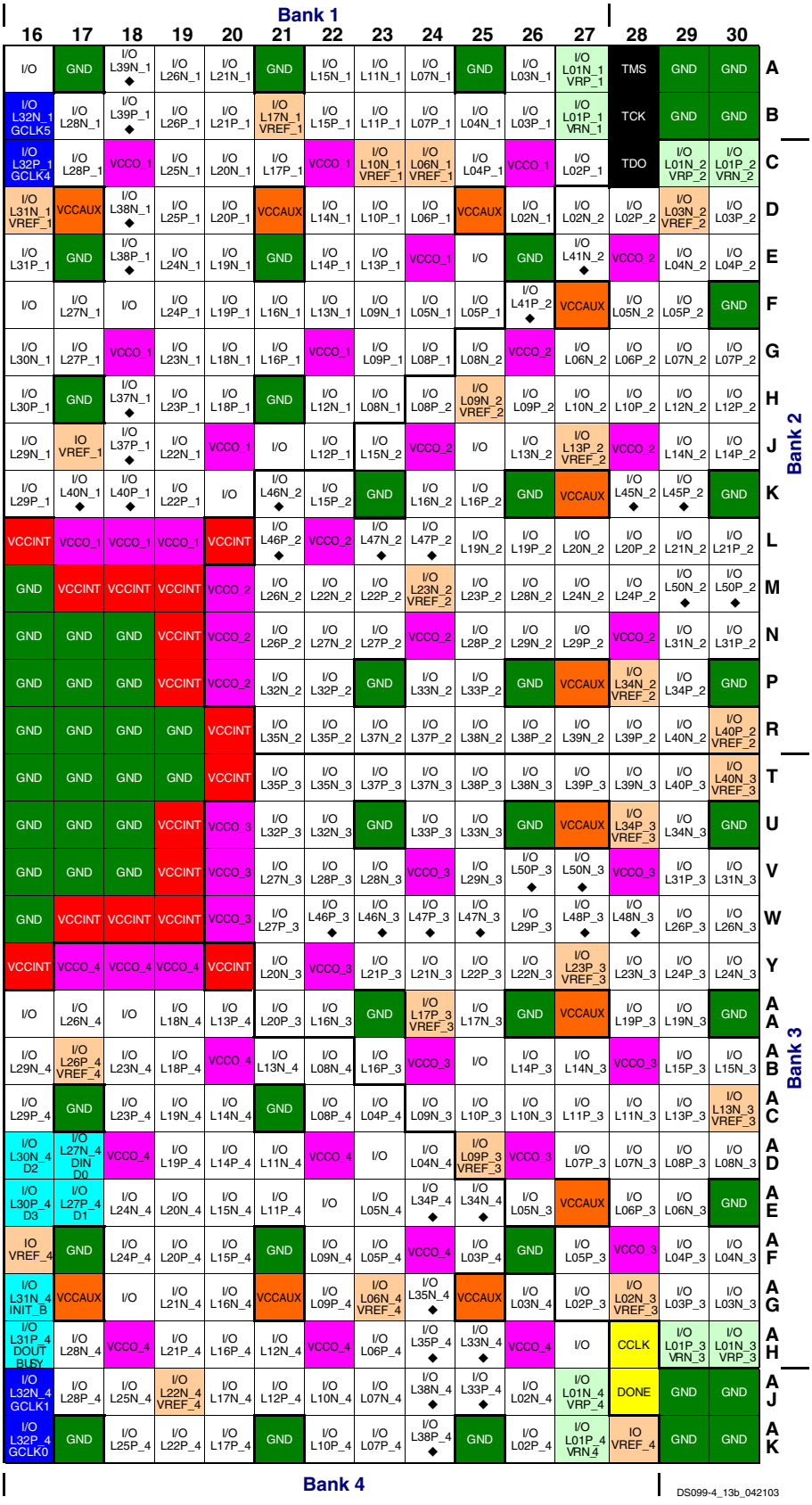



*Figure 14:* **FG900 Package Footprint (top view)**




***Right Half of Package (top view)***

See http://www.xilinx.com/bvdocs/publications/ds099-4.pdf for complete, detailed pinout information.

## Revision History

| Date | Version No. | Description |
|---|---|---|
| 04/03/03 | 1.0 | Initial Xilinx release. |
| 04/21/03 | 1.1 | Added information on the VQ100 package footprint, including a complete pinout table (Table 16) and footprint diagram (Figure 8).<br>Updated Table 15 with final I/O counts for the VQ100 package. Also added final differential I/O pair counts for the TQ144 package.<br>Added clarifying comments to HSWAP_EN pin description on page 83.<br>Updated the footprint diagram for the FG900 package shown in Figure 14a and Figure 14b. Some thick lines separating I/O banks were incorrect.<br>Made cosmetic changes to Figure 1, Figure 3, and Figure 4.<br>Updated Xilinx hypertext links.<br>Added XC3S200 and XC3S400 to Pin Name column in Table 18. |
| 05/12/03 | 1.1.1 | AM32 pin was missing GND label in FG1156 package diagram (Figure 15). |
| 07/11/03 | 1.1.2 | Corrected misspellings of GCLK in Table 1 and Table 2. Changed CMOS25 to LVCMOS25 in **Dual-Purpose Pin I/O Standard During Configuration** section. Clarified references to Module 2. For XC3S5000 in FG1156 package, corrected N.C. symbol to a black square in key and package drawing. |

## The Spartan-3 Family Data Sheet

**DS099-1**, ***Spartan-3 1.2V FPGA Family: Introduction and Ordering Information*** (Module 1)

***DS099-2, Spartan-3 1.2V FPGA Family: Functional Description*** (Module 2)

**DS099-3, *Spartan-3 1.2V FPGA Family: DC and Switching Characteristics*** (Module 3)

**DS099-4**, ***Spartan-3 1.2V FPGA Family: Pinout Tables*** (Module 4)

# Designing with Spartan-3 FPGAs


SPARTAN™-3
FPGA
MAKE IT YOUR ASIC

XAPP462 (v1.0) July 11, 2003

# Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs


## Summary

Digital Clock Managers (DCMs) provide advanced clocking capabilities to Spartan™-3 FPGA applications. DCMs optionally multiply or divide the incoming clock frequency to synthesize a new clock frequency. DCMs also eliminate clock skew, thereby improving system performance. Similarly, a DCM optionally phase shifts the clock output to delay the incoming clock by a fraction of the clock period. The DCMs integrate directly with the FPGA's global low-skew clock distribution network.


## Introduction

DCMs integrate advanced clocking capabilities into the Spartan-3 global clock distribution network. Consequently, Spartan-3 DCMs solve a variety of common clocking issues, especially in high-performance, high frequency applications:

- **Multiply or Divide an Incoming Clock Frequency** or synthesize a completely new frequency by a mixture of clock multiplication and division.
- **Condition a Clock,** ensuring a clean output clock with a 50% duty cycle.
- **Phase Shift** a clock signal, either by a fixed fraction of a clock period or by precise increments.
- **Eliminate Clock Skew**, either within the device or to external components, to improve overall system performance and to eliminate clock distribution delays.
- **Mirror, Forward, or Rebuffer a Clock Signal**, often to deskew and convert the incoming clock signal to a different I/O standard—for example, forwarding and converting an incoming LVTTL clock to LVDS.
- **Any or all the above functions, simultaneously**.

*Table 1:* **Digital Clock Manager Features and Capabilities**

| Feature | Description | DCM Signals |
|---|---|---|
| Digital Clock Managers (DCMs) per device | • 4, except in XC3S50<br>• 2 in XC3S50 | All |
| Digital Frequency Synthesizer (DFS) Input Frequency Range* | 1 MHz to ~326 MHz | CLKIN |
| Delay-Locked Loop (DLL) Input Frequency Range* | 24 MHz to ~326 MHz | CLKIN |
| Clock Input Sources | • Global buffer input pad<br>• Global buffer output<br>• General-purpose I/O (no deskew)<br>• Internal logic (no deskew) | CLKIN |
| Frequency Synthesizer Output | Multiply CLKIN by the fraction (M/D) where M={2..32}, D={1..32} | • CLKFX<br>• CLKFX180 |

*Table 1:* **Digital Clock Manager Features and Capabilities *(Continued)***

| Feature | Description | DCM Signals |
|---|---|---|
| Clock Divider Output | Divide CLKIN by 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 | CLKDV |
| Clock Doubler Output | Multiply CLKIN frequency by 2 | • CLK2X<br>• CLK2X180 |
| Clock Conditioning, Duty-Cycle Correction | Always provided on most outputs. Optional on CLK0, CLK90, CLK180, CLK270. 50% duty cycle ± 100 ps* | All |
| Quadrant Phase Shift Outputs | 0° (no phase shift),<br>90° (¼ period),<br>180° (½ period),<br>270° (¾ period) | • CLK0<br>• CLK90<br>• CLK180<br>• CLK270 |
| Half-period Phase Shift Outputs | Output pairs with 0º and 180º phase shift, ideal for DDR applications | • CLK0, CLK180<br>• CLK2X, CLK2X180<br>• CLKFX, CLKFX180 |
| Dynamic or Fixed Phase Shift resolution | Down to 1/256$^{th}$ of a clock period (or ~30 to 50 ps)* | All |
| Number of Clock Outputs to General-purpose Interconnect | Up to all 9 | All |
| Number of Clock Outputs to Global Clock Network | Any 4 of 9 | All |
| Number of Clock Outputs to Output Pins | Up to all 9 | All |

* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

## Document Overview

This application note covers an assortment of topics related to Digital Clock Managers, not all of which are relevant to every specific FPGA application.

The "DCM Functional Overview" section provides a brief introduction to the DCM and its functions. Similarly the "DCM Primitive" section describes all the connection ports and attributes or constraints associated with a DCM. Likewise the "DCM Wizard" and the "VHDL and Verilog Instantiation" sections demonstrate the various methods to specify a DCM design.

The "DCM Clock Requirements" and the "Input and Output Clock Frequency Restrictions" sections explain the frequency requirements on the DCM clock input and the various DCM clock outputs. Similarly, the "Clock Jitter or Phase Noise" section highlights the effect jitter has on output clock quality.

Finally, the "Eliminating Clock Skew", "Clock Conditioning", "Phase Shifting – Delaying the Clock by a Fraction of a Period", "Clock Multiplication, Clock Division, and Frequency Synthesis", and "Clock Forwarding, Mirroring, Rebuffering" sections illustrate various applications using the DCM block.

## DCM Locations and Clock Distribution Network Interface

As shown in Figure 1, most Spartan-3 FPGAs have four DCM blocks, except for the XC3S50, which has two DCM blocks. The DCM blocks are located at the top and bottom of the block RAM/multiplier columns along the left and right edges. The XC3S50 has two DCMs, along the top and bottom of the block RAM/multiplier column along the left edge of the device.

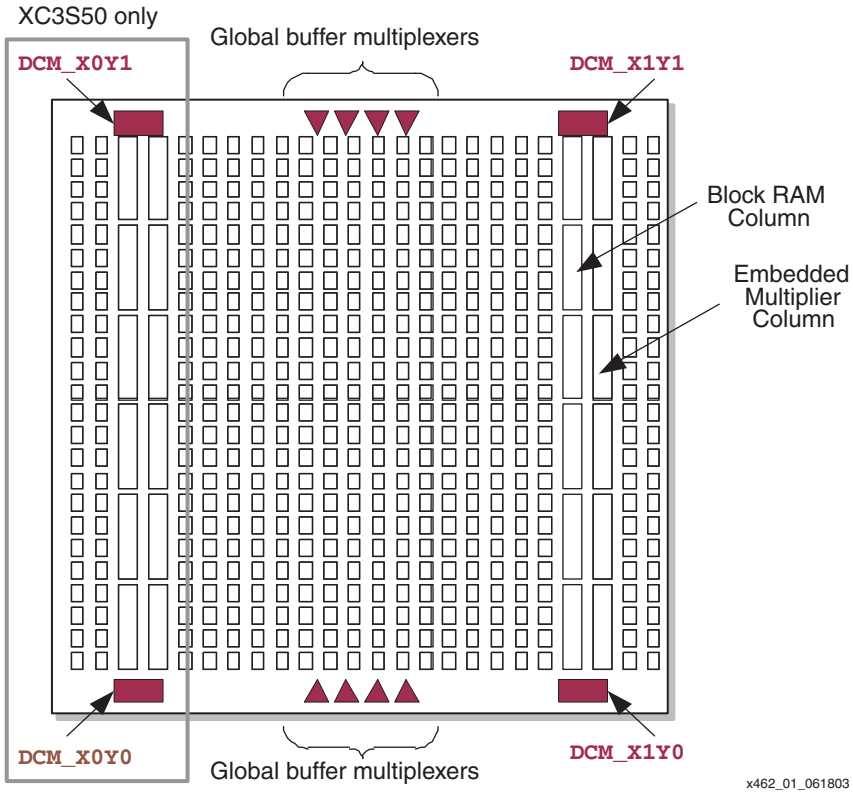
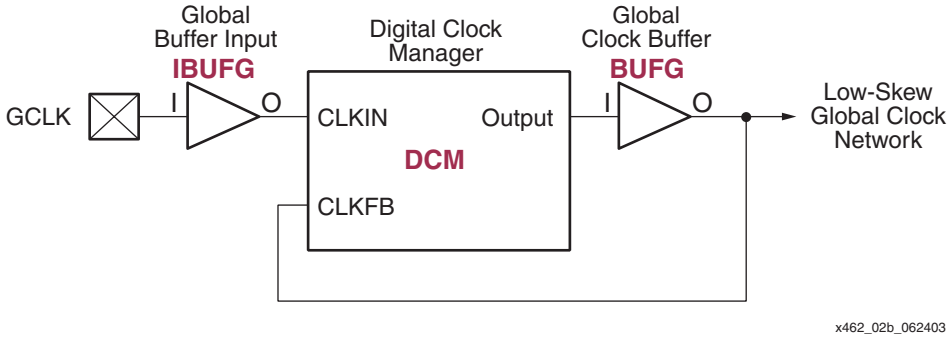



*Figure 1:* **Location of the Four DCM Blocks on Spartan-3 FPGAs**

The DCM blocks have dedicated connections to the global buffer inputs and global buffer multiplexers on the same edge of the device, either top or bottom. As shown in Figure 2, DCMs are an integral part of the FPGA's global clocking infrastructure. DCMs are an optional element in the clock distribution network and are available when required by the application. In Figure 2a, a clock input feeds directly into the low-skew, high-fanout global clock network via a global input buffer and global clock buffer.

If the application requires some or all of the DCM's advanced clocking features, the DCM fits neatly between the global buffer input and the buffer itself, as shown in Figure 2b.


**a. Global Buffer Inputs and Clock Buffers Drive a Low-Skew Global Network in the FPGA**




**b. A Digital Clock Manager (DCM) Inserts Directly into the Global Clock Path**

*Figure 2:* **DCMs are an Integral Part of the FPGA's Global Clock Network**

## DCM Functional Overview

The single entity called a Digital Clock Manager (DCM) actually consists of four distinct functional units as depicted in Figure 3 and described below. These units operate independently or in tandem.

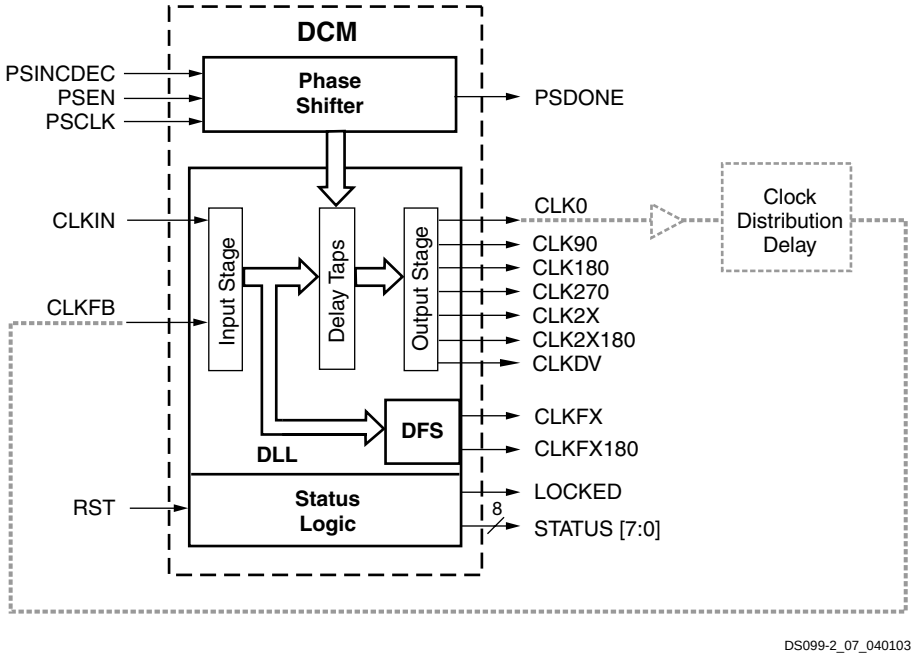



*Figure 3:* **DCM Functional Block Diagram**

## Delay-Locked Loop (DLL)

The Delay-Locked Loop (DLL) unit provides an on-chip digital deskew circuit that generates zero-propagation-delay clock output signals. The deskew circuit compensates for the delay on the routing network by monitoring an output clock, either the CLK0 or the CLK2X. The DLL unit effectively eliminates the delay from the external clock input port to the individual clock loads within the device. The well-buffered global network minimizes the clock skew on the network caused by loading differences.

The input signals to the DLL unit are CLKIN and CLKFB. The output signals from the DLL are CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, and CLKDV.

The DLL unit generates the outputs for the Clock Doubler (CLK2X, CLK2X180), the Clock Divider (CLKDV) and the Quadrant Phase Shifted Outputs functions.

## Digital Frequency Synthesizer (DFS)

The Digital Frequency Synthesizer (DFS) provides a wide and flexible range of output frequencies based on the ratio of two user-defined integers, a Multiplier (CLKFX_MULTIPLY) and a Divisor (CLKFX_DIVIDE). The output frequency is derived from the input clock (CLKIN) by simultaneous frequency division and multiplication. This feature can be used with or without the DLL feature of the DCM. If the DLL is not used, then there is no phase relationship between CLKIN and the DFS outputs.

The DFS unit generates the Frequency Synthesizer (CLKFX, CLKFX180) outputs.

## Phase Shift (PS)

The Phase Shift (PS) unit controls the phase relations of the DCM's clock outputs to the CLKIN input.

The Phase Shift unit shifts the phase of all nine DCM clock output signals by a fixed fraction of the input clock period. The fixed phase shift value is set at design time and loaded into the DCM during FPGA configuration.

The Phase Shift unit also provides a digital interface for the FPGA application to dynamically advance or retard the current shift value by 1/256$^{th}$ of the clock period.

The input signals to the Phase Shift unit are PSINCDEN, PSEN, and PSCLK. The output signals are PSDONE and the STATUS[0] signal.

## Status Logic

The Status Logic indicates the current state of the DCM via the LOCKED and STATUS[0], STATUS[1], and STATUS[2] output signals. The LOCKED output signal indicates whether the DCM outputs are in phase with the CLKIN input. The STATUS output signals indicate the state of the DLL and PS operations.

The RST input signal resets the DCM logic and returns it to its post-configuration state. Likewise, a reset forces the DCM to reacquire and lock to the CLKIN input.

## DCM Primitive

The DCM primitive represents all the Digital Clock Manger functionality. The DCM primitive appears in Figure 4, and the DCM's Connection Ports and Attributes, Properties, or Constraints are summarized below.

### Symbol

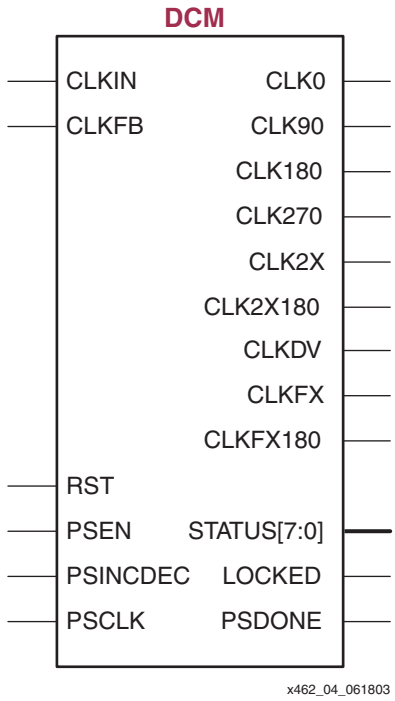



*Figure 4:* **DCM Primitive**

### Connection Ports

Table 3 lists the various connection ports to the Digital Clock Manager. Each port connection has a brief description, which includes the signal direction, and which DCM function units require the connection. Table 2 provides the abbreviated name for each function unit used in Table 3.

*Table 2:* **Functional Unit Abbreviations for Table 3**

| Abbreviation | Functional Unit |
|---|---|
| DLL | Delay-Locked Loop |
| PS | Phase Shifter |
| DFS | Digital Frequency Synthesizer |

*Table 3:* **DCM Connection Ports**

| Port | Direction | Description | Functional Unit | | |
|---|---|---|---|---|---|
| | | | DLL | PS | DFS |
| CLKIN | Clock Input | Clock input to DCM. Always required. The CLKIN frequency and jitter must fall within the limits specified in the Spartan-3 Data Sheet. The frequency limits are controlled by the DLL_FREQUENCY_MODE and DFS_FREQUENCY_MODE attributes. | ✓ | ✓ | ✓ |
| CLKFB | Input | Clock feedback input to DCM. The feedback input is required unless the Digital Frequency Synthesis outputs, CLKFX or CLKFX180, are used stand-alone. The source of the CLKFB input must be the CLK0 or CLK2X output from the DCM and the CLK_FEEDBACK must be set to 1X or 2X accordingly. The feedback point ideally includes the delay added by the clock distribution network, either internally or externally. See "Feedback from a Reliable Source." | ✓ | Optional | ✓ |
| RST | Input | Asynchronous reset input. Resets the DCM logic to its post-configuration state. Causes DCM reacquire and relock to the CLKIN input. Invertible within DCM block. Non-inverted behavior shown below. See "RST Input Behavior."<br>**0**: No effect.<br>**1**: Reset DCM block. Hold RST pulse High for at least 2 ns. | ✓ | ✓ | ✓ |
| PSEN | Input | Dynamic phase shift enable. Invertible within DCM block. Non-inverted behavior shown below. See "Dynamic Fine Phase Shifting."<br>**0**: Disable dynamic phase shifter. Ignore inputs to phase shifter.<br>**1**: Enable dynamic phase shifter operations on next rising PSCLK clock edge. | | ✓ | |
| PSINCDEC | Input | Increment/decrement dynamic phase shift. Invertible within DCM block. Non-inverted behavior shown below. See "Dynamic Fine Phase Shifting."<br>**0**: Increment phase shift value on next enabled, rising PSCLK clock edge.<br>**1**: Decrement phase shift value on next enabled, rising PSCLK clock edge. | | ✓ | |
| PSCLK | Clock Input | Clock input to dynamic phase shifter, clocked on rising edge. Invertible within DCM block. The frequency limits are controlled by the DLL_FREQUENCY_MODE attribute. See "Dynamic Fine Phase Shifting." | | ✓ | |
| CLK0 | Clock Output | Same frequency as CLKIN, 0° phase shift (i.e., not phase shifted). Conditioned to 50% duty cycle when DUTY_CYCLE_CORRECTION attribute is TRUE. Either CLK0 or CLK2X is required as a feedback source for DLL functions. See "Half-Period Phase Shifted Outputs," and "Quadrant Phase Shifted Outputs." | ✓ | | |

*Table 3:* **DCM Connection Ports *(Continued)***

| Port | Direction | Description | Functional Unit | | |
|---|---|---|---|---|---|
| | | | DLL | PS | DFS |
| CLK90 | Clock Output | Same frequency as CLKIN, 90° phase shifted (quarter period). Not available if DLL_FREQUENCY_MODE attribute set to HIGH. Conditioned to 50% duty cycle when DUTY_CYCLE_CORRECTION attribute is TRUE. See "Quadrant Phase Shifted Outputs." | ✓ | | |
| CLK180 | Clock Output | Same frequency as CLKIN, 180° phase shifted (half period). Conditioned to 50% duty cycle when DUTY_CYCLE_CORRECTION attribute is TRUE. See "Half-Period Phase Shifted Outputs," and "Quadrant Phase Shifted Outputs." | ✓ | | |
| CLK270 | Clock Output | Same frequency as CLKIN, 270° phase shifted (three-quarters period). Not available if DLL_FREQUENCY_MODE attribute set to HIGH. Conditioned to 50% duty cycle when DUTY_CYCLE_CORRECTION attribute is TRUE. See "Quadrant Phase Shifted Outputs." | ✓ | | |
| CLK2X | Clock Output | Double-frequency clock output, 0° phase shift. Not available if DLL_FREQUENCY_MODE attribute set to HIGH. When available, the CLK2X output is always 50% duty cycle. Either CLK0 or CLK2X is required as a feedback source for DLL functions. Clock Doubler (CLK2X, CLK2X180) output. See "Half-Period Phase Shifted Outputs." | ✓ | | |
| CLK2X180 | Clock Output | Double-frequency clock output, 180° phase shifted. Not available if DLL_FREQUENCY_MODE attribute set to HIGH. When available, the CLK2X180 output is always 50% duty cycle. Clock Doubler (CLK2X, CLK2X180) output. See "Half-Period Phase Shifted Outputs." | ✓ | | |
| CLKDV | Clock Output | Divided clock output, controlled by the CLKDV_DIVIDE attribute. The CLKDV output has a 50% duty cycle unless the DLL_FREQUENCY_MODE attribute is HIGH and the CLKDV_DIVIDE attribute is a non-integer value. Locking time is longer when CLKDV_DIVIDE is non-integer value. Clock Divider (CLKDV) output. $$F_{CLKDV} = \frac{F_{CLKIN}}{CLKDV\_DIVIDE}$$ | ✓ | | |
| CLKFX | Clock Output | Synthesized clock output, controlled by the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes. Always has 50% duty cycle. If the CLKFX or CLKFX180 clock outputs are used stand-alone, then no clock feedback is required. See "Frequency Synthesizer (CLKFX, CLKFX180)," and "Half-Period Phase Shifted Outputs." $$F_{CLKFX} = F_{CLKIN} \bullet \frac{CLKFX\_MULTIPLY}{CLKFX\_DIVIDE}$$ | | | ✓ |
| CLKFX180 | Clock Output | Synthesized clock output CLKFX, phase shifted by 180° (appears to be inverted version of CLKFX). Always has 50% duty cycle. If only CLKFX or CLKFX180 clock outputs are used on the DCM, then no feedback loop is required. See "Frequency Synthesizer (CLKFX, CLKFX180)," and "Half-Period Phase Shifted Outputs." | | | ✓ |

*Table 3:* **DCM Connection Ports *(Continued)***

| Port | Direction | Description | Functional Unit | | |
|---|---|---|---|---|---|
| | | | DLL | PS | DFS |
| STATUS[0] | Output | Dynamic Phase Shift Overflow. Control output for Dynamic Fine Phase Shifting. The dynamic phase shifter has reached its minimum or maximum limit value. The limit value is either ±255 or a lesser value if the phase shifter reached the end of the delay line. See "Dynamic Fine Phase Shifting."<br>**0**: The Phase Shifter has not yet reached its limit value.<br>**1**: Phase Shifter has reached its limit value. | | ✓ | |
| STATUS[1] | Output | CLKIN Input Stopped Indicator. Available only when CLKFB feedback input is connected. Held in reset until LOCKED output is asserted. Requires at least one CLKIN cycle to become active. Never asserted if CLKIN never toggles.<br>**0**: CLKIN input is toggling.<br>**1**: CLKIN input is not toggling. | ✓ | ✓ | ✓ |
| STATUS[2] | Output | CLKFX or CLKFX180 Output Stopped Indicator. See Frequency Synthesizer (CLKFX, CLKFX180).<br>**0**: CLKFX and CLKFX180 outputs are toggling.<br>**1**: CLKFX and CLKFX180 outputs are not toggling, even though LOCKED output may still be High. | | | ✓ |
| STATUS[7:3] | Output | Reserved | | | |
| LOCKED | Output | All DCM features have locked onto CLKIN frequency. Clock outputs are now valid, assuming CLKIN is within specified limits (as described in "DCM Clock Requirements"). See "Frequency Synthesizer (CLKFX, CLKFX180)."<br>**0**: DCM is attempting to lock onto CLKIN frequency. DCM clock outputs are not valid.<br>**1**: DCM is locked onto CLKIN frequency. DCM clock outputs are valid.<br>**1-to-0**: DCM lost lock. Reset DCM. | ✓ | ✓ | ✓ |
| PSDONE | Output | Dynamic phase shift operation complete. See "Dynamic Fine Phase Shifting."<br>**0**: No phase shift operation active or phase shift operation in progress.<br>**1**: Requested phase shift operation is complete. Output High for one PSCLK cycle. Okay to provide next dynamic phase shift operation. | | ✓ | |

## Attributes, Properties, or Constraints

Table 4 lists the various attributes for the Digital Clock Manager. All attributes are set at design time and programmed during configuration. Most, except for the Dynamic Fine Phase Shift function, cannot be changed by the FPGA application at run-time. To set an attribute, set <ATTRIBUTE>=<SETTING> as appropriate for the design entry tool.

*Table 4:* **DCM Attributes**

| Attribute | Allowable Settings and Description |
|---|---|
| DLL_FREQUENCY_MODE | Specifies the allowable frequency range for the CLKIN input, the PSCLK input, and for the output clocks from the DCM's Delay-Locked Loop (DLL) unit. The DLL clock outputs include CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKDV.<br><br>**LOW**: Default. The DLL function unit operates in its low-frequency mode. All DLL-related outputs are available. The frequency for all clock inputs and outputs must fall within the low-frequency DLL limits specified in the Spartan-3 Data Sheet.<br><br>**HIGH**: The DLL function unit operates in its high-frequency mode. The Clock Doubler (CLK2X, CLK2X180) outputs are not available. The Quadrant Phase Shifted Outputs CLK90 and CLK270 are not available. The duty cycle for the CLKDV output is not 50% if the CLKDV_DIVIDE attribute is a non-integer. The frequency for all clock inputs and outputs must fall within the high-frequency DLL limits specified in the Spartan-3 Data Sheet. |
| CLK_FEEDBACK | Defines the frequency of the feedback clock.<br><br>**1X**: Default. CLK0 feedback. Same frequency as CLKIN.<br><br>**2X**: CLK2X feedback. Double the frequency of CLKIN.<br><br>**None**: No feedback. Allowed if using only the CLKFX or CLKFX180 outputs. |
| DUTY_CYCLE_CORRECTION | Enables or disables the 50% duty-cycle correction for the CLK0, CLK90, CLK180, and CLK270 outputs from the DLL unit.<br><br>**TRUE**: Default. Enable duty-cycle correction.<br><br>**FALSE**: Disable duty-cycle correction. |
| CLKDV_DIVIDE | Defines the frequency of the CLKDV output. Allowable values for CLKDV_DIVIDE include 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, 16.<br>$$F_{CLKDV} = \frac{F_{CLKIN}}{CLKDV\_DIVIDE}$$<br>Locking time is longer and there is more output jitter when CLKDV_DIVIDE is a non-integer value. |
| CLKFX_MULTIPLY | Defines the multiplication factor for the frequency of the CLKFX and CLKFX180 outputs. Used in conjunction with CLKFX_DIVIDE attribute. Allowable values for CLKFX_MULTIPLY include integers ranging from 2 to 32. Default value is 4.<br>$$F_{CLKFX} = F_{CLKIN} \bullet \frac{CLKFX\_MULTIPLY}{CLKFX\_DIVIDE}$$ |

*Table 4:* **DCM Attributes *(Continued)***

| Attribute | Allowable Settings and Description |
|---|---|
| CLKFX_DIVIDE | Defines the division factor for the frequency of the CLKFX and CLKFX180 outputs. Used in conjunction with CLKFX_MULTIPLY attribute. Allowable values for CLKFX_DIVIDE include integers ranging from 1 to 32. Default value is 1.<br>$$F_{CLKFX} = F_{CLKIN} \bullet \frac{CLKFX\_MULTIPLY}{CLKFX\_DIVIDE}$$ |
| PHASE_SHIFT | The PHASE_SHIFT attribute is applicable only if the CLKOUT_PHASE_SHIFT attribute is set to FIXED or VARIABLE. Defines the rising-edge skew between CLKIN and all the DCM clock outputs at configuration and consequently phase shifts the DCM clock outputs.<br>The skew or phase shift value is specified as an integer that represents a fraction of the clock period as expressed in the following equations. The integer value must range from –255 to 255. The default is 0. Actual allowable values depend on input clock frequency. The actual range is less when $T_{CLKIN}$ > FINE_SHIFT_RANGE. The FINE_SHIFT_RANGE specification represents the total delay of all taps in the delay line. See "Fine Phase Shifting," for more information. |
| CLKOUT_PHASE_SHIFT | Sets the phase shift mode. Together with the PHASE_SHIFT constraint, implements the Digital Phase Shifter (DPS) feature of the DCM. Affects all DCM clock outputs from both the DLL and DFS units. See "Fine Phase Shifting," for more information.<br>**NONE**: Default. CLKIN and CLKFB are in phase (no skew) and phase relationship cannot be changed. Equivalent to FIXED setting with a PHASE_SHIFT value of 0.<br>**FIXED**: Phase relationship is set at configuration by the PHASE_SHIFT attribute value and cannot be changed by the application.<br>**VARIABLE**: Phase relationship is set at configuration by the PHASE_SHIFT attribute value but can be changed by the application using the dynamic phase shift controls, PSEN, PSCLK, PSINCDEC, and PSDONE. |
| DESKEW_ADJUST | Controls the clock delay alignment between the FPGA clock input pin and the DCM output clocks. See "Skew Adjustment."<br>**SYSTEM_SYNCHRONOUS**: Default. All devices clocked by a common, system-wide clock source.<br>**SOURCE_SYNCHRONOUS**: Clock is provided by the data source, i.e., source-synchronous applications.<br>Do not use this setting to phase shift DCM clock outputs. Instead, use the CLKOUT_PHASE_SHIFT and PHASE_SHIFT constraints to achieve accurate phase shifting. |

*Table 4:* **DCM Attributes *(Continued)***

| Attribute | Allowable Settings and Description |
|---|---|
| DFS_FREQUENCY_MODE | Specifies the allowable frequency range for the CLKFX and CLKFX180 output clocks from the DCM's Digital Frequency Synthesizer (DFS). If any DLL clock outputs are used, then the more restrictive DLL_FREQUENCY_MODE limits the CLKIN input frequency.<br><br>**LOW**: Default. The DFS function unit operates in its low-frequency mode. The frequency for the CLKFX and CLKFX180 outputs must fall within the low-frequency DFS limits specified in the Spartan-3 Data Sheet. The frequency limits for the CLKIN input depend on if any DLL clock outputs are used.<br><br>**HIGH**: The DFS function unit operates in its high-frequency mode. The frequency for the CLKFX and CLKFX180 outputs must fall within the high-frequency DFS limits specified in the Spartan-3 Data Sheet. The frequency limits for the CLKIN input depend on if any DLL clock outputs are used. |
| STARTUP_WAIT | Controls whether the FPGA configuration signal DONE waits for the DCM to assert its LOCKED signal before going High.<br><br>**FALSE**: Default. DONE asserted at end of configuration without waiting for DCM to assert LOCKED.<br><br>**TRUE**: DONE signal does not go High until the LOCKED signal goes HIGH on the associated DCM. STARTUP_WAIT does not prevent LOCKED from going High. The FPGA startup sequence must also be modified to insert a LCK (lock) cycle before the postponed cycle (see "Bitstream Generation Settings"). Either the DONE cycle or GWE cycle are typical choices.<br><br>If more than one DCM is so configured, the FPGA waits until all DCMs are locked. |
| CLKIN_DIVIDE_BY_2 | Optionally divides the CLKIN in half before entering DCM block. In some applications, reduces the input clock frequency to within acceptable limits.<br><br>**FALSE**: Default. CLKIN input directly feeds the DCM block.<br><br>**TRUE**: Divides CLKIN frequency in half and provides roughly a 50% duty cycle clock before entering the DCM block. Helpful with high-frequency clocks to meet the DCM input clock frequency or duty-cycle requirements. Divides clock frequency in half when determining operating frequency modes and calculating phase shift limits. |

*Table 4:* **DCM Attributes *(Continued)***

<table>
<tr><th>Attribute</th><th>Allowable Settings and Description</th></tr>
<tr><td>FACTORY_JF</td><td>Controls how often the DCM's DLL unit adjusts its tap settings. The FACTORY_JF setting affects the jitter characteristics of the DLL element.<br>The settings are automatically adjusted based on the DLL_FREQUENCY_MODE attribute.<br><table><tr><th>DLL_FREQUENCY_MODE</th><th>FACTORY_JF</th></tr><tr><td>LOW</td><td>0xC080</td></tr><tr><td>HIGH</td><td>0xF0F0</td></tr></table>Do not change the default values unless otherwise recommended (see "Adjusting FACTORY_JF Setting").</td></tr>
<tr><td>LOC</td><td>Specifies the physical location of the DCM, as shown in Figure 1.<br><table><tr><td>DCM_X0Y0</td><td>Lower left DCM.</td></tr><tr><td>DCM_X1Y0</td><td>Lower right DCM. Not available in XC3S50.</td></tr><tr><td>DCM_X0Y1</td><td>Upper left DCM.</td></tr><tr><td>DCM_X1Y1</td><td>Upper right DCM. Not available in XC3S50.</td></tr></table></td></tr>
</table>

## Compatibility with Other Xilinx FPGA Families

The Spartan-3 Digital Clock Manager (DCM) is nearly functionally identical to the DCM units found in Virtex™-II and Virtex-II Pro FPGA families. However, the Spartan-3 DCM is the third-generation in DCM design with some improved capabilities over previous FPGA families. Specifically, Spartan-3 has improved immunity to noise on the $V_{CCAUX}$ supply compared to Virtex-II and has more flexible phase shifting than both Virtex-II and Virtex-II Pro families. The DCMs on both Virtex-II and Virtex-II Pro families have a higher output frequency limit.

The Spartan-3 DCM is a significant enhancement over the Spartan-II/IIE Delay-Locked Loop (DLL) function. A Spartan-3 DCM provides all the capabilities of the Spartan-II/IIE DLL with new capabilities such as the Frequency Synthesizer and phase shifting functions. The Spartan-3 Frequency Synthesizer multiplies an input clock by up to a factor of 32. The Spartan-II/IIE DLL has limited frequency multiplication capabilities—namely, an input clock can be doubled. Similarly, the Spartan-3 DCM has a wider divider range compared to Spartan-IIE DLLs.

## DCM Clock Requirements

The DCM is built for maximum flexibility, but there are certain requirements on clock frequency and clock stability, both frequency variation and clock jitter.

### Input Clock Frequency Range

The DCM clock input frequency depends on whether the DLL functional unit, the DFS unit, or both are utilized in the application.

Table 5 shows the clock input, CLKIN, frequency range for the Digital Frequency Synthesizer (DFS) unit. The DFS unit, if used stand-alone, has a wider frequency range than the DLL unit. If the application uses both units, then the more restrictive DLL requirements apply. The table shows the data sheet specification name and an estimated value. The actual value depends on which speed grade is required for the design and the value specified in the data sheet takes precedence over the estimate.

*Table 5:* **Digital Frequency Synthesizer (DFS) Unit Clock Input Frequency Requirements**

| Function | Minimum Frequency | Maximum Frequency |
|---|---|---|
| Digital Frequency Synthesizer (DFS) | CLKIN_FREQ_FX_MIN ~ 1.00 MHz* | CLKIN_FREQ_FX_MAX ~326 MHz* |

* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

Table 6 shows the clock input, CLKIN, frequency range for the Delay-Locked Loop (DLL) unit. The DLL frequency restrictions apply regardless if the DLL is used stand-alone or with the DFS unit. The table shows the frequency range when the DLL unit operates in either is low- or high-frequency mode. The mode is controlled by the DLL_FREQUENCY_MODE attribute. Likewise, the table shows the data sheet specification name and an estimated value. The actual value depends on which speed grade is required for the design and the value specified in the data sheet takes precedence over the estimate.

*Table 6:* **Delay-Locked Loop (DLL) Unit Clock Input Frequency Requirements**

| Function | DLL Frequency Mode Attribute (DLL_FREQUENCY_MODE) | | | |
|---|---|---|---|---|
| | = LOW | | = HIGH | |
| | Minimum Frequency | Maximum Frequency | Minimum Frequency | Maximum Frequency |
| Delay Locked Loop (DLL) | CLKIN_FREQ_DLL_LF_MIN ~ 24 MHz* | CLKIN_FREQ_DLL_LF_MAX ~ 180 MHz* | CLKIN_FREQ_DLL_HF_MIN ~ 48 MHz* | CLKIN_FREQ_DLL_HF_MIN ~326 MHz* |

* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

## Output Clock Frequency Range

The various DCM output clocks also have a specified frequency range. See the "Input and Output Clock Frequency Restrictions" section for more information.

## Input Clock and Clock Feedback Variation

As described later in the "A Stable, Monotonic Clock Input" section, the DCM expects a stable, monotonic clock input. However, for maximum flexibility, the DCM tolerates a certain amount of clock jitter on the CLKIN input and a reasonable amount of frequency variation on both the CLKIN input and the CLKFB clock feedback input.

There are two types of jitter tolerance on the CLKIN input. Cycle-to-cycle jitter indicates how much the CLKIN input period is allowed to change from one cycle to the next. The maximum allowable cycle-to-cycle change is shown in Table 7, including the data sheet specification name and an estimated value.

*Table 7:* **Maximum Allowable Cycle-to-Cycle Jitter**

| Functional Unit | Frequency Mode | |
|---|---|---|
| | Low | High |
| Digital Frequency Synthesizer (DFS) | CLKIN_CYC_JITT_FX_LF ~ ±300 ps* | CLKIN_CYC_JITT_FX_HF ~ ±150 ps* |
| Delay Locked Loop (DLL) | CLKIN_CYC_JITT_DLL_LF ~ ±300 ps* | CLKIN_CYC_JITT_DLL_HF ~ ±150 ps* |

* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

The other applicable type of jitter is called period jitter. Period jitter indicates the maximum range of the period variation over millions of clock cycles. Cycle-to-cycle jitter shows the

change from one clock to the next while period jitter shows the total range of changes over time. The maximum allowable period jitter appears in Table 8, including the data sheet specification name, and an estimated value.

*Table 8:* **Maximum Allowable Period Jitter**

| Functional Unit | Frequency Mode | |
|---|---|---|
| | Low | High |
| Digital Frequency Synthesizer (DFS) | CLKIN_PER_JITT_FX_LF<br>~ ±1,000 ps* (±1 ns) | CLKIN_PER_JITT_FX_HF<br>~ ±1,000 ps* (±1 ns) |
| Delay Locked Loop (DLL) | CLKIN_PER_JITT_DLL_LF<br>~ ±1,000 ps* (±1 ns) | CLKIN_PER_JITT_DLL_HF<br>~ ±1,000 ps* (±1 ns) |

* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

Another source of stability for the DCM is the clock feedback path used by the DLL unit. The feedback path delay variance must also be within the limit shown in Table 9. This limit only applies to an external feedback path as any on-chip variance is minimal when connected to a global clock line.

*Table 9:* **External Feedback Path Delay Variation**

| Description | Specification |
|---|---|
| Maximum allowable variation in off-chip CLKFB feedback path | CLKFB_DELAY_VAR_EXT<br>~ ±1,000 ps* (±1 ns) |

* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

## LOCKED Output Behavior

The DCM's LOCKED output indicates when all the enabled DCM functions have locked to the CLKIN input. When the DCM asserts LOCKED, the output clocks are valid for use within the FPGA application.

Figure 5 shows the behavior of the LOCKED output. The LOCKED output is Low immediately after the FPGA finishes its configuration process and is Low whenever the RST input is asserted.

After configuration, the DCM always attempts to lock, whether the CLKIN signal is valid yet or not. If the input clock is not yet stable, the FPGA circuit should assert the RST input until the CLKIN input stabilizes. The DLL unit uses both the CLKIN input and the CLKFB feedback input to determine when locking is complete, that is, when the rising edges of CLKIN and CLKFB are in phase. The DFS unit monitors CLKIN to determine if a valid frequency is present on CLKIN. To achieve lock, the DCM may need to sample several thousand clock cycles.

The DCM asserts it LOCKED output High upon locking onto CLKIN. The DCM clock outputs are then valid and available for use within the FPGA application. The DCM timing section of the Spartan-3 Data Sheet provides worst-case locking times. In general, the DLL unit outputs lock faster with increasing clock frequency. The DFS unit outputs require significantly longer to lock, depending on the multiply and divide factors. Smaller multiply and divide factors result in faster lock times.

To guarantee that the system clock is established before the FPGA completes its configuration process, the DCM can optionally delay the completion of the configuration process until after the DCM locks. The STARTUP_WAIT attribute activates this feature.

Until LOCKED is High, there is no guarantee how the DCM clock outputs behave. The DCM output clocks are not valid until LOCKED is High and before that time can exhibit glitches, spikes, or other spurious behavior.

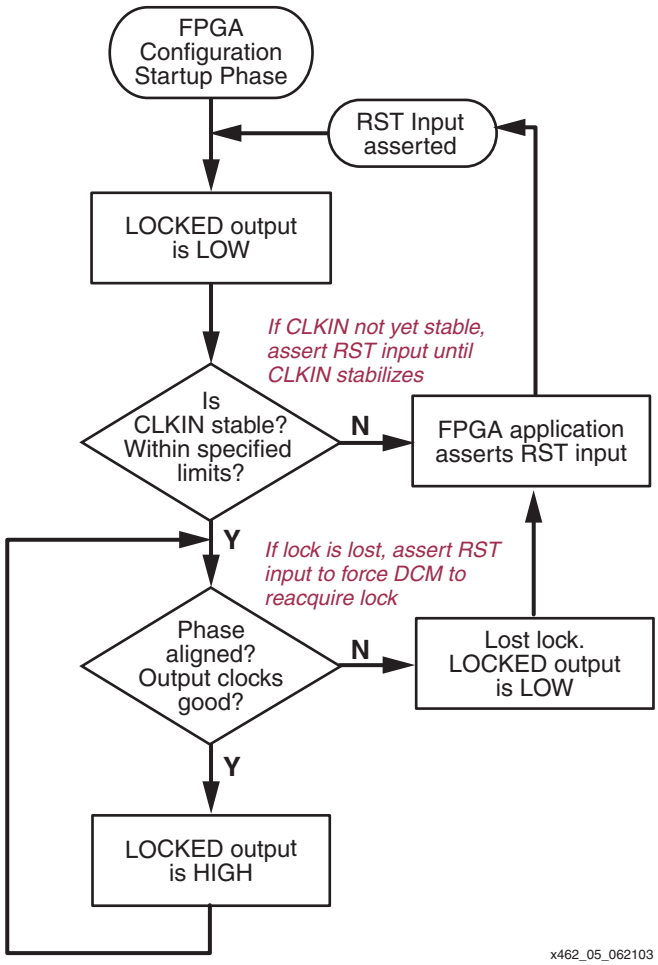



*Figure 5:* **Functional Behavior of LOCKED Output**

While the CLKIN input stays within the specified limits, the DCM continues to adjust its internal delay taps to maintain lock. However, if the CLKIN input strays well beyond the specified limits, then the DCM loses lock and deasserts the LOCKED output.

Once the DCM loses lock, it does not automatically attempt to reacquire lock. When the DCM loses lock—i.e., LOCKED was High, then goes Low—the FPGA application must take the appropriate action. For example, once lock is lost, resetting the DCM via the RST input forces the DCM to reacquire lock.

## RST Input Behavior

The asynchronous RST input forces the DCM to its post-configuration state. Use the RST pin when reconfiguring the FPGA or when changing the input clock frequency beyond the allowable range. The active-High RST pin either must connect to a dynamic signal or must be tied to ground. The RST input must be asserted for 2 ns or longer.

If the input clock frequency is not yet stable after configuration, assert RST until the clock stabilizes. When using external feedback, hold the DCM in reset immediately after configuration. Figure 20, page 138 shows an example reset technique using an SRL16 shift register primitive.

If the DCM loses lock—i.e., the LOCKED output was High then goes Low—then the FPGA application must assert RST to force the DCM to reacquire the input clock frequency.

If the DCM LOCKED output is High, then the LOCKED signal deactivates within four source clock cycles after RST is asserted. Asserting RST forces the DCM to reacquire lock.

Asserting RST also resets the DCM's delay tap position to zero. Due to the tap position changes, glitches may occur on the DCM clock output pins. Similarly, RST affects the duty cycle on the clock outputs.

Asserting RST also resets the present variable phase shift value back to the value specified by the PHASE_SHIFT attribute.

## DCM Wizard

To simplify applications using DCMs, the Xilinx ISE development software includes a software wizard that provides step-by-step instructions for configuring a DCM. As shown in Figure 6, DCM Wizard generates a vendor-specific logic synthesis file instantiating the DCM in either VHDL or Verilog syntax. Similarly, DCM Wizard generates a user constraints (UCF) file for the specific implementation. Finally, all the user specifications are saved in a Xilinx Architecture Wizard (XAW) settings file.

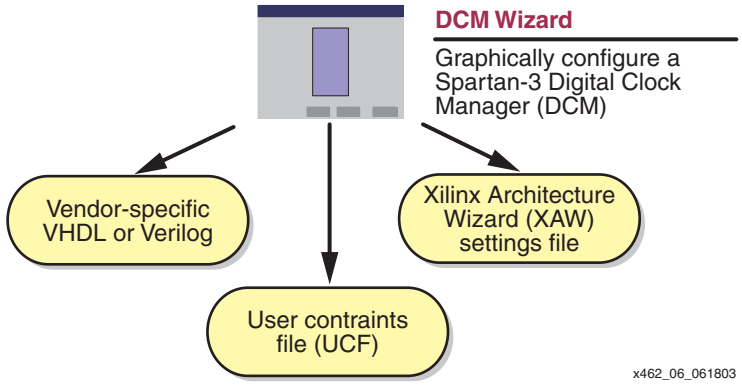



*Figure 6:* **DCM Wizard Provides a Graphical Interface for Configuring Digital Clock Managers**

### Invoking DCM Wizard

There are multiple methods to invoke DCM Wizard, either from the Windows Start button or from within the Xilinx ISE Project Navigator software.

#### From Windows Start Button

To invoke DCM Wizard from the Windows Start button, click **Start → Programs → Xilinx ISE 5 → Accessories → Architecture Wizard**. The setup window shown in Figure 7 appears.

- Specify the name of the Xilinx Architecture Wizard (.xaw) file that holds the option settings for this DCM.
- Optionally, click **Browse** and select a directory location for the `*.xaw` file.
- Select the logic synthesis language for the output file, either VHDL or Verilog.
- Choose the targeted logic synthesis tool. DCM Wizards creates vendor-specific output for the specified synthesis tool.
- Select the targeted Spartan-3 device.

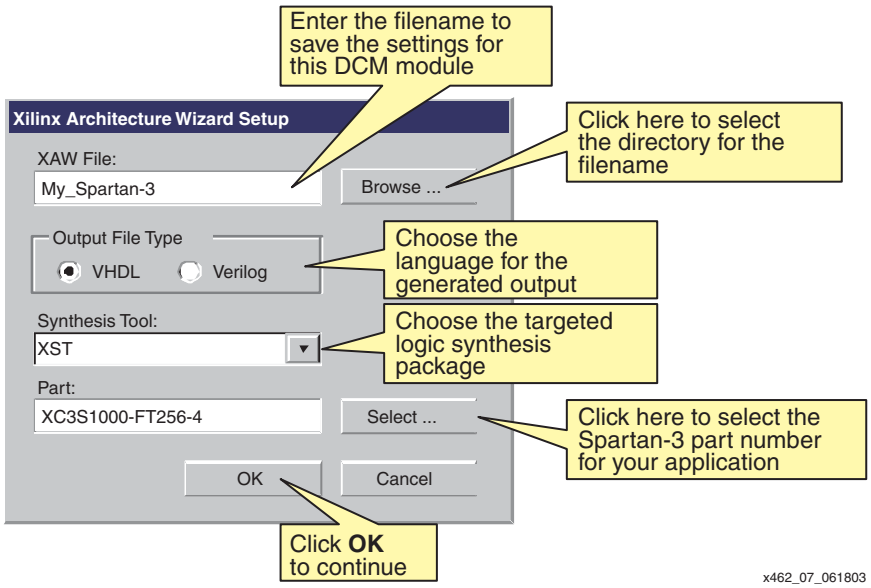



*Figure 7:* **Set Up the Architecture Wizard**

### From within Project Navigator

Optionally, invoke DCM Wizard from within Project Navigator, either from the menu bar or from within the "Sources in Project" window. From the menu bar, select **Project → New Source**. Alternately, right-click in the "Sources in Project" window and choose **New Source**.

Select **Architecture Wizard** from the available list, as shown in Figure 8. Enter the file name for the Xilinx Architecture Wizard (`*.xaw`) file, and select the directory where the file will be saved. Click **Next >** to continue.

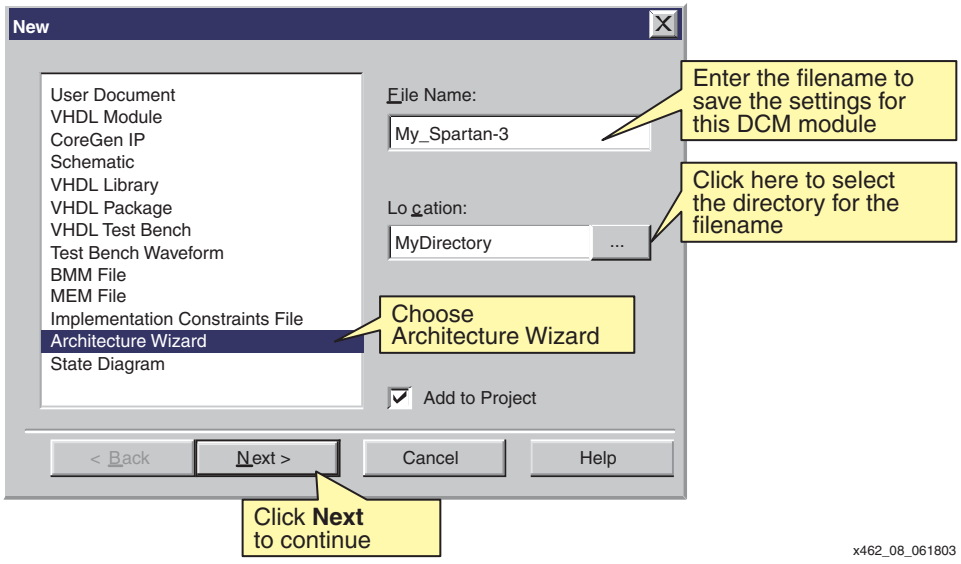



*Figure 8:* **Configuring a New Architecture Wizard in the Project Navigator**

## Wizard Selection

The previous procedures are common to any of the ISE Architecture Wizards. Spartan-3 FPGAs supports the DCM Wizard, as shown in Figure 9. Click **OK** to continue.

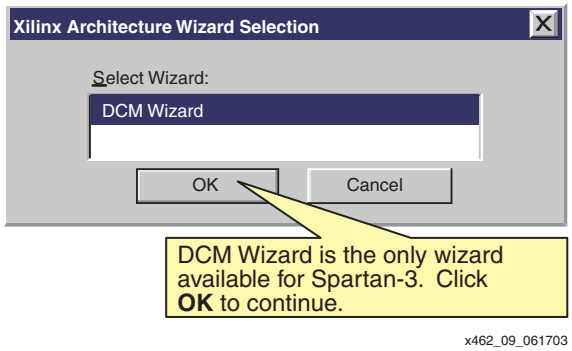



*Figure 9:* **DCM Wizard is the only Wizard Available for Spartan-3 FPGAs**

## General Setup

Specify most of the DCM's options using the DCM Wizard General Setup panel, as shown in Figure 10. The text in ovals shows the DCM primitive attribute name for the corresponding setting.

- Enter the name for this specific DCM instance. This name is used within the Verilog or VHDL output file.
- To select the outputs and functions used in the final application, check the option boxes next to the desired DCM clock outputs. Checking the output boxes enables related option settings below.
- Enter the frequency of the CLKIN clock input. Either specify the frequency in MHz, or specify the clock period in nanoseconds. The specified value also sets DCM's DLL_FREQUENCY_MODE attribute.
- Specify whether the CLKIN source is internal or external to the FPGA. If **External**, then DCM Wizard automatically inserts a global buffer input (IBUFG) primitive. If **Internal**, then the source signal is provided as a top-level input within the generated HDL source file.
- If the CLKDV output box is checked, then specify the **Divide by Value** for the Clock Divider circuit. This setting defines the DCM's CLKDV_DIVIDE attribute.
- Specify the feedback path to the DCM. If only the CLKFX or CLKFX180 outputs are used, then select **None**. Otherwise, feedback is required. If the feedback is from within the FPGA, choose **Internal**. If the feedback loop is from outside the FPGA, choose **External**. Furthermore, specify the source of the DCM feedback, either from CLK0 (**1X**) or from CLK2X (**2X**). This setting defines the DCM's CLK_FEEDBACK attribute.

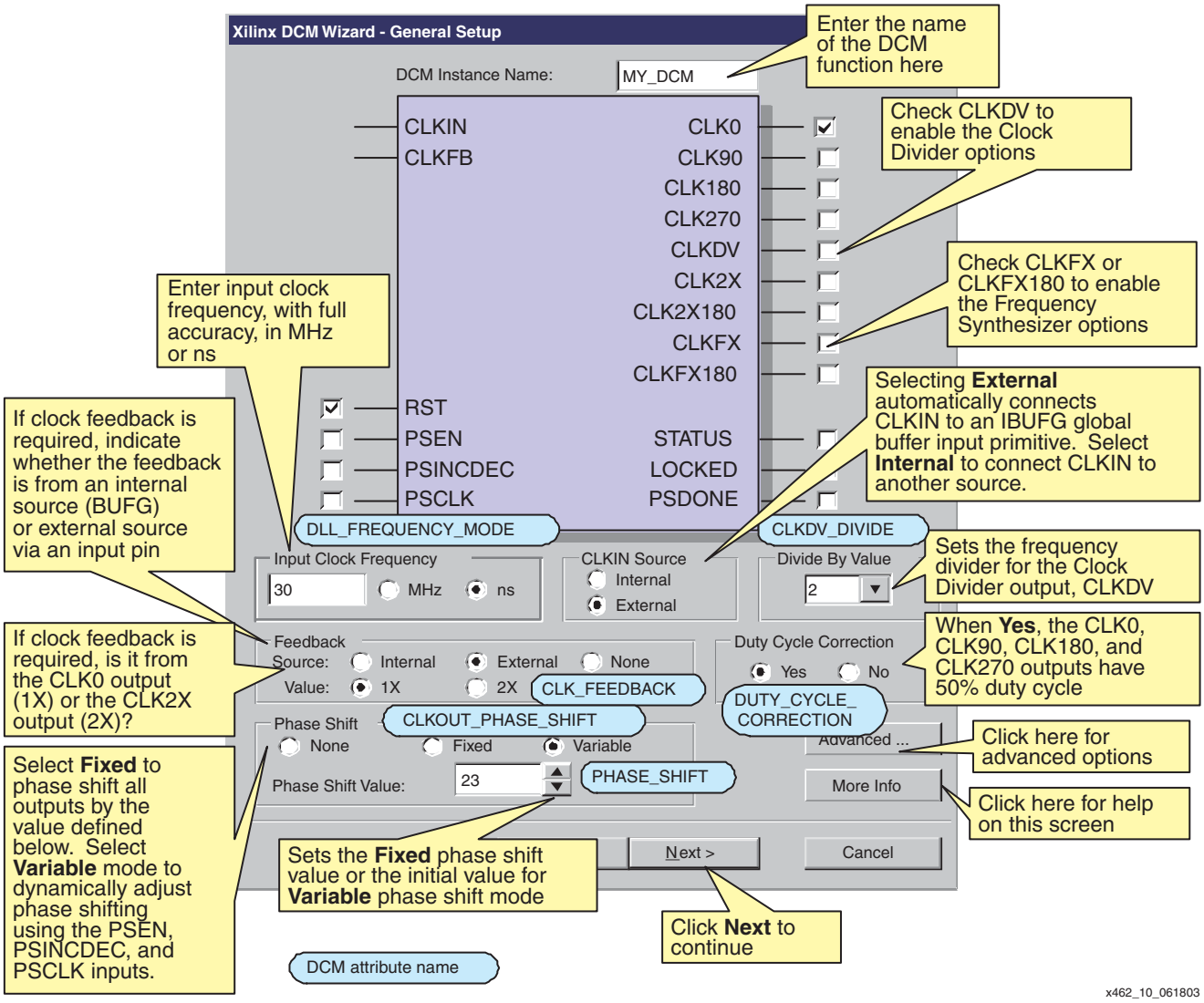



*Figure 10:* **A Majority of DCM Options are Set in the General Setup Panel**

- Specify whether to phase shift all DCM outputs. By default, there is no phase shifting (**None**). If phase shifting is required by the application, choose whether the phase shift value is **Fixed** or **Variable**. Selecting **Variable** also enables the dynamic phase shift controls, PSEN, PSINCDEC, PSCLK, and PSDONE. This setting defines the DCM's CLKOUT_PHASE_SHIFT attribute. For both **Fixed** and **Variable** modes, specify the related **Phase Shift Value**, which provides either the fixed phase shift value or the initial value for the dynamic phase shift. This setting defines the DCM's PHASE_SHIFT attribute.
- To open the Advanced Options window, click **Advanced**.
- When finished, click **Next >** to continue to the Clock Buffers panel.

## Advanced Options

Various advanced DCM options are grouped together in the Advanced Options window, shown in Figure 11:

- By default, the DCM has no effect on the FPGA's configuration process. Click **Yes** to have the FPGA wait for the DCM to assert its LOCKED output before asserting the DONE signal at the end of configuration. This setting defines the DCM's STARTUP_WAIT

attribute. If set to **Yes**, additional bitstream generation option changes are required, as described in the "Setting Configuration Logic to Wait for DCM LOCKED Output" section.

- If the CLKIN input frequency is too high for a particular DCM feature, click **Yes** under **Divide Input Click by 2** to reduce the input frequency by half with roughly a 50% duty cycle before entering the DCM block. This setting defines the DCM's CLKIN_DIVIDE_BY_2 attribute.
- If required for source-synchronous data transfer applications, modify the **DCM Deskew Adjust** value to **SOURCE_SYNCHRONOUS**. Do not use any values other than SOURCE_SYNCHRONOUS or SYSTEM_SYNCHRONOUS without first consulting Xilinx. This setting defines the DCM's DESKEW_ADJUST attribute. See "Skew Adjustment."
- Click **OK** when finished to apply any changes and return to the General Setup window.

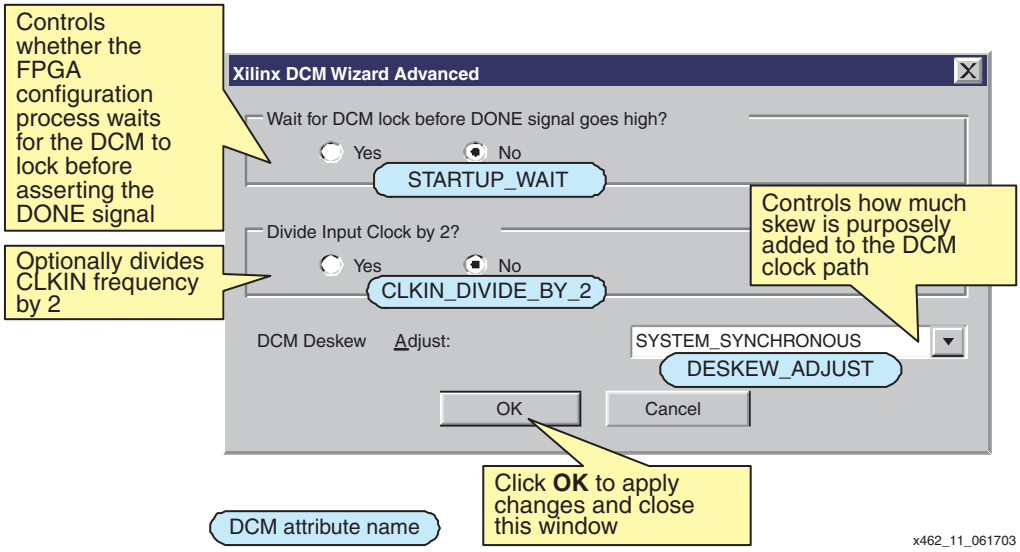



*Figure 11:* **DCM Advanced Options Panel**

## Clock Buffers

Define the clock buffer output type for each DCM clock output, shown in Figure 12. By default, DCM Wizard automatically assigns all outputs to a global buffer (BUFG). However, there are only four global buffers along each the top or bottom edge of the device, shared by two DCMs. In the XC3S50, there is a single DCM along the top or bottom edge that optionally connects to all four global buffers along the edge.

- To assign clock buffer types for each DCM clock output, click **Customize** under **Clock Buffer Settings**.
- For each DCM clock output, select a **Clock Buffer** output type using the drop-down list. Table 10 lists the available Clock Buffer options.
- If using an **Enabled Buffer** output type, either specify a signal name for the buffer enable (CE) input or use the automatically generated name.
- If using a **Clock Mux** output type, either specify a signal name for the select (S) input or use the automatically generated name.
- When finished, click **Next >** or **Finish** to continue. The **Next >** option only appears if the CLKFX or CLKFX180 outputs were selected in the General Setup panel. Otherwise, click **Finish** to generate the HDL output (see "Generating HDL Output").

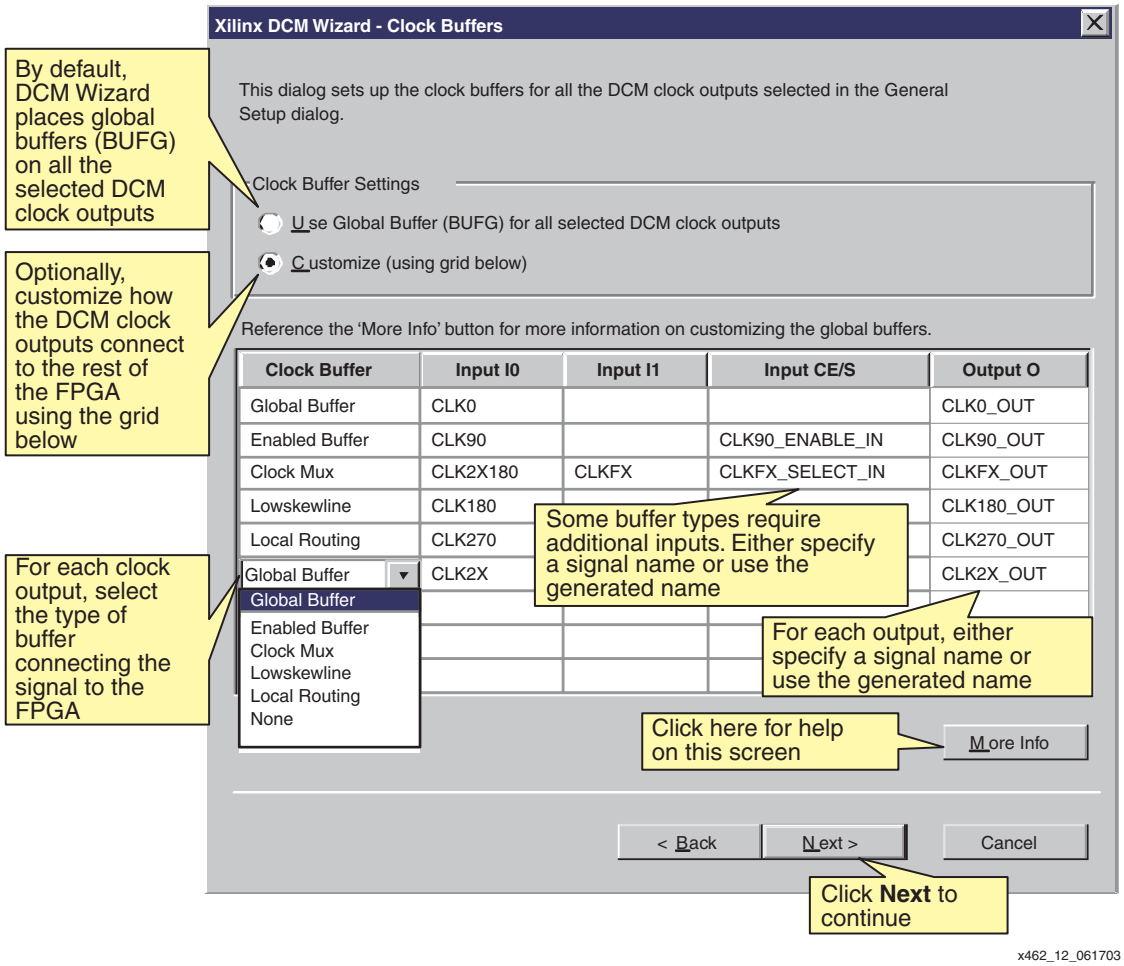



*Figure 12:* **DCM Wizard Provides a Variety of Buffer Options for each DCM Output**

*Table 10:* **Settings for Clock Buffer Output Types**

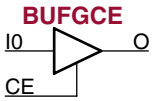
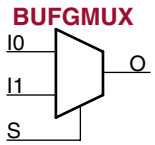
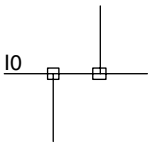
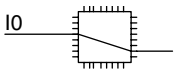
| Clock Buffer Selection | Diagram | Description |
|---|---|---|
| Global Buffer |  | Connect to one of four global buffers (BUFG) along the same edge as the DCM. |
| Enabled Buffer |  | Connect to one of the four global buffers configured as an enable clock buffer (BUFGCE). The CE input enables the buffer when High. When CE is Low, the buffer output is zero.<br>CE \| O<br>0 \| 0<br>1 \| I0 |

*Table 10:* **Settings for Clock Buffer Output Types**

| Clock Buffer Selection | Diagram | Description |
|---|---|---|
| Clock Mux |  | Connect to one of the four global buffers configured as a clock multiplexer (BUFGMUX). The S input selects the clock source.<br>S: 0 → O: I0; S: 1 → O: I1 |
| Lowskewline |  | Connect to low-skew programmable interconnect. |
| Local Routing |  | Connect to local interconnect, skew not critical. |
| None | | Disable DCM output. |

## Clock Frequency Synthesizer

The Clock Frequency Synthesizer panel, shown in Figure 13, only appears if the CLKFX or CLKFX180 outputs were selected in the General Setup panel.

Here, specify either the desired output frequency or enter the specific values for the multiply and divide factors. The frequency limits—or delay limits if CLKIN was specified in ns—appear under **Valid Ranges for Selected Speed Grade**. The range is displayed for both possible values of the DFS_FREQUENCY_MODE attribute. The range is tighter if the DCM uses any of the DLL-related clock outputs.

- Click **Use output frequency** and enter the requested value, in as much precision as possible, either in megahertz (**MHz**) or in nanoseconds (**ns**). Click **Calculate** to compute the values for the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes. If no solution is available using the possible multiply and divide values, DCM Wizard issues an error message asking for another output frequency value. If a solution exists, then the multiply and divide values, plus the resulting jitter values (see "Clock Jitter or Phase Noise") appear under **Generated Output**.
- Optionally, click **Use Multiply (M) and Divide (D) values** and enter the desire values. Click **Calculate** to calculate the resulting output frequency and jitter, displayed under **Generated Output**.
- Finally, click **Finish** to generate the HDL output (see "Generating HDL Output").

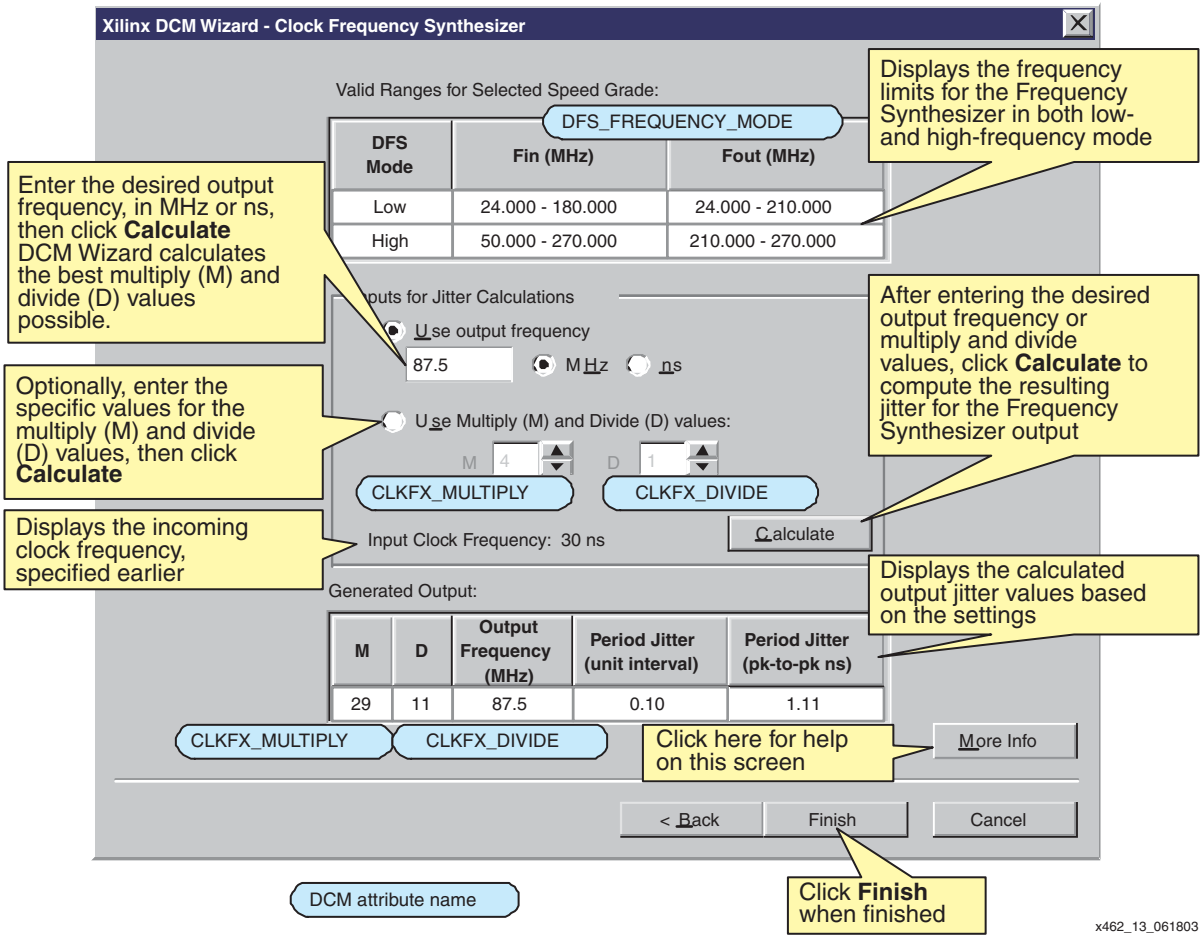



*Figure 13:* **Set the Multiply and Divide Values for the Digital Frequency Synthesizer and Calculate the Resulting Jitter**

## Generating HDL Output

After entering all the parameters and clicking **Finish**, DCM Wizard automatically generates the requested VHDL or Verilog HDL output file, as shown in Figure 14. DCM Wizard also generates a User Constraints File (UCF) based on the settings.

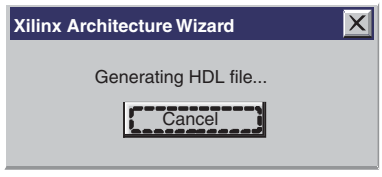



*Figure 14:* **DCM Wizard Generates Either a VHDL or Verilog HDL Output File**

# VHDL and Verilog Instantiation

DCM Wizard is the easiest method to create a VHDL or Verilog HDL description of a DCM. However, Verilog and VHDL source examples are also available.

### Language Templates within Project Navigator

There are DCM language templates available within the ISE 5.2i and later Project Navigator. To select a DCM template, select **Edit → Language Templates** from the Project Navigator menu. From the Templates tree shown in Figure 15, expand either the **Verilog** or **VHDL** folder, then

the **Component Instantiation** folder, then the **DCM** folder. Under the DCM folder, select the desired DCM source file. The selected source file appears in the adjacent window.

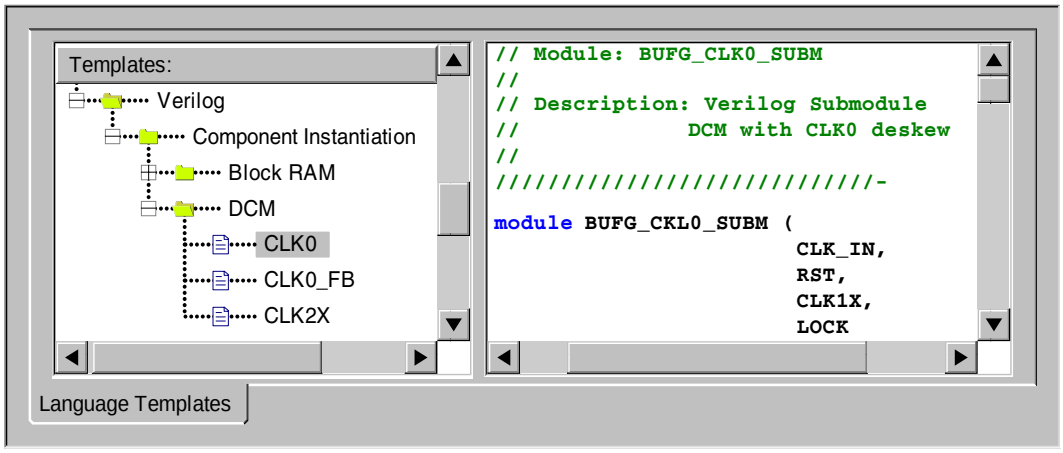



*Figure 15:* **DCM Designs in Project Navigator Language Templates**

Use the file either as a reference or cut the content of the window into a new source file.

### VHDL and Verilog Reference Files

The same VHDL and Verilog source files are also available for download from the Xilinx FTP site from the following locations.

- VHDL DCM Reference Files
  ftp://ftp.xilinx.com/pub/applications/xapp/xapp462_vhdl.zip
- Verilog DCM Reference Files
  ftp://ftp.xilinx.com/pub/applications/xapp/xapp462_verilog.zip

## Eliminating Clock Skew

One of the fundamental functions of a DCM is to eliminate clock skew. Eliminating clock skew is important for most designs that operate at 50 MHz or more. Furthermore, the concepts involved in clock skew elimination also apply to many of the other applications of a DCM.

### What is Clock Skew?

Clock skew inherently exists in every synchronous system. The pristine clock edge generated by the clock source arrives at different times at different points in the system—either within a single device or on the clock inputs to the different devices connected to the clock. This difference in arrival times is defined as clock skew.

Figure 16 illustrates clock skew in an example system. A clock source drives the clock input to an FPGA. The clock enters through an input pin on the FPGA, is distributed within the FPGA using the internal low-skew global clock network, and arrives at a flip-flop within the FPGA. Each element in the clock path delays the arrival of the clock edge at the flip-flop. Consequently, the clock input at the flip-flop—Point (B)—is delayed, or skewed compared to the original clock source at Point (A). In this example, this clock skew or difference in arrival time for this path is called $\Delta b$.

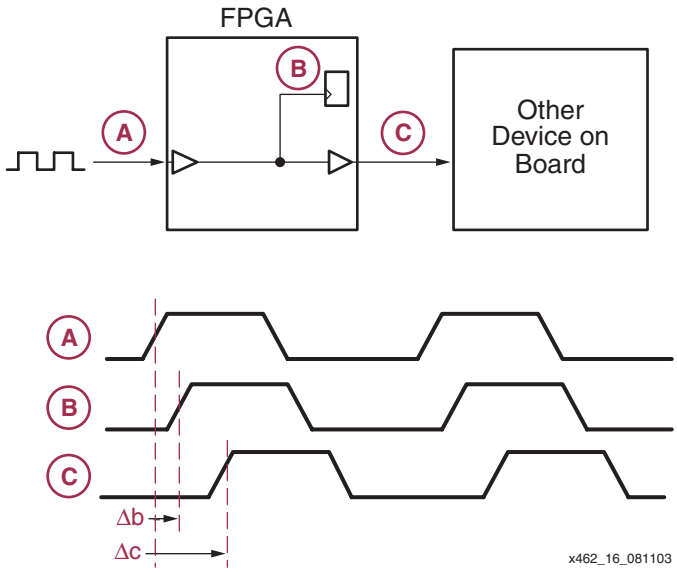



*Figure 16:* **Clock Skew Inherently Exists in Every Synchronous System**

Similarly, the clock source is rebuffered in the FPGA and drives another device on the board. In this case, again the clock source enters the FPGA via an input pin, is distributed via the global clock network, feeds an output pin on the FPGA, and finally connects to the other device via a trace on the printed circuit board (PCB). Because there is more total delay in this clock path, the resulting skew, $\Delta c$, is also larger.

## Clock Skew: The Performance Thief

Clock skew potentially reduces the overall performance of the design by increasing setup times and lengthening clock-to-output delays—both of which increase the clock cycle time. Similarly, clock skew might require lengthy hold times on some devices. Otherwise, unreliable operation might result.

## Make it Go Away!

Is there a way to eliminate clock skew? Fortunately, a Digital Clock Manager (DCM) provides such capabilities. Figure 17 shows the same example design as Figure 16, except this time implemented in a Spartan-3 FPGA. Two DCMs eliminate the clock skew: One DCM eliminates the skew for clocked items within the FPGA, the other DCM eliminates the skew when clocking the other device on the board. The result is practically ideal alignment between the clock at Points (A), (B), and (C)!

How is clock skew elimination accomplished? Remember, clock skew is caused by the delay in the clock path. In Figure 17, the clock at Point (B) was skewed by $\Delta b$ and the clock at Point (C) was skewed by $\Delta c$. What if there was a way to provide Point (B) with an early version of the clock, advanced by $\Delta b$ and a way to provide Point (C) with an early version of the clock, advanced by $\Delta c$? The result would be that all clocks would arrive at their destinations with perfect clock edge alignment. Such perfect alignment reduces setup times, shortens clock-to-output delays, and increases overall system performance.

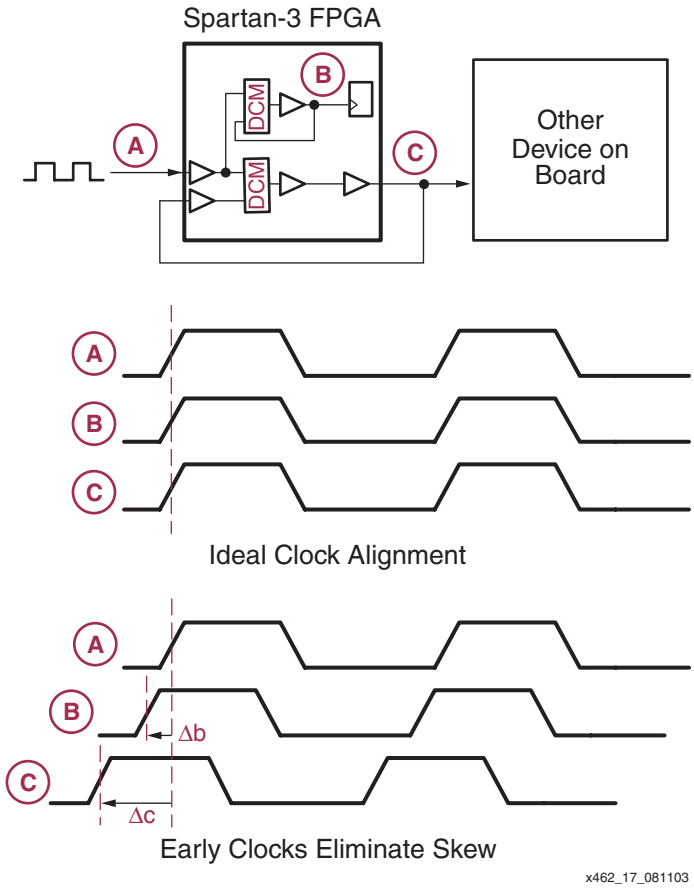



*Figure 17:* **Eliminating Clock Skew in a Spartan-3 FPGA Design**

## Predicting the Future by Closely Examining the Past

Even though Spartan-3 FPGAs employ highly advanced digital logic, unfortunately they cannot predict the future. However, a DCM applies its knowledge of the past behavior of the clock to predict the future. Most input clocks to a system have a never-changing, monotonic frequency. Consequently, the input clock has a nearly constant period, T.

Because it is impossible to insert a negative delay to counteract the clock skew, the DCM *delays* the clocks enough so that they appear to be advanced in time. How is this accomplished? The clock cycle is repetitive and has a fixed period, T. As shown in Figure 18, the clock at Point (B) appears to be advanced in time by the delay Δb. In reality however, the clock is delayed by (T – Δb). Similarly, the clock at Point (B) is delayed by (T – Δc).

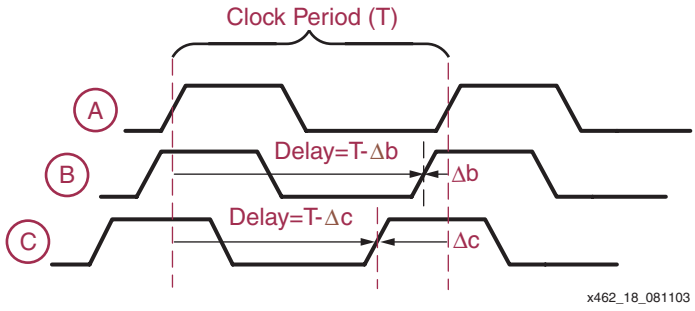



*Figure 18:* **Delaying a Fixed Frequency Clock Appears to Predict the Future**

The clock period, T, is easy to derive knowing the frequency of the incoming monotonic clock signal. But what are the clock skew delays $\Delta b$ and $\Delta c$? With careful analysis, they can be determined after examining the behavior of multiple systems under different conditions. In reality, this is impractical. Furthermore, the values of $\Delta b$ and $\Delta c$ are different between devices and vary with temperature and voltage on the same device.

Instead of attempting to determine the $\Delta b$ and $\Delta c$ delays in advance, the Spartan-3 DCM employs a Delay-Locked Loop (DLL) that constantly monitors the delay via a feedback loop, as shown in Figure 17. In this particular example, two DCMs are required—one to compensate for the clock skew to internal signals and another to compensate for the skew to external devices, each with their own clock feedback loop. The DLL constantly adapts to subtle changes caused by temperature and voltage.

## Locked on Target

In order to determine and insert the correct delay, the DCM samples up to several thousand clock cycles. Once the DCM inserts the correct delay, the DCM asserts its LOCKED output signal.

Do not use the DCM clock outputs until the DCM asserts its LOCKED signal. Until the DCM locks onto the input clock signal, the output clocks are invalid. While the DCM attempts to lock onto the clock signal, the output clocks can exhibit glitches, spikes, or other spurious movements.

In an application, the LOCKED signal qualifies the output clock. Think of LOCKED as a “clock signal good” indicator.

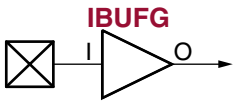
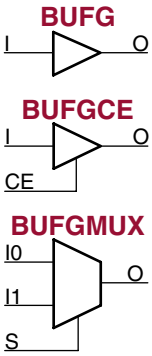
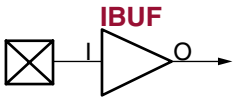
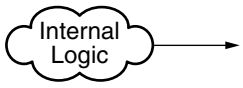
## A Stable, Monotonic Clock Input

To operate properly, the DCM requires a stable, monotonic clock input. Consequently, the DCM can predict future clock periods and adjust the output clock timing appropriately. Once locked, the DCM tolerates clock period variations up to the value specified in the Spartan-3 Data Sheet. See “DCM Clock Requirements” section.

Should the input clock vary well outside the specified limits, the DCM loses lock and the LOCKED output switches Low. If the DCM loses lock, reset the DCM to reacquire lock. If the input clock stays within the specified limits, then the output clocks always are valid when the LOCKED output is High. However, it is possible for the clock to stray well outside the limits, for the LOCKED output to stay High, and for either the CLKDV or CLKFX outputs to be invalid. In short, a stable, monotonic clock input guarantees problem-free designs.

The recommended input path to a DCM’s CLKIN input is via one of the four global buffer inputs (IBUFG) along the same half of the device. Using the IBUFG path, the delay from the pad, through the global buffer, to the DCM is eliminated from the deskewed output. Other paths are possible, however, as shown in Table 11. The signal driving the CLKIN input can also originate a general-purpose input pin (IBUF primitive) via general-purpose interconnect, from in global buffer input (IBUFG), or from a global buffer multiplexer (BUFGMUX, BUFGCE). Similarly, an LVDS clock input may provide the clock signal. The deskew logic is characterized for a single-ended clock input such as LVCMOS or LVTTL. Differential signals may incur a slight amount of phase error due to I/O timing. See the Spartan-3 Data Sheet, Module 3 for specific I/O timing differences.

*Table 11:* **CLKIN Input Sources**

| CLKIN Source | Description |
|---|---|
| Via global buffer input   | A global buffer input, IBUFG, is the preferred source for an external clock to the DCM. The delay from the pad, through the global buffer, to the CLKIN input is characterized, and this delay is removed from the deskewed clock output. |
| Global Clock Buffer   | A global clock buffer, using either a BUFG, BUFGCE, or BUFGMUX primitive, is a preferred source for an internally generated clock to the DCM. The delay through the global buffer is characterized, and this delay is removed from the deskewed clock output. |
| Via general-purpose I/O   | Any user-I/O pin, IBUF, becomes an alternate source for an external clock. The pad-to-DCM delay cannot be predetermined due to the numerous potential input paths, and consequently, the delay is not compensated by the DCM. |
| Derived from internal logic   | Logic within the FPGA also may be the clock source. Again, the logic-to-DCM delay cannot be predetermined as it is not compensated by the DCM. |

## Feedback from a Reliable Source

In order to lock in on the proper delay, the DCM monitors both the incoming clock and a feedback clock, tapped after the clock distribution delay. There are no restrictions on the total delay in the clock feedback path. If required, the DLL effectively delays the output clock by multiple clock periods. Consequently, a DCM can compensate for either internal or external delays, but the clock feedback must connect to the correct feedback point.

## Removing Skew from an Internal Clock

To eliminate skew within the FPGA, the feedback tap is the same clock as that seen by the clocked elements within the FPGA, shown in Figure 19. The feedback clock is typically the CLK0 output (no phase shift) from the DCM, connected to the output of a global clock buffer (BUFG) or a global clock multiplexer (BUFGMUX or BUFGCE primitive) on the same edge of the device. Alternatively, the DCM's CLK2X output (no phase shift, frequency doubled) may be used instead of the CLK0 output.

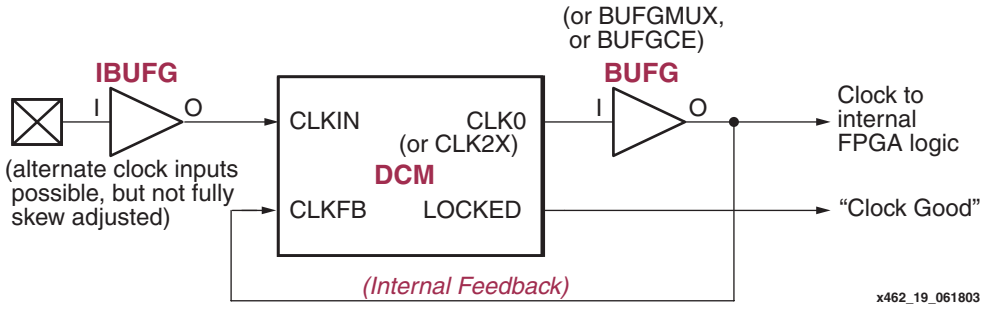



*Figure 19:* **Eliminating Skew on Internal Clock Signals**

## Removing Skew from an External Clock

Constructing the DCM feedback for an external clock is slightly more complex. Ideally, the clock feedback originates from the point where the signal feeds any external clocked inputs, after any long printed-circuit board traces or external clock rebuffering, as shown in Figure 20.

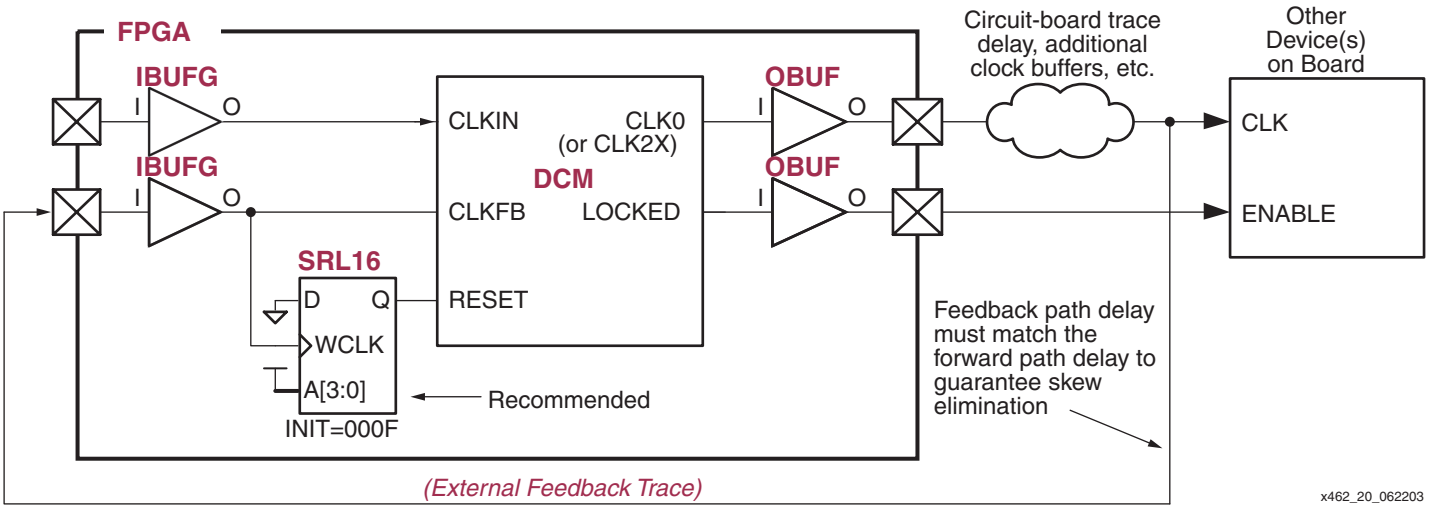



*Figure 20:* **Eliminating Skew on External Clock Signals**

The LOCKED signal indicates when the DCM achieves lock, qualifying the clock signal. The LOCKED signal can enable external devices or an inverted version can connect to an active-Low chip enable.

### Reset DCM After Configuration

When using external feedback, apply a reset pulse to the DCM immediately after configuration to ensure consistent locking. An SRL16 primitive, initialized with 0x000F, supplies the necessary reset pulse, as shown in Figure 20. See "RST Input Behavior."

### Why Reset?

Why is this extra reset pulse required? For an optimum locking process, a DCM configured with external feedback requires both the CLKIN and either the CLK0 or CLK2X signals to be present and stable when the DCM begins to lock. During the configuration process, the external feedback, CLKFB, is not available because the FPGA's I/O buffers are not yet active.

At the end of configuration, the DCM begins the capture process once the device enters the startup sequence. Because the FPGA's global 3-state signal (GTS) still is asserted at this time, any output pins remain in a 3-state (high-impedance, floating) condition. Consequently, the CLKFB signal is in an unknown logic state.

When CLKFB eventually appears after the GTS is deasserted, the DCM proceeds to capture. However, without the reset pulse, the DCM might not lock at the optimal point, which potentially introduce slightly more jitter and greater clock cycle latency through the DCM.

Without the reset, another possible issue might occur if the CLKFB signal, while in the 3-state condition, cross-couples with another signal on the board due to a printed-circuit board signal integrity problem. The DCM might sense this invalid cross-coupled signal as CLKFB and use it to proceed with a lock. This possibly prevents the DCM from properly locking once the GTS signal deasserts and the true CLKFB signal appears.

## What is a Delay-Locked Loop?

Two basic types of circuits remove clock delay:

- Delay-Locked Loops (DLLs) and
- Phase-Locked Loops (PLLs)

In addition to their primary function of removing clock distribution delay, DLLs and PLLs typically provide additional functionality such as frequency synthesis, clock conditioning, and phase shifting.

### Delay-Locked Loop (DLL)

As shown in Figure 21, a DLL in its simplest form consists of a tapped 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 continuously samples the input clock as well as the feedback clock to properly adjust the delay line. Delay lines are constructed either using a voltage controlled delay or as a series of discrete delay elements. For best, ruggedly stable performance, the Spartan-3 DLL uses an all-digital delay line.

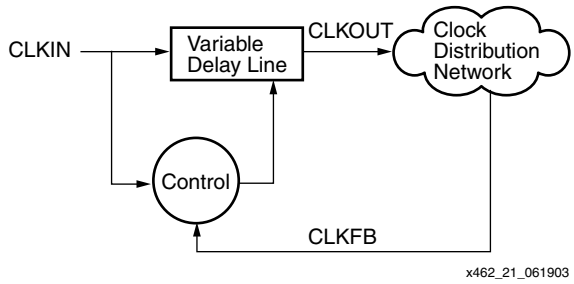



*Figure 21:* **Delay-Locked Loop (DLL) Block Diagram**

A DLL works by inserting delay between the input clock and the feedback clock until the two rising edges align, effectively delaying the feedback clock by almost an entire period—minus the clock distribution delay, of course. In DLL and PLL parlance, the feedback clock is 360° out of phase, which means that they appear to be exactly in phase again.

After the edges from the input clock line up with the edges from the feedback clock, the DLL “locks”, and the two clocks have no discernible difference. Thus, the DLL output clock compensates for the delay in the clock distribution network, effectively removing the delay between the source clock and its loads. Voila!

**Phase-Locked Loop (PLL)**

While designed for the same basic function, a PLL uses a different architecture to accomplish the task. As shown in Figure 22, the fundamental difference between the PLL and DLL is that instead of a delay line, the PLL uses a voltage-controlled oscillator, which generates a clock signal that approximates the input clock CLKIN. The control logic, consisting of a phase detector and filter, adjusts the oscillator frequency and phase to compensate for the clock distribution delay. The PLL control logic compares the input clock to the feedback clock CLKFB and adjusts the oscillator clock until the rising edge of the input clock aligns with the feedback clock. The PLL then "locks."

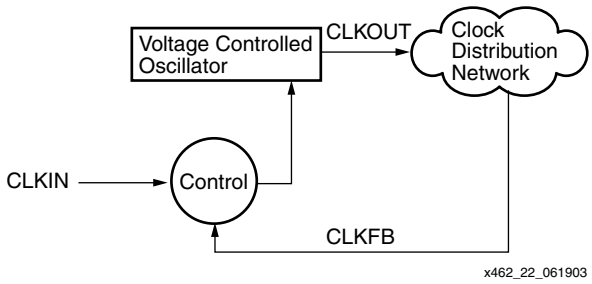



*Figure 22:* **Phase-Locked Loop (PLL) Block Diagram**

**Implementation**

A DLL or PLL is assembled using either analog or digital circuitry; each approach has its own advantages. An analog implementation with careful circuit design produces a DLL or PLL with a finer timing resolution. Additionally, analog implementations sometimes consume less silicon area.

Conversely, digital implementations offer advantages in noise immunity, lower power consumption and better jitter performance. Digital implementations also provide the ability to stop the clock, facilitating power management. Analog implementations can require additional power supplies, require close control of the power supply, and pose problems in migrating to new process technologies.

**DLL vs. PLL**

When choosing between a PLL or a DLL for a particular application, understand the differences in the architectures. The oscillator used in the PLL inherently introduces some instability, which degrades the performance of the PLL when attempting to compensate for the delay of the clock distribution network. Conversely, the unconditionally stable DLL architecture excels at delay compensation and clock conditioning. On the other hand, the PLL typically has more flexibility when synthesizing a new clock frequency.

## Skew Adjustment

Most of this section discusses how to remove skew and how to phase align an internal or external clock to the clock source. In actuality, the DCM purposely adds a small amount of skew via an advanced attribute called DESKEW_ADJUST. In DCM Wizard, the DESKEW_ADJUST attribute is controlled via the Advanced Options window.

There are two primary applications for this attribute, SYSTEM_SYNCHRONOUS and SOURCE_SYNCHRONOUS. The overwhelming majority of applications use the default SYSTEM_SYNCHRONOUS setting. The purpose of each mode is described below.

### System Synchronous

In a Source Synchronous system, all devices within a data path share a common clock source, as shown in Figure 23. This is the traditional and most-common system configuration. The SYSTEM_SYNCHRONOUS option, which is the default value, adds a small amount of clock delay so that there is zero hold time when capturing data. Hold time is essentially the timing different between the best-case data path and the worst-case clock path. The DCM's clock skew elimination function advances the clock, essentially dramatically shortening the worst-case clock path. However, if the clock path is advance so far that the clock appears before the

data, then hold time results. The SYSTEM_SYNCHRONOUS setting injects enough additional skew on the clock path to guarantee zero hold times, but at the expense of a slightly longer clock-to-output time.

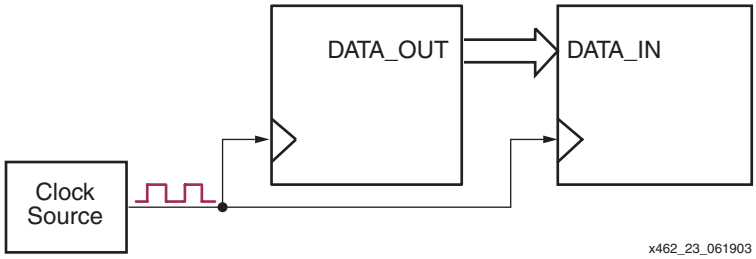



*Figure 23:* **System-Synchronous Applications are Clocked by a Single, System-Wide Clock Source**

### Source Synchronous

SOURCE_SYNCHRONOUS mode is an advanced setting, used primarily in high-speed data communications interfaces. In Source Synchronous applications, both the data and the clock are derived from the same clock source, as shown in Figure 24. The transmitting devices sends the both data and clock to the receiving device. The receiving device then adjusts the clock timing for best data reception. High-speed Dual-Data Rate (DDR) and LVDS connections are examples of such systems.

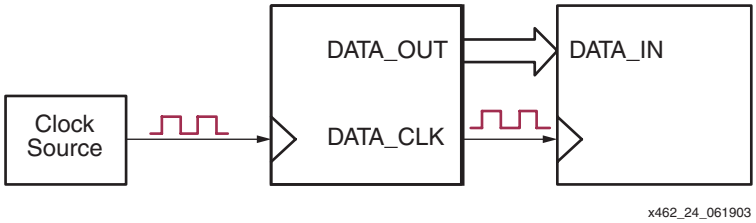



*Figure 24:* **In Source-Synchronous Applications, the Data Clock is Provided by the Data Source**

The SOURCE_SYNCHRONOUS setting essentially zeros out any phase difference between the incoming clock and the deskewed output clock from the DCM. The FPGA application must then adjust the clock timing using either the Fixed or Dynamic Fine Phase Shift mode. The following application notes provide additional information on Source Synchronous design and using dynamic phase alignment:

- XAPP268: *Dynamic Phase Alignment*
  http://www.xilinx.com/xapp/xapp268.pdf
- XAPP622: *SDR LVDS Transmitter/Receiver*
  http://www.xilinx.com/xapp/xapp622.pdf

Similarly, the following application note delves into more details on system-level timing. Although the application note is written for the Virtex-II and Virtex-II Pro FPGA architectures, most of the concepts apply directly to Spartan-3 FPGAs.

- XAPP259: *System Interface Timing Parameters*
  http://www.xilinx.com/xapp/xapp259.pdf

### Timing Comparisons

Figure 25 compares the effect of both SYSTEM_SYNCHRONOUS and SOURCE_SYNCHRONOUS settings using a Dual-Data Rate (DDR) application. In DDR applications, two data bits appear on each data line—one during the first half-period of the clock, the second during the second half-period.

In SYSTEM_SYNCHRONOUS mode, a small amount of skew is purposely added to the DCM clock path so that there is zero hold time.

In SOURCE_SYNCHRONOUS mode, no additional skew is inserted to the DCM clock path. However, the FPGA application must insert additional skew or phase shifting so that the clock appears at the ideal location in the data window.

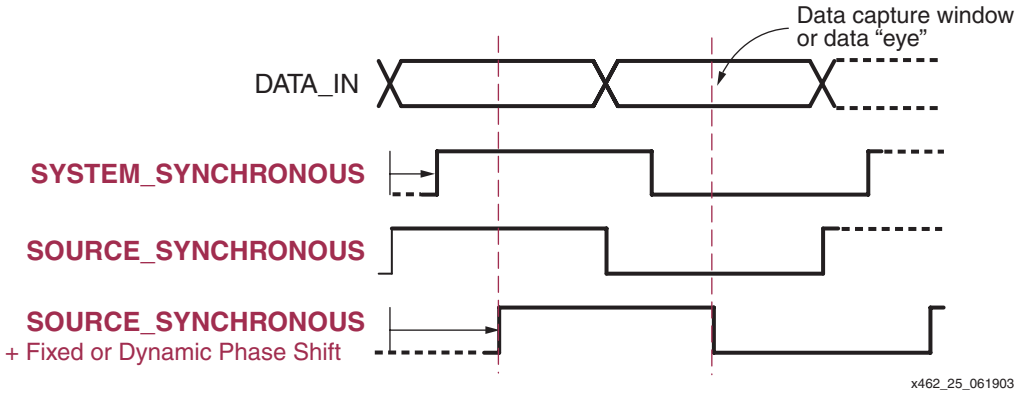



*Figure 25:* **Comparing SYSTEM_SYNCHRONOUS and SOURCE_SYNCHRONOUS Timing in a Dual-Data Rate (DDR) Application**

## Clock Conditioning

Clock conditioning is a function where an incoming clock with a duty cycle other than 50% is reshaped to have a 50% duty cycle. Figure 26 shows an example where an incoming clock, with roughly a 40% High time and a 60% Low time (40%/60% duty cycle), is reshaped into a nearly perfect 50% duty cycle—nearly perfect because there is some residual duty-cycle distortion specified by the CLKOUT_DUTY_CYCLE_DLL and CLKOUT_DUTY_CYCLE_FX values in the Spartan-3 Data Sheet. The distortion is estimated at less than 150 ps.

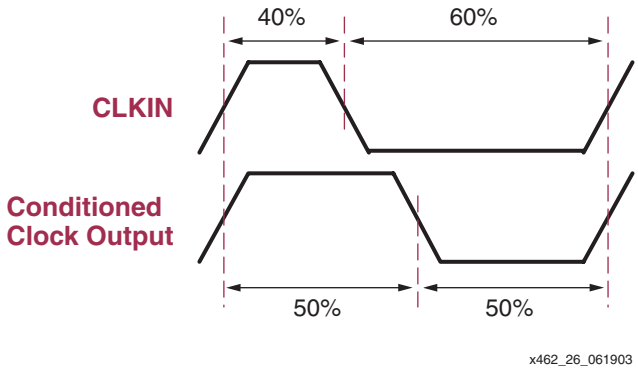



*Figure 26:* **DCM Duty-Cycle Correction Feature Provides 50% Duty Cycle Outputs**

Clocks with 50% duty cycle are mandatory for high-speed communications interfaces such as LVDS or Dual-Data Rate (DDR) and for clock forwarding or clock mirroring applications. See “Dual-Data Rate (DDR) Clocking Example.”

The DCM automatically conditions most clock outputs so that they have a 50% duty cycle. Other clock outputs are optionally conditioned, depending either on the operating conditions or on attribute settings, as shown in Table 12.

*Table 12:* **Conditioned Clock Output with 50% Duty Cycle**

| DCM Clock Output | 50% Duty Cycle Output |
|---|---|
| CLK0<br>CLK90 | When DUTY_CYCLE_CORRECTION attribute set to TRUE |
| CLK180<br>CLK270 | **DLL_FREQUENCY_MODE Attribute**<br>**LOW**: When DUTY_CYCLE_CORRECTION attribute set to TRUE<br>**HIGH**: Outputs not available |
| CLK2X<br>CLK2X180 | **DLL_FREQUENCY_MODE Attribute**<br>**LOW**: Always<br>**HIGH**: Outputs not available |
| CLKDV | **DLL_FREQUENCY_MODE Attribute**<br>**LOW**: Always<br>**HIGH**: When CLKDV_DIVIDE attribute is an integer value |
| CLKFX<br>CLKFX180 | Always |

The Quadrant Phase Shifted Outputs, CLK0, CLK90, CLK180, and CLK270 have optional clock conditioning, controlled by the DUTY_CYCLE_CORRECTION attribute. By default, the DUTY_CYCLE_CORRECTION attribute is set to TRUE, meaning that these outputs are conditioned to a 50% duty cycle. Setting this attribute to FALSE disables the clock-conditioning feature, in which case the effected clock outputs have roughly the same duty cycle as the incoming clock. Exact replication of the CLKIN duty cycle is not guaranteed.

## Phase Shifting – Delaying the Clock by a Fraction of a Period

A DCM also optionally phase shifts an incoming clock, effectively delaying the clock by a fraction of the clock period.

The DCM supports four different types of phase shifting. Each type may be used independently, or in conjunction with other phase shifting modes. The phase shift capabilities for each clock output appear in Table 13.

1. Half-Period Phase Shifted Outputs, most with conditioned 50% duty cycle. A pair of outputs provides a rising edge at 0° and 180° phase shift—or, at the beginning and half-period points during the clock period.
2. Quadrant Phase Shifted Outputs of 0° (CLK0), 90° (CLK90), 180° (CLK180), and 270° (CLK270), with optional 50% duty-cycle conditioning.
3. Fixed Fine Phase Shifting of all DCM clock outputs with a resolution of 1/256$^{th}$ of a clock cycle.
4. Dynamic Fine Phase Shifting of all DCM clock outputs from within the FPGA application, again with a resolution of 1/256$^{th}$ of a clock cycle.

*Table 13:* **Phase Shift Capabilities by Clock Output**

| Clock Output | Half-Period | Quadrant | Fixed or Dynamic |
|---|---|---|---|
| CLK0 | ✓ | ✓ | ✓ |
| CLK90 | | ✓ | ✓ |
| CLK180 | ✓ | ✓ | ✓ |
| CLK270 | | ✓ | ✓ |
| CLK2X | ✓ | | ✓ |
| CLK2X180 | ✓ | | ✓ |
| CLKDV | | | ✓ |
| CLKFX | ✓ | | ✓ |
| CLKFX180 | ✓ | | ✓ |

## Half-Period Phase Shifted Outputs

The Half-Period Phase Shift outputs provide a non-shifted clock output, and the equivalent clock output but shifted by half a period (180° phase shift). The Half-Period Phase Shift outputs appear in pairs, as shown in Table 14.

*Table 14:* **Half-Period Phase Shifted Outputs**

| Output Pairs | | Comment |
|---|---|---|
| No Phase Shift | 180° Phase Shift | |
| CLK0 | CLK180 | Same frequency as CLKIN input. 50% duty cycle corrected by default and controlled by the DUTY_CYCLE_CORRECTION attribute. |
| CLK2X | CLK2X180 | Outputs from the Clock Doubler (CLK2X, CLK2X180). Twice the frequency of the CLKIN input, always with 50% duty cycle. |
| CLKFX | CLKFX180 | Outputs from the Frequency Synthesizer (CLKFX, CLKFX180). Output frequency depends on Frequency Synthesizer attributes. Always with 50% duty cycle. |

The Half-Period Phase Shift outputs are ideal for duty-cycle critical applications such as high-speed Dual-Data Rate (DDR) designs and clock mirrors. The Half-Period Phase Shift output pairs provide two clocks, one with a rising edge at the beginning of the clock period, and another rising edge precisely aligned at half the clock period, as shown in Figure 27.

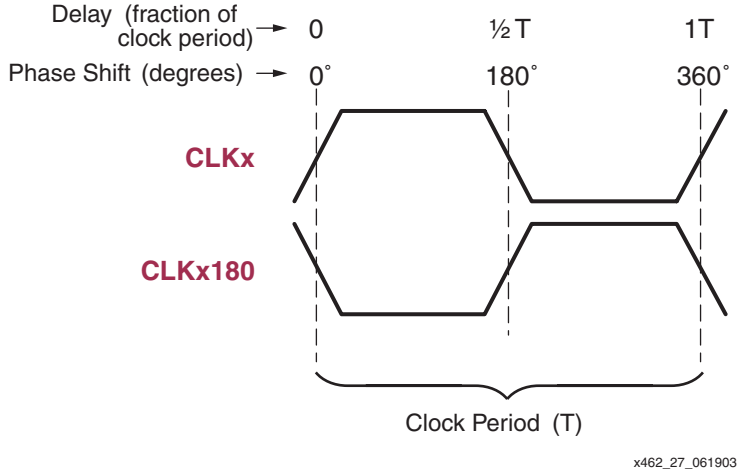



*Figure 27:* **Half-Period Phase Shift Outputs**

### Half-Period Phase Shift Outputs Reduce Duty-Cycle Distortion

When the DCM clock outputs are duty-cycle corrected to 50%, it appears that the 180° phase-shifted clock is just an inverted version on the non-shifted clock. For low-frequency applications, this is essentially true.

However, at very high operating frequencies, duty-cycle distortion—due to differences in rise and fall times of individual transistors—becomes relevant within the FPGA device. Starting with a 50% clock cycle, such distortion causes differences between the clock High and clock Low times, which is consistent from cycle to cycle.

### Dual-Data Rate (DDR) Clocking Example

In Figure 28, a single DCM clock output, CLKx, drives both clocks on a Dual-Data Rate (DDR) output flip-flop. One DDR clock input uses the clock output as is, the other input inverts the clock within the DDR flip-flop. The CLKx output from the DCM has a 50% duty cycle, but after traveling through the FPGA's clock network, the duty cycle becomes slightly distorted. In this exaggerated example, the distortion truncates the clock High time and elongates the clock Low time. Consequently, the C1 clock input triggers slightly before half the clock period. At lower frequencies, this distortion is usually negligible. However, high-performance DDR-based systems require precise clocking.

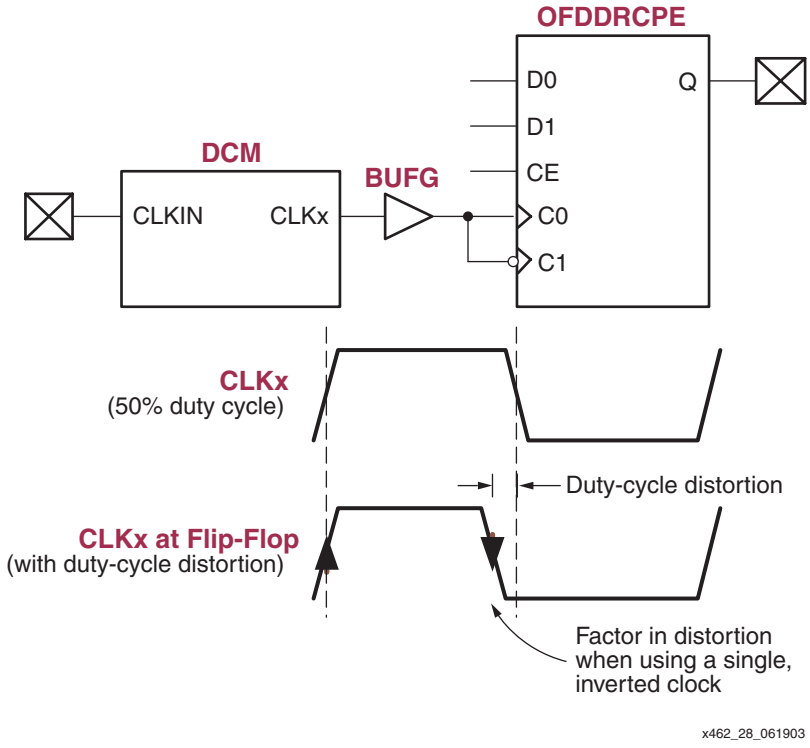



*Figure 28:* **Dual-Data Rate (DDR) Output Using Both Edges of a Single Clock Induces Duty-Cycle Distortion**

Figure 29 shows a slightly modified circuit compared to Figure 28. In this case, the DCM provides both a non-shifted and a 180° phase-shifted output to the DDR output flip-flop. The CLKx clock signal precisely triggers the DDR flip-flop's C0 input at the start of the clock period. Similarly, the CLKx180 clock signal precisely triggers the DDR flip-flop's C1 input halfway through the clock period. The cost of this approach is an additional global buffer and global clock line, but it potentially reduces the potential duty-cycle distortion by approximately 300 ps (this value is an estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value).

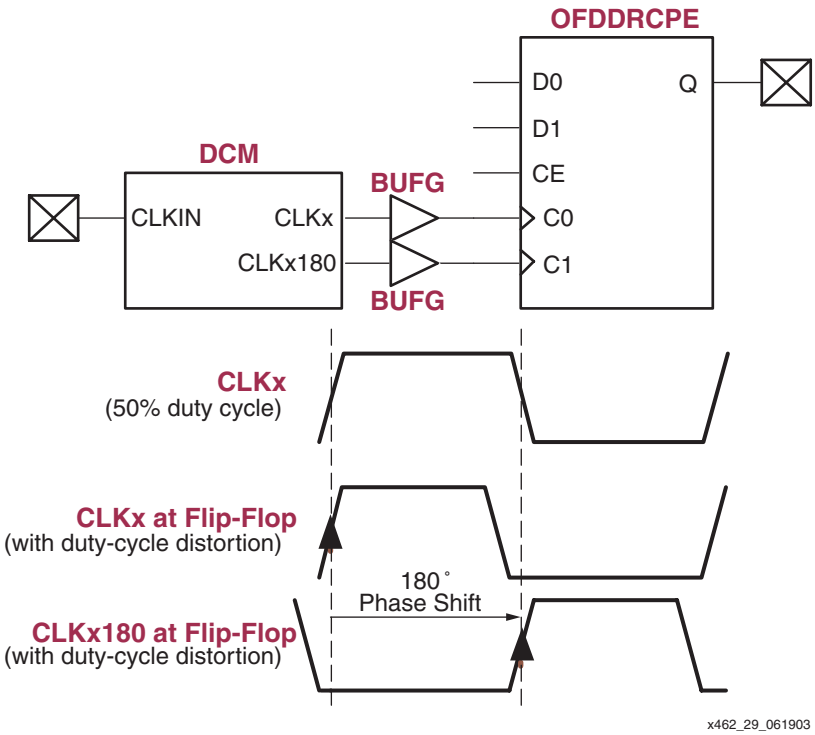



*Figure 29:* **Using Half-Period Phase Shift Outputs Reduces Potential Duty-Cycle Distortion**

Table 15 shows the specified duty-cycle distortion values as measured using DDR output flip-flops and LVDS outputs. There may be additional distortion on other output types caused by asymmetrical rise and fall times, which can be simulated using IBIS.

*Table 15:* **Duty-Cycle Distortion Parameters**

| Parameter | Description | Estimated Value |
|---|---|---|
| $T_{DCD\_CLK0}$ | Duty-cycle distortion when local inversion provides negative-edge clock to DDR element in an I/O block. See Figure 28. | ~400 ps* |
| $T_{DCD\_CLK180}$ | Duty-cycle distortion when DCM CLKx180 output provides clock to DDR element in an I/O block. See Figure 29. | ~60 ps* |

* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

## Quadrant Phase Shifted Outputs

The Quadrant Phase Shift outputs shift the CLKIN input, each by a quarter period, as shown in Figure 30 and Table 17. Because the Quadrant Phase Shift outputs require a feedback path back to the CLKFB input, the CLK0 output is phase aligned to the rising edge of the CLKIN input. The CLK90 output is phase shifted 90° from the CLKIN input, and so forth.

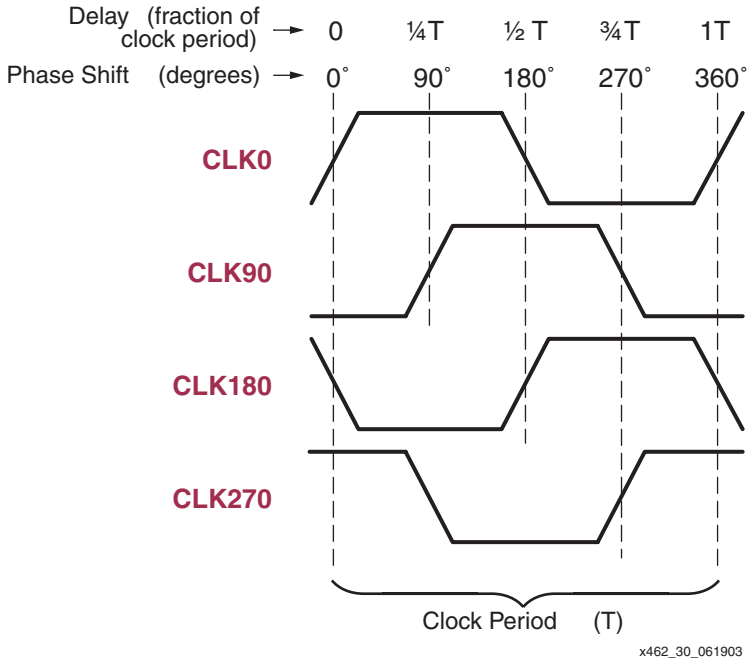



*Figure 30:* **Quadrant Phase Shift Outputs Shift CLKIN, Each by a Quarter Period (Shown with Duty-Cycle Correction Enabled)**

### Output Availability Depends on DLL Frequency Mode

The availability of the Quadrant Phase Shift outputs depends on the DLL's frequency mode, controlled by the DLL_FREQUENCY_MODE attribute. All four Quadrant Phase Shift outputs are available in low-frequency mode (DLL_FREQUENCY_MODE=LOW), as shown in Table 16. Only the CLK0 and CLK180 outputs are available in both modes.

*Table 16:* **Quadrant Phase Shift Output Availability by DLL Frequency Mode**

| Output | DLL_FREQUENCY_MODE | |
|---|---|---|
| | LOW | HIGH |
| CLK0 | ✓ | ✓ |
| CLK90 | ✓ | |
| CLK180 | ✓ | ✓ |
| CLK270 | ✓ | |

### Optional 50/50 Duty Cycle Correction

As a group, the Quadrant Phase Shift outputs are optionally conditioned to a 50% duty cycle, controlled by the DUTY_CYCLE_CORRECT attribute. When TRUE, which is the default, all four outputs have a 50% duty cycle. When FALSE, the outputs do not have the same duty cycle as the CLKIN input. See the "Clock Conditioning" section for more information.

### Four Phases, Delayed Clock Edges, Phased Pulses

One view of the Quadrant Phase Shift outputs is that each provides a rising clock that is delay one quarter period from the preceding pulse, as shown in Table 17. These outputs provide flexible timing for such applications as memory interfaces and peripheral control.

With the duty-cycle correction option enabled (DUTY_CYCLE_CORRECTION = TRUE), there are other ways to view the outputs. For example, the outputs also provide falling-edge clocks

separated by a quarter phase. Again, see Table 17. Similarly, each output produces a High-going pulse, and a Low-going pulse, both half a period wide. For example, the CLK90 output shown in Figure 30 produces a High-going pulse, centered within the CLK0 clock period.

*Table 17:* **Quadrant Phase Shift Outputs and Characteristics (DUTY_CYCLE_CORRECTION=TRUE)**

| DCM Output | Phase Shift | Delayed by Period Fraction | Rising Edge | Falling Edge | Comment |
|---|---|---|---|---|---|
| CLK0 | 0° | 0 | 0 | ½T | Deskewed input clock, no phase shift |
| CLK90 | 90° | ¼T | ¼T | ¾T | High-going pulse, ½T wide, in middle of period |
| CLK180 | 180° | ½T | ½T | 0T | Inverted CLK0, rising clock edge in middle of period |
| CLK270 | 270° | ¾T | ¾T | ¼T | Low-going pulse, ½T wide, in middle of period |

## Fine Phase Shifting

The DCM provides additional controls over clock skew using fine phase shifting. Fine-phase adjustment affects all nine DCM output clocks simultaneously. The fine phase shift capability requires the DCM's DLL functional unit. Consequently, clock feedback via the CLKFB input is required.

Physically, the fine phase shift control adjusts the phase relationship between the rising edges of the CLKIN and CLKFB inputs. The net effect, however, is the all DCM outputs are phase shifted with relation to the CLKIN input.

By default, fine phase shifting is disabled (CLKOUT_PHASE_SHIFT=NONE), meaning that the clock outputs are phase aligned with CLKIN. In this case, there is no skew between the input clock, CLKIN, and the feedback clock, measured at the appropriate feedback point (see "Feedback from a Reliable Source" section). When fine phase shifting is enabled, the output clock edges can be phase shifted so that they are advanced or are retarded compared to the CLKIN input, as shown in Figure 31.

There are two fine phase shift modes as described below. Both are commonly used in high-speed data communications applications. See the "Source Synchronous" section.

1. **Fixed Fine Phase Shift** mode sets the phase shift value at design time. The phase shift value is loaded during the FPGA configuration process and cannot be changed by the application.
2. **Dynamic Fine Phase Shift** mode has an initial phase shift value, similar to Fixed Fine Phase Shift, which is set during FPGA configuration. However, the phase shift value can be changed by the application after the DCM's LOCKED output goes High.

## Fixed Fine Phase Shifting

In Fixed Fine Phase Shift mode, the phase shift value is specified at design time and set during the FPGA configuration process. The application cannot change the value during run time.

Two attributes control this mode. The CLKOUT_PHASE_SHIFT attribute is set to FIXED, and the PHASE_SHIFT attribute controls the amount of phase shift. If PHASE_SHIFT is 0, then the output clocks and the CLKIN input are phase aligned, as shown in Figure 31. If PHASE_SHIFT is a negative integer, then the clock output(s) are phase shifted before CLKIN. If PHASE_SHIFT is a positive integer, then the clock output(s) are phase shifted after CLKIN.

### Fixed Fine Phase Shift Range

The PHASE_SHIFT attribute is always an integer value, ranging between –255 and +255. However, the actual limits may be lower depending on the CLKIN input frequency, as described below.

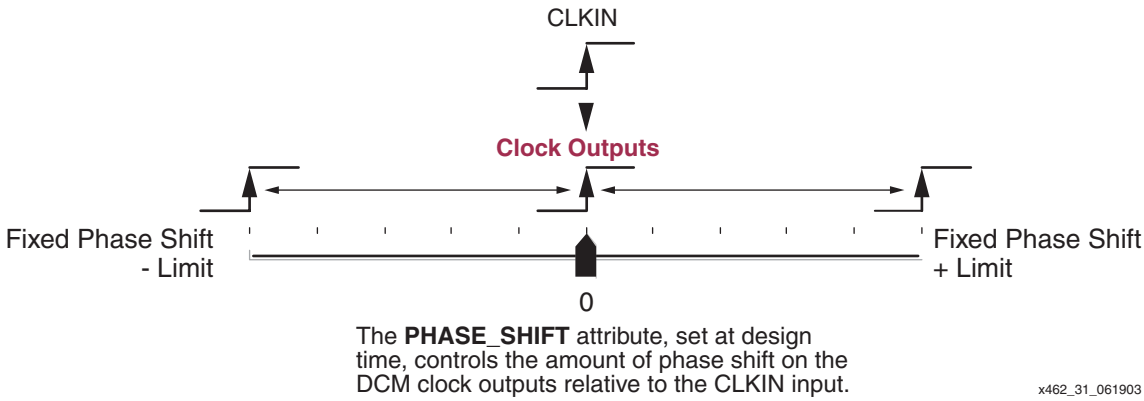



*Figure 31:* **Fixed-Value Fine Phase Shift Control**

The minimum and maximum limit of the PHASE_SHIFT attribute depend on two values.

1. The period of the CLKIN input, $T_{CLKIN}$, measured in nanoseconds.
2. The value of the FINE_SHIFT_RANGE specification for the Spartan-3 device and speed grade, found in the Spartan-3 Data Sheet, Module 3. FINE_SHIFT_RANGE is the total delay achievable by the phase shift delay line, which is a function of the number of delay taps used in the circuit. The actual delay line may be longer than FINE_SHIFT_RANGE, but only the delay up to FINE_SHIFT_RANGE is guaranteed.

Using these two values, calculate the SHIFT_DELAY_RATIO using Equation 1. The limits for the PHASE_SHIFT attribute are different, depending on whether the result is less than or if it is greater than or equal to one.

$$SHIFT\_DELAY\_RATIO = \frac{FINE\_SHIFT\_RANGE}{T_{CLKIN}} \quad \text{Eq. 1}$$

***SHIFT_DELAY_RATIO < 1***

If the clock period is longer than the specified FINE_SHIFT_RANGE, then the SHIFT_DELAY_RATIO < 1, meaning that maximum fine phase shift is limited by FINE_SHIFT_RANGE. When SHIFT_DELAY_RATIO < 1, then the PHASE_SHIFT limits are set according to Equation 2:

$$PHASE\_SHIFT_{LIMITS} = \pm \left[ INTEGER\left( 256 \bullet \frac{FINE\_SHIFT\_RANGE}{T_{CLKIN}} \right) \right] \quad \text{Eq. 2}$$

For example, assume that $F_{CLKIN}$ is 75 MHz ($T_{CLKIN}$ = 13.33 ns) and FINE_SHIFT_RANGE is 10.00 ns(1). In this case, the PHASE_SHIFT value is limited to ±191.

Consequently, the phase shift value when SHIFT_DELAY_RATIO < 1 is shown by Equation 3. To determine the phase shift resolution, set PHASE_SHIFT = 1.

$$T_{PhaseShift} = \left( \frac{PHASE\_SHIFT}{\left| PHASE\_SHIFT_{LIMITS} \right|} \right) \bullet FINE\_SHIFT\_RANGE \quad \text{Eq. 3}$$

1. Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

***SHIFT_DELAY_RATIO ≥ 1***

By contrast, if the clock period is shorter than the specified FINE_SHIFT_RANGE, then the SHIFT_DELAY_RATIO ≥ 1, meaning that maximum fine phase shift is limited to ±255.

$$PHASE\_SHIFT_{LIMITS} = \pm 255 \quad \text{Eq. 4}$$

Consequently, the phase shift value when SHIFT_DELAY_RATIO ≥ 1 is shown by Equation 5. To determine the phase shift resolution, set PHASE_SHIFT = 1.

$$T_{PhaseShift} = \left(\frac{PHASE\_SHIFT}{256}\right) \bullet T_{CLKIN} \quad \text{Eq. 5}$$

### Minimum Phase Shift Size

The minimum phase shift size is controlled by the greater of two limiting factors.

1. The minimum delay-line tap resolution, listed as the DCM_TAP_MIN specification in the Spartan-3 Data Sheet (estimated at ~30ps), or
2. 1/256$^{th}$ of the clock period.

### Other Design Considerations

In Fixed Phase Shift mode, the Dynamic Phase Shift control inputs must be tied to GND, which DCM Wizard does automatically.

### DCM Wizard

To use Fixed Phase Shift mode, click **Fixed** in the Phase Shift section of DCM Wizard's General Setup panel, shown in Figure 32. This action sets the CLKOUT_PHASE_SHIFT attribute to FIXED.

Enter the **Phase Shift Value**, which must be an integer within the limits described above. This action sets the PHASE_SHIFT attribute value. DCM Wizard checks that the phase shift value is within the limits.

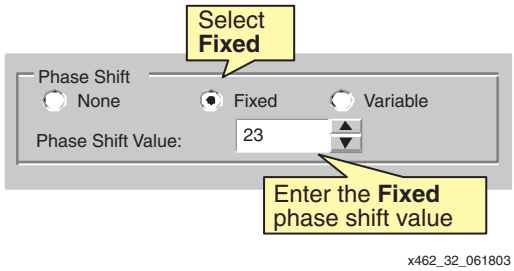



*Figure 32:* **Selecting Fixed Fine Shift Mode**

## Dynamic Fine Phase Shifting

In Dynamic Fine Phase Shift mode, the initial skew or phase shift is still controlled by the PHASE_SHIFT attribute during configuration, just as it is for Fixed Fine Shift mode. However, in dynamic mode, the FPGA application can adjust the current phase shift location after the DCM's LOCKED output goes High using the Dynamic Fine Phase Shift control inputs, PSEN, PSCLK, and PSINCDEC.

### Operation

Use the phase shift control inputs to adjust the current phase shift value, as shown in Figure 33. The rising edge of PSCLK synchronizes all dynamic phase shift operations. A valid operation starts by asserting the PSEN enable input for one and only one PSCLK clock period. Asserting PSEN for more than one rising PSCLK clock edge may cause undesired behavior.

The value on the PSINCDEC increment/decrement control input determines the phase shift direction. When PSINCDEC is High, the present dynamic phase shift value is incremented by one unit. Similarly, when PSINCDEC is Low, the present dynamic phase shift value is decremented by one unit.

The actual phase shift operation timing varies and the operation completes when the DCM asserts the PSDONE output High for a single PSCLK clock period. Between enabling PSEN until PSDONE is asserted, the DCM output clocks slide, bit by bit, from their original phase shift value to their new phase shift value. During this time, the DCM remains locked on the incoming clock and continues to assert its LOCKED output.

When or after PSDONE is asserted High, the next phase shift operation can be initiated.

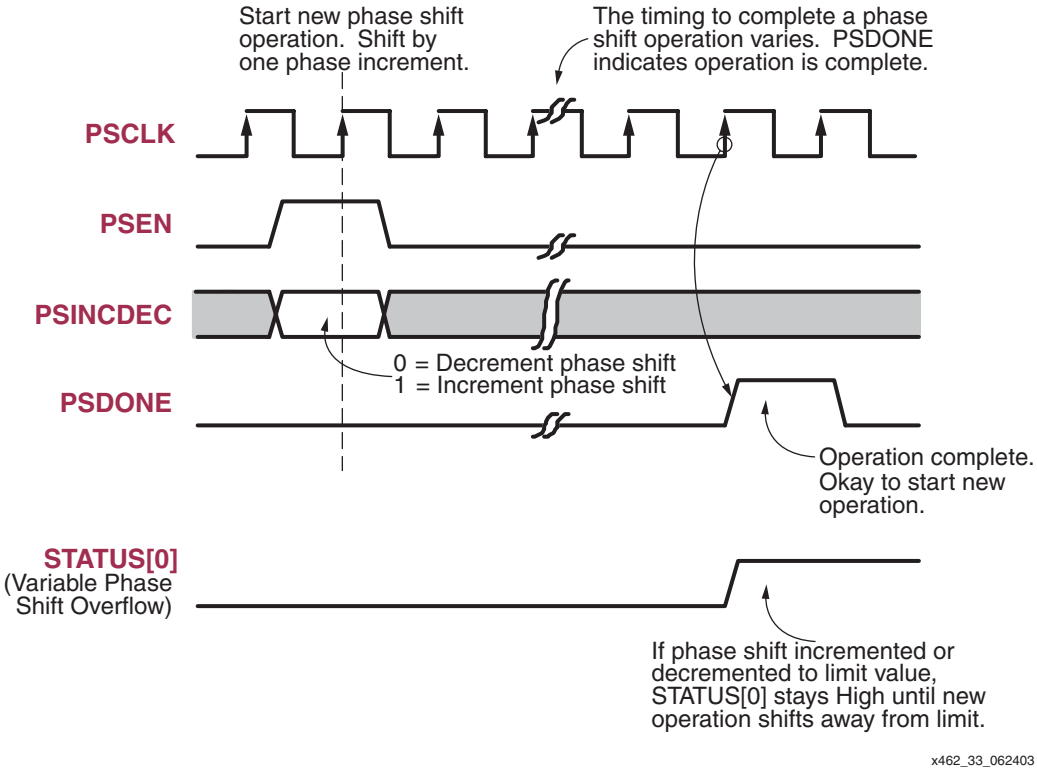



*Figure 33:* **Dynamic Fine Phase Shift Control Interface**

To enable Dynamic Fine Phase Shift mode, set the CLKOUT_PHASE_SHIFT attribute to VARIABLE. The PHASE_SHIFT attribute value sets the initial phase shift location, established after FPGA configuration. The FPGA application can the dynamically adjust the skew or phase shift on the DCM's output clocks after the DCM's LOCKED output goes High. If the DCM is reset, the PHASE_SHIFT value reverts to its initial configuration value.

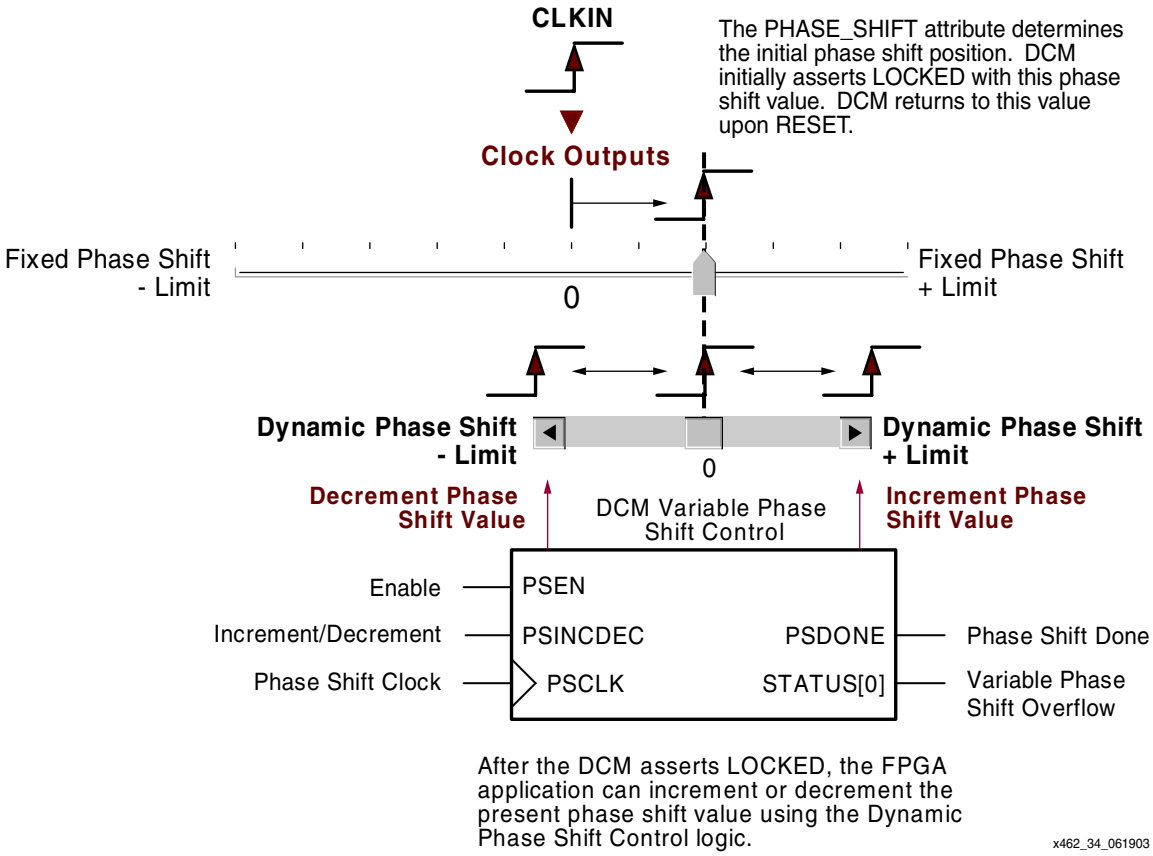



*Figure 34:* **Dynamic Phase Shift Controls**

### Dynamic Fine Phase Shift Range

Just as the PHASE_SHIFT has a minimum and maximum phase shift limit, so does the Dynamic Phase Shift, as shown in Figure 34. Similarly, the limit depends on the ratio of the FINE_SHIFT_RANGE versus the input clock period, as calculated by the SHIFT_DELAY_RATIO equation above.

#### *SHIFT_DELAY_RATIO < 2*

If the specified FINE_SHIFT_RANGE value is less than twice the clock period (SHIFT_DELAY_RATIO < 2), then the maximum dynamic fine phase shift value is limited by FINE_SHIFT_RANGE, the maximum delay tap value. When SHIFT_DELAY_RATIO < 2, then the dynamic phase shift limits are set according to Equation 6.

$$DynamicPhaseShift_{LIMITS} = \pm\left[INTEGER\left(128 \bullet \frac{FINE\_SHIFT\_RANGE}{T_{CLKIN}}\right)\right] \quad \text{Eq. 6}$$

For example, assume that $F_{CLKIN}$ is 75 MHz ($T_{CLKIN}$ = 13.33 ns) and FINE_SHIFT_RANGE is 10.00 ns(1). In this case, the Dynamic Phase Shift value is limited to ±96.

1. Estimate only. See the Spartan-3 Data Sheet, Module 3 for correct specified value.

When SHIFT_DELAY_RATIO < 2, the dynamic phase shift value is shown by Equation 7. To determine the dynamic phase shift resolution, set Dynamic Phase Shift = 1.

$$T_{PhaseShift} = \left(\frac{DynamicPhaseShift}{\left|DynamicPhaseShift_{LIMITS}\right|}\right) \bullet FINE\_SHIFT\_RANGE \quad \text{Eq. 7}$$

### *SHIFT_DELAY_RATIO ≥ 2*

If the incoming clock period is less than or equal to half the specified FINE_SHIFT_RANGE, then the SHIFT_DELAY_RATIO ≥ 2, meaning that maximum fine phase shift is limited to ±255.

$$DynamicPhaseShift_{LIMITS} = \pm 255 \quad \text{Eq. 8}$$

Consequently, the phase shift value when SHIFT_DELAY_RATIO ≥ 2 is shown by Equation 9. To determine the phase shift resolution, set PHASE_SHIFT = 1.

$$T_{PhaseShift} = \frac{DynamicPhaseShift}{256} \bullet T_{CLKIN} \quad \text{Eq. 9}$$

### Controls

As shown in Figure 33, page 151 and Figure 34, page 152, the DCM's Dynamic Phase Shift control signals allow the FPGA application to adjust the present phase relationship between the CLKIN input and the DCM clock outputs. Table 18 shows the detailed relationship between control inputs, the current and next phase relationship, how the operation affects the delay tap, and the control outputs.

*Table 18:* **Dynamic Phase Shifter Control (assumes no internal inversion)**

| PSEN | PSINC-DEC | PSCLK | Current Phase Shift | Next Phase Shift | Delay Line | PSDONE | STATUS[0] (Overflow) | Operation |
|---|---|---|---|---|---|---|---|---|
| 0 | X | X | X | No change | No change | ? | ? | Dynamic phase shift disabled. |
| 1 | 0 | ↑ | > -Limit | Current – 1 | Current – 1 | 1* | 0 | Decrement phase shift and phase pointer. |
| 1 | 0 | ↑ | ≤ -Limit and > –255 | Current – 1 | No Change | 1* | 1 | End of delay line. No phase shift change. Phase pointer decremented. |
| 1 | 0 | ↑ | –255 | –255 | No Change | 1* | 1 | End of delay line. No phase shift change. Phase pointer at limit. |
| 1 | 1 | ↑ | < +Limit | Current + 1 | Current + 1 | 1* | 0 | Increment phase shift and phase pointer. |
| 1 | 1 | ↑ | ≥ +Limit and < +255 | Current + 1 | No Change | 1* | 1 | End of delay line. No phase shift change. Phase pointer incremented. |
| 1 | 1 | ↑ | +255 | +255 | No Change | 1* | 1 | End of delay line. No phase shift change. Phase pointer at limit. |

**Notes:**
X = don't care.
? = indeterminate, depends on current application state.
1* = PSDONE asserted High for one PSCLK period.
-Limit = minimum delay line position.
+Limit = maximum delay line position.
Assert PSEN for only one PSCLK cycle.

When PSEN is Low, the Dynamic Phase Shifter is disabled and all other inputs are ignored. All present shift values and the delay line position remain unchanged.

If the delay line has not reached its limits (-Limit or –255 when decrementing, +Limit or +255 when incrementing), then the FPGA application can change the existing phase shift value by asserting PSEN High and the appropriate increment/decrement value on PSINCDEC before the next rising edge of PSCLK. The phase shift value increments or decrements as instructed. At the end of the operation, PSDONE goes High for a single PSCLK period indicating that the phase shift operation is complete. STATUS[0] remains Low because no phase shift overflow condition occurred.

When the DCM is incremented beyond +255 or below –255, the delay line position remains unchanged at its limit value of +255 or –255 and no phase change occurs. STATUS[0] goes High, indicating a dynamic phase shift overflow. When a new phase shift operation changes the value in the opposition direction—i.e., away from the limit value—STATUS[0] returns Low.

If the phase shift does not reach +255 or –255, but the phase shift exceeds the delay-line range—indicated by +Limit and –Limit in Table 18—then no phase change occurs. However, STATUS[0] again goes High. The STATUS[0] output indicates when the delay tap reaches the end of the delay line. In the FPGA application, however, use the limit value calculated using either Equation 6 or Equation 8. The calculated delay limit is a guaranteed value. A specific device, due to processing, voltage, or temperature, may have a longer line delay, but this cannot be guaranteed from device to device. The phase shift value—but not the delay line positions—continues to increment or decrement until it reaches its +255 or –255 limit. When a new phase shift operation changes the value in the opposition direction—i.e., away from the limit value—the STATUS[0] signal returns Low. The phase shift value is incremented or decremented back to a value that corresponds to a valid absolute delay in the delay line.

### DCM Wizard

The Dynamic Phase Shift options are part of the DCM Wizard's General Setup panel, shown in Figure 35. To enable dynamic fine phase shifting, select **Variable**, as shown in Figure 35. Enter an initial **Phase Shift Value** in the text box provided. The initial value behaves exactly like the Fixed Fine Phase Shifting mode described above.

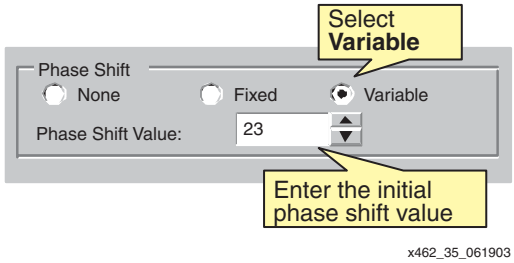



*Figure 35:* **Selecting Dynamic Fine Phase Shift Mode in DCM Wizard**

Choosing **Variable** mode also enables the Dynamic Phase Shift control signals, PSEN, PSINCDEC, PSCLK, and PSDONE, as shown in Figure 36. Check the input and output boxes to use these controls in the FPGA application. Also, check the STATUS output box to enable the STATUS[0] signal. STATUS[0] indicates when the dynamic phase shifter reaches is maximum or minimum limit value.

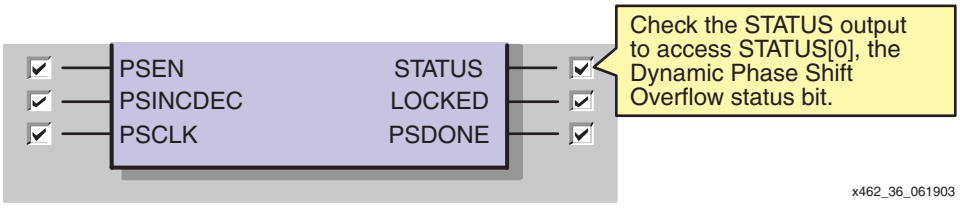



*Figure 36:* **Check the Dynamic Phase Shift Control Outputs in DCM Wizard**

### Example Applications

See application note XAPP268 for an example of how to use the Dynamic Phase Shift function to perform dynamic phase alignment.

- XAPP268: *Dynamic Phase Alignment*
  http://www.xilinx.com/xapp/xapp268.pdf

## Clock Multiplication, Clock Division, and Frequency Synthesis

A DCM provides flexible methods for generating new clock frequencies—one of the most common DCM applications. Spartan-3 DCMs provide up to three independent frequency synthesis functions, listed below, and in Figure 37, and summarized in Table 19. An application may use one or all three functions simultaneously. Detailed descriptions for each function follows.

1. A Clock Doubler (CLK2X, CLK2X180) that doubles the frequency of the input clock.
2. A Clock Divider (CLKDV) that reduces the input frequency by a fixed divider value.
3. A Frequency Synthesizer (CLKFX, CLKFX180) for generating a completely new frequency from an incoming clock frequency.

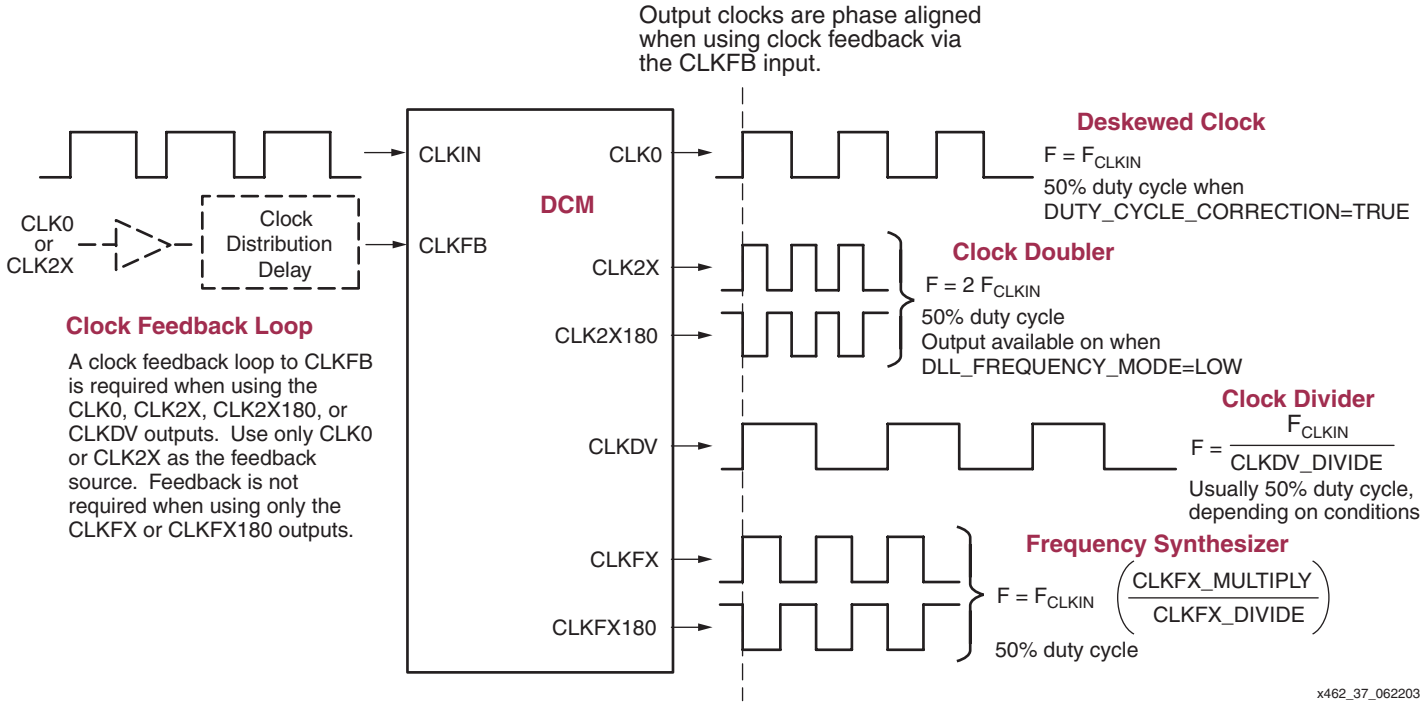



*Figure 37:* **Clock Synthesis Options**

All the frequency synthesis outputs, except CLKDV, always have a 50/50 duty cycle. CLKDV usually has a 50% duty cycle except when dividing by a non-integer value at high frequency, as shown in Table 23. The Clock Doubler (CLK2X, CLK2X180) circuit is not available at high frequencies.

All the DCM clock outputs, except CLKFX and CLKFX180, are generated by the DCM's Delay-Locked Loop (DLL) unit and consequently require some form of clock feedback to the CLKFB pin. The DCM's Digital Frequency Synthesizer (DFS) unit generates the CLKFX and CLKFX180 clock outputs. If the application uses only the CLKFX or CLKFX180 outputs, then the feedback path may be eliminated, which also extends the DCM's operating range. The Frequency Synthesizer has a feedback path within the DCM, based on CLKIN.

*Table 19:* **DCM Frequency Synthesis Options**

| Function | DCM Output(s) | Frequency | DCM Functional Unit | Feedback Required? | 50% Duty Cycle? |
|---|---|---|---|---|---|
| Deskewed Clock | CLK0 | $F_{CLKIN}$ | DLL | Yes | When DUTY_CYCLE_ CORRECTION = TRUE |
| Clock Doubler | CLK2X<br>CLK2X180 | $2 \bullet F_{CLKIN}$ | DLL | Yes | Always |
| Clock Divider | CLKDV | $\frac{F_{CLKIN}}{CLKDV\_DIVIDE}$ | DLL | Yes | Always except when dividing by non-integer value in high-frequency mode |
| Frequency Synthesizer | CLKFX<br>CLKFX180 | $F_{CLKIN} \bullet \left(\frac{CLKFX\_MULTIPLY}{CLKFX\_DIVIDE}\right)$ | DFS | Optional. No feedback extends clock input frequency limits. | Always |

If clock feedback is used, then all the output clocks are phase aligned. Obviously, full clock-edge alignment across all the DCM outputs occurs only occasionally because some of the outputs are divided clock values. For example, the CLKDV output is aligned to CLKIN and CLK0 every CLKDV_DIVIDE cycles. Similarly, the CLK2X output is aligned to CLK0 every other clock cycle. The CLKFX output is aligned to CLKIN every CLKFX_DIVIDE cycles of CLKIN and every CLKFX_MULTIPLY cycles of CLKFX.

## Frequency Synthesis Applications

The potential applications for frequency synthesis are almost boundless. Some example applications include the following.

- Generating a completely new clock frequency for the FPGA and external logic using an available clock frequency on the board.
- Generate a high-frequency internal clock from a slower external clock source to reduce system EMI.
- Dividing a high-speed serial data clock to process data in parallel within the FPGA, as shown in Figure 38.
- Multiplying a parallel data clock before converting to a high-speed serial data format, also shown in Figure 38.
- Multiplying an input clock to overclock internal logic to reduce resources by time-sharing logic when implementing moderately fast functions.

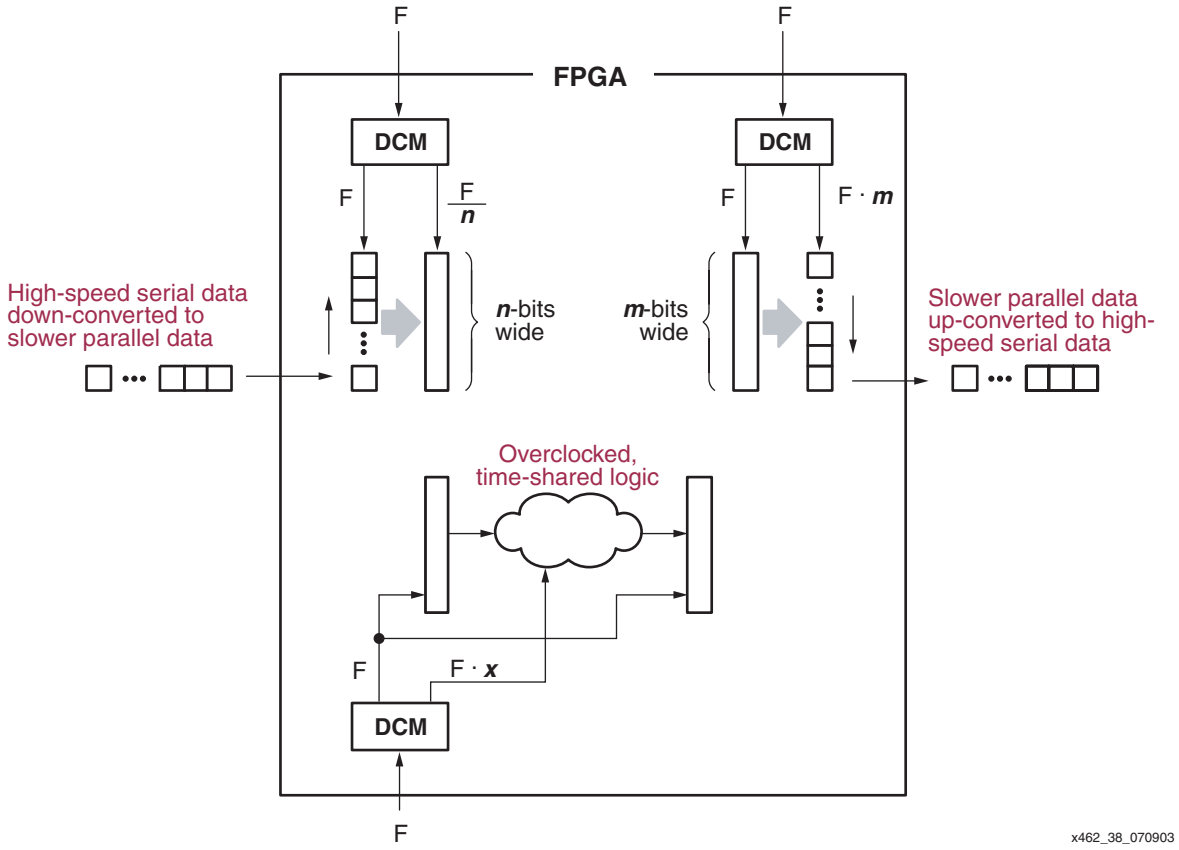



*Figure 38:* **Common Applications of Frequency Synthesis**

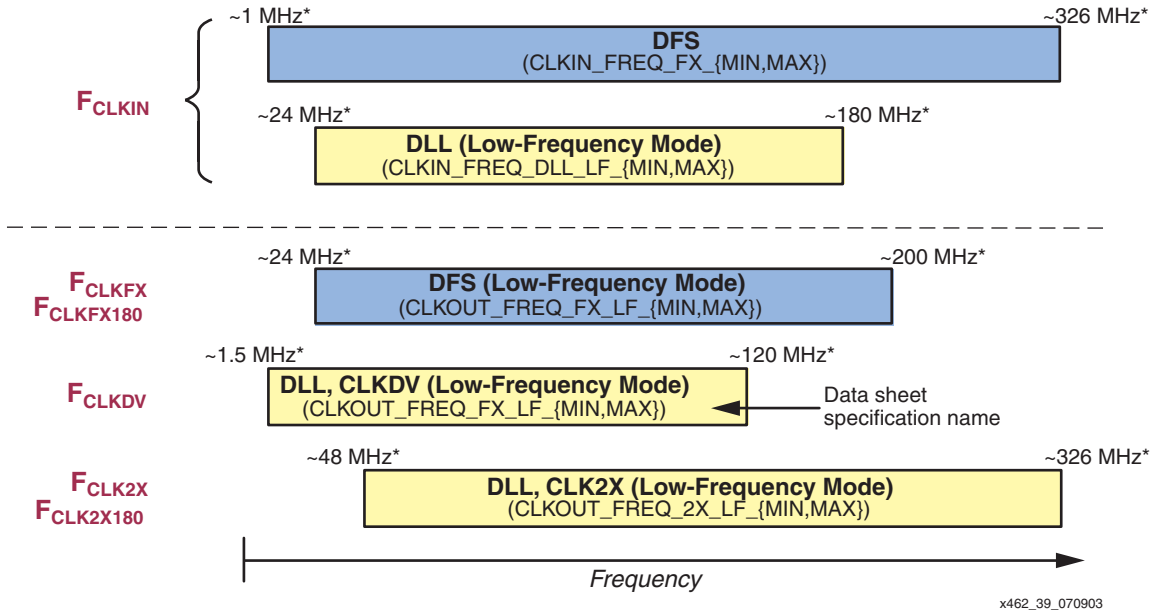
## Input and Output Clock Frequency Restrictions

The input and output clock frequency restrictions for frequency synthesis depend on which DCM clock outputs are used. For example, the CLKFX and CLKFX180 outputs only use the DCM's Digital Frequency Synthesis (DFS) unit. All the other clock outputs use the DCM's Delay-Locked Loop (DLL) unit. The DLL unit has tighter frequency restrictions than the DFS. Consequently, operating the DFS unit without the DLL allows a wider frequency operating range. When using both the DFS and DLL units, the DLL frequency range limits the application.

Also, both the DLL and DFS have a low- and a high-frequency operating mode and the mode settings determine the allowable frequency operating range.

A valid DCM design requires that the CLKIN frequency be within the operating range specified in the Spartan-3 Data Sheet, Module 3. Likewise, the output frequency for any of the clock outputs used must fall within their respective specified operating range.

Figure 39 shows how the various clock input and clock output specifications line up by frequency range. Only the low-frequency operating modes are shown. The data sheet specification for each name is provided within the shaded boxes. Table 20 provides example DCM applications and how the frequency restrictions apply.




* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

*Figure 39:* **Input and Output Clock Frequency Restrictions (Low-Frequency Mode)**

*Table 20:* **DCM Frequency Restriction Examples**

| Input Frequency | Output Frequency | Comments |
|---|---|---|
| 1.2 MHz | 12.8 MHz | Not possible in a single DCM. $F_{CLKIN}$ is within acceptable range for DFS unit, but $F_{CLKFX}$ requires at least a 24MHz output frequency. |
| 1.2 MHz | 32.4 MHz | Possible in a single DCM using DFS unit. Set CLKFX_MULTIPLY=27. $F_{CLKFX}$ is within the DFS output frequency range. |
| 25 MHz | 2.5 MHz<br>30 MHz | Possible in a single DCM using both the DFS and DLL units. Use the CLKDV output for a 2.5MHz signal, setting CLKDV_DIVIDE=10. Use the CLKFX output for a 30MHz signal, setting CLKFX_MULTIPLY=6 and CLKFX_DIVIDE=5. All input and output frequencies are within appropriate ranges. |

## Clock Doubler (CLK2X, CLK2X180)

The Clock Doubler unit doubles the frequency of the incoming CLKIN input, as summarized in Table 21. The Clock Doubler is part of the DLL functional unit and requires a clock feedback path back to CLKFB from either the CLK0 or CLK2X output. The outputs from the Clock Doubler are CLK2X and CLK2X180. Both outputs are always conditioned to a 50% duty cycle. Both have the same output frequency but CLK2X180 is 180° phase shifted from CLK2X, essentially inverting the CLK2X output. Having both phases is essential for high-performance Dual-Data Rate (DDR) or clock forwarding applications.

The CLK2X and CLK2X180 outputs are only available when the DLL_FREQUENCY_MODE attribute is LOW. If required by the application, reduce the CLKIN input frequency using the optional divide-by-two feature (see "Advanced Options").

*Table 21:* **Clock Doubler Summary**

<table>
<tr><td>DCM Output(s)</td><td>CLK2X<br>CLK2X180</td></tr>
<tr><td>Output Frequency</td><td>$2 \bullet F_{CLKIN}$</td></tr>
<tr><td>DCM Functional Unit</td><td>Delay-Locked Loop (DLL)</td></tr>
<tr><td>Feedback Required?</td><td>Yes</td></tr>
<tr><td>50% Duty Cycle?</td><td>Yes</td></tr>
<tr><td colspan="2">Controlling Attributes</td></tr>
<tr><td>DLL_FREQUENCY_MODE</td><td>The CLK2X and CLK2X180 outputs are only valid when DLL_FREQUENCY_MODE = LOW</td></tr>
<tr><td>CLKIN</td><td>The CLKIN frequency limits are determined by the DLL_FREQUENCY_MODE attribute. The Clock Doubler outputs are not available in high-frequency mode but may be required for other DCM clock outputs.
<table>
<tr><th>DLL_ FREQUENCY_ MODE</th><th>Minimum Frequency</th><th>Maximum Frequency</th></tr>
<tr><td>LOW</td><td>CLKIN_FREQ_DLL_LF_MIN (~24 MHz)*</td><td>CLKIN_FREQ_DLL_LF_MAX (~180 MHz)*</td></tr>
<tr><td>HIGH</td><td>CLKIN_FREQ_DLL_HF_MIN (~48 MHz)*</td><td>CLKIN_FREQ_DLL_HF_MAX (~326 MHz)*</td></tr>
</table></td></tr>
<tr><td>CLK2X<br>CLK2X180</td><td>The CLKDV frequency limits are determined by the DLL_FREQUENCY_MODE attribute.
<table>
<tr><th>DLL_ FREQUENCY_ MODE</th><th>Minimum Frequency</th><th>Maximum Frequency</th></tr>
<tr><td>LOW</td><td>CLKOUT_FREQ_2X_LF_MIN (48 MHz)*</td><td>CLKOUT_FREQ_2X_LF_MAX (~325 MHz)*</td></tr>
<tr><td>HIGH</td><td>Not available</td><td>Not available</td></tr>
</table></td></tr>
</table>

* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

## Clock Divider (CLKDV)

The Clock Divider unit, summarized in Table 22, divides the incoming CLKIN frequency by the value specified by the CLKDV_DIVIDE attribute, set at design time. The Clock Divider unit is part of the DLL functional unit and requires a clock feedback path back to CLKFB from either the CLK0 or CLK2X output.

*Table 22:* **Clock Divider Summary**

<table>
<tr><td>DCM Output(s)</td><td>CLKDV</td></tr>
<tr><td>Output Frequency</td><td>$\frac{F_{CLKIN}}{CLKDV\_DIVIDE}$</td></tr>
<tr><td>DCM Functional Unit</td><td>Delay-Locked Loop (DLL)</td></tr>
<tr><td>Feedback Required?</td><td>Yes, using either CLK0 or CLK2X output from DCM</td></tr>
<tr><td>50% Duty Cycle?</td><td>Yes, except when DLL_FREQUENCY_MODE=HIGH and CLKDV_DIVIDE is a non-integer value</td></tr>
<tr><td colspan="2">Controlling Attributes</td></tr>
<tr><td>DLL_FREQUENCY_MODE</td><td>CLKDV is available in both modes. Potentially affects duty cycle of output (see “CLKDV Clock Conditioning”), depending on divider value.</td></tr>
<tr><td>CLKDV_DIVIDE</td><td>Controls the output frequency per the equation above. Legal values include 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 DLL locks faster on integer values than on non-integer values. Likewise, integer values result in lower output jitter.</td></tr>
<tr><td colspan="2">Frequency Constraints</td></tr>
<tr><td>CLKIN</td><td>The CLKIN frequency limits are determined by the DLL_FREQUENCY_MODE attribute.
<table>
<tr><th>DLL_ FREQUENCY_ MODE</th><th>Minimum Frequency</th><th>Maximum Frequency</th></tr>
<tr><td>LOW</td><td>CLKIN_FREQ_DLL_LF_MIN (24 MHz)*</td><td>CLKIN_FREQ_DLL_LF_MAX (~180 MHz)*</td></tr>
<tr><td>HIGH</td><td>CLKIN_FREQ_DLL_HF_MIN (48 MHz)*</td><td>CLKIN_FREQ_DLL_HF_MAX (~325 MHz)*</td></tr>
</table></td></tr>
<tr><td>CLKDV</td><td>The CLKDV frequency limits are determined by the DLL_FREQUENCY_MODE attribute.
<table>
<tr><th>DLL_ FREQUENCY_ MODE</th><th>Minimum Frequency</th><th>Maximum Frequency</th></tr>
<tr><td>LOW</td><td>CLKOUT_FREQ_DV_LF_MIN (1.5 MHz)*</td><td>CLKOUT_FREQ_DV_LF_MAX (~120 MHz)*</td></tr>
<tr><td>HIGH</td><td>CLKOUT_FREQ_DV_HF_MIN (3.0 MHz)*</td><td>CLKOUT_FREQ_DV_HF_MAX (~240 MHz)*</td></tr>
</table></td></tr>
</table>

* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

### CLKDV Clock Conditioning

The CLKDV output is conditioned to a 50% duty cycle unless the DLL_FREQUENCY_MODE attribute is set to HIGH and CLKDV_DIVIDE is a non-integer value. Under these conditions, the CLKDV duty cycle is shown in Table 23. A Spartan-3 DCM requires CLKIN to have at least a 60%/40% (or 40%/60%) or better duty cycle. Consequently, the CLKDV output, divided by 1.5 in high-frequency mode cannot provide a clock input to a second cascaded DCM.

*Table 23:* **CLKDV Duty Cycle with DLL_FREQUENCY_MODE=HIGH**

| CLKDV_DIVIDE Attribute | Duty Cycle | High Time/ Total Cycle |
|---|---|---|
| Integer | 50.000% | 1/2 |
| 1.5 | 33.333% | 1/3 |
| 2.5 | 40.000% | 2/5 |
| 3.5 | 42.857% | 3/7 |
| 4.5 | 44.444% | 4/9 |
| 5.5 | 45.454% | 5/11 |
| 6.5 | 46.154% | 6/13 |
| 7.5 | 46.667% | 7/15 |

### CLKDV Jitter Depends on Frequency Mode and Integer or Non-Integer Value

Similarly, integer values for the CLKDV_DIVIDE attribute result in half the output jitter and faster DLL locking times.

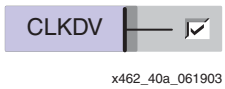
*Table 24:* **CLKDV Output Jitter**

| CLKDV_DIVIDE | CLKDV Output Period Jitter |
|---|---|
| Integer Value | CLKOUT_PER_JITT_DV1 (±~150 ps)* |
| Non-integer Value | CLKOUT_PER_JITT_DV2 (±~300 ps)* |

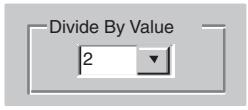
* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

### DCM Wizard

The Clock Divider controls are in DCM Wizard's General Setup window. Check the CLKDV output box, shown in Figure 40a. Then, choose the Clock Divider's **Divide by Value** using the drop-down list, shown in Figure 40b.




**a. Check the CLKDV output box**




**b. Select the Divide by Value from the Drop-Down List**

*Figure 40:* **Specifying the Clock Divider in DCM Wizard**

## Frequency Synthesizer (CLKFX, CLKFX180)

The Frequency Synthesizer provides the most flexible means to multiply, divide, or multiply and divide an input frequency. As shown in Table 25, the two Frequency Synthesizer outputs are CLKFX and CLKFX180. The CLKFX180 output has the same frequency as CLKFX but is phase shifted 180°, or half a clock period. Because both Frequency Synthesizer outputs have 50% duty cycles, CLKFX180 appears to be an inverted version of CLKFX.

Two attributes, set at design time, control the synthesized output frequency, as shown in the equation in Table 25. The CLKIN clock input is multiplied the fraction formed by CLKFX_MULTIPLY as the numerator and CLKFX_DIVIDE as the denominator. For example, to create a 155MHz output using a 75MHz CLKIN input, the Frequency Synthesizer multiplies CLKIN by the fraction 31/15. Note that it does not multiply CLKIN by 31 first, then divide by the result by 15. Multiplying CLKIN by 31 would result in a 2.325GHz output frequency—well outside the frequency range of the Spartan-3 DCM.

The multiplier and divider values should be reduced to their simplest form, which results in faster lock times. For example, reduce the fraction 6/8 to 3/4.

Frequency synthesis always requires some form of clock feedback. However, the DFS unit has an internal feedback loop based on CLKIN and does not require a separate loop on CLKFB if used without the DLL unit.

The CLKFX output is phase aligned with the CLKIN input every CLKFX_DIVIDE cycles of CLKIN and every CLKFX_MULTIPLY cycles of CLKFX. For example, if CLKFX_MULTIPLY = 3 and CLKFX_DIVIDE = 5, then the CLKFX output is phase aligned with the CLKIN input every five CLKIN cycles and every three CLKFX cycles. After the DCM asserts its LOCKED output, the DFS unit is resynchronized to the CLKIN input at each concurrence and phase alignment is nearly perfect at these edges.

*Table 25:* **Frequency Synthesizer Summary**

| | |
|---|---|
| DCM Output(s) | CLKFX<br>CLKFX180 (same as CLKFX, phase shifted 180°) |
| Output Frequency | $F_{CLKIN} \bullet \frac{CLKFX\_MULTIPLY}{CLKFX\_DIVIDE}$ |
| DCM Functional Unit | Digital Frequency Synthesizer (DFS) |
| Feedback Required? | No. Uses internal feedback based on CLKIN. Optionally can use CLKFB input if required for Delay-Locked Loop (DLL) functions. |
| 50% Duty Cycle? | Yes, always. |
| **Controlling Attributes** | |
| DFS_FREQUENCY_MODE | Affects frequency limits on CLKIN and the CLKFX, CLKFX180 outputs. |
| DLL_FREQUENCY_MODE | Only affects the Frequency Synthesizer if the application uses any DLL outputs. Potentially reduces the CLKIN frequency to the more restrictive DLL limits. If only the CLKFX or CLKFX180 outputs are used, then DFS_FREQUENCY_MODE alone defines the frequency limits. |
| CLKFX_MULTIPLY | Controls the output frequency per the equation above. Legal values include integer values ranging from 2 to 32. Default value is 4. |
| CLKFX_DIVIDE | Controls the output frequency per the equation above. Legal values include integer values ranging from 1 to 32. Default value is 1. |

*Table 25:* **Frequency Synthesizer Summary *(Continued)***

<table>
<tr><th colspan="2">Frequency Constraints</th></tr>
<tr><td>CLKIN</td><td>The CLKIN frequency limits are determined by the DFS_FREQUENCY_MODE attribute unless the application uses any outputs from the Delay-Locked Loop (DLL) unit. If the DLL unit is used, then the more restrictive DLL clock limits apply.
<table>
<tr><th>Minimum Frequency</th><th>Maximum Frequency</th></tr>
<tr><td>CLKIN_FREQ_FX_MIN (~1.0 MHz)*</td><td>CLKIN_FREQ_FX_MAX (~326 MHz)*</td></tr>
</table></td></tr>
<tr><td>CLKFX<br>CLKFX180</td><td>The CLKFX and CLKFX180 output frequency limits are determined by the DFS_FREQUENCY_MODE attribute.
<table>
<tr><th>DFS_ FREQUENCY_ MODE</th><th>Minimum Frequency</th><th>Maximum Frequency</th></tr>
<tr><td>LOW</td><td>CLKIN_FREQ_FX_LF_MIN (~24 MHz)*</td><td>CLKIN_FREQ_FX_LF_MAX (~210 MHz)*</td></tr>
<tr><td>HIGH</td><td>CLKIN_FREQ_FX_HF_MIN (~210 MHz)*</td><td>CLKIN_FREQ_FX_HF_MAX (~325 MHz)*</td></tr>
</table></td></tr>
</table>

* Estimate only. See the Spartan-3 Data Sheet, Module 3 for the correct specified value.

### DCM Wizard

To enable the Frequency Synthesizer in DCM Wizard, check the CLKFX, CLKFX180, or both clock outputs in the General Setup window, as shown in Figure 41.

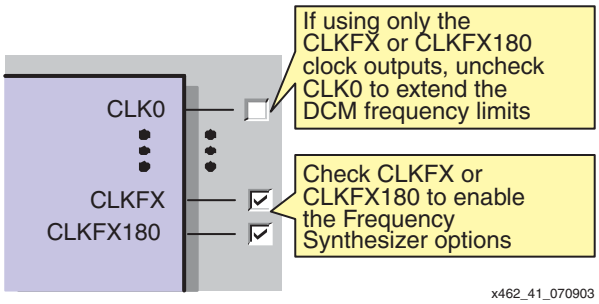



*Figure 41:* **Enabling Frequency Synthesizer in DCM Wizard**

If using the CLKFX or CLKFX180 clock outputs stand-alone, then optionally extend the frequency limits by disabling any DLL clock outputs and any feedback.

- By default, the CLK0 output is always checked. If just using CLKFX or CLKFX180, uncheck CLK0.
- Disable DCM feedback by selecting **None**, as shown in Figure 42. Without feedback, the CLKFX and CLKFX180 frequency range is extended to both lower and higher frequencies.

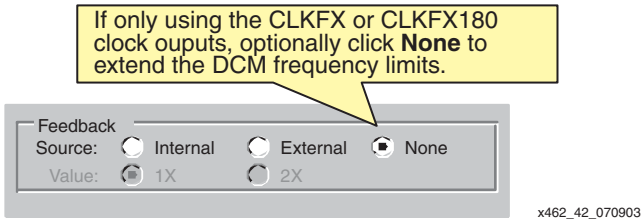



*Figure 42:* **Select No Feedback (None) to Extend Frequency Synthesizer Frequency Limits**

Finally, enter the desired output frequency or the Multiply and Divide values, as described in the DCM Wizard Clock Frequency Synthesizer panel section.

## Clock Forwarding, Mirroring, Rebuffering

Because DCMs provide advanced clock control features and Spartan-3 I/O pins support a variety of I/O voltage standards, Spartan-3 FPGAs commonly are used to rebuffer or mirror clock signals, often changing the input clock from one voltage standard to another. Likewise, the DCM conditions an incoming clock signal so that it has a 50% duty cycle.

Figure 20 shows a simple example where a DCM conditions an incoming clock to a 50% duty cycle, and then either forwards the clock at the same frequency using the CLK0 output, or doubles the frequency using the DCM CLK2X output. Similarly, the input and output clocks are phase aligned once the DCM asserts its LOCKED output. The clock feedback path to CLKFB monitors and eliminates the clock distribution delay at the external clock feedback point.

If a 50/50 duty cycle is important on the output clock, make sure that the output I/O standard can switch fast enough to preserve the 50% duty cycle. Verify the duty cycle performance using IBIS simulation on the output signal. Some I/O standards have asymmetric rise and fall times that distort the duty cycle higher frequencies, as can be seen in the IBIS simulation. The DCI versions of HSTL, SSTL, and LVCMOS I/O standards have better symmetry.

To guarantee a 50/50 duty cycle above 100 MHz, the DCM's duty cycle correction capability is mandatory, even if the CLKIN source provides a clean 50% duty cycle. Consequently, the DUTY_CYCLE_CORRECTION attribute must equal TRUE when using the CLK0, CLK90, CLK180, or CLK270 outputs for clock forwarding. The other DCM clock outputs are normally always clock corrected to a 50% duty cycle (see "Clock Conditioning").

For best duty-cycle performance—especially at 200 MHz and greater—use a circuit similar to that shown in Figure 43. Use both the CLKx and CLKx180 outputs from the DCM to drive the C0 and C1 inputs, respectively, on a Dual-Data Rate (DDR) output flip-flop. Connect the D0 input of the DDR flip-flop to $V_{CC}$ and the D1 input to GND. Each DCM output drives a separate global buffer, which minimizes duty-cycle distortion. At higher frequencies, it is best not to distribute just one clock and invert one phase locally within the DDR flip-flop, as this adds approximately 150 ps of duty-cycle distortion.

At frequencies of 250 MHz or higher, distribute clocks using a differential signaling standard, such as LVDS. In Figure 43, for example, both the CLKIN clock input and the clock output use LVDS. Additionally, the clock feedback path uses LVDS. For optimal performance, both the clock input and the clock feedback paths require differential global buffer inputs (IBUFGDS), which unfortunately consumes all the global buffer inputs along one edge of the device. However, this solution provides the best-quality clock forwarding solution at high frequencies.

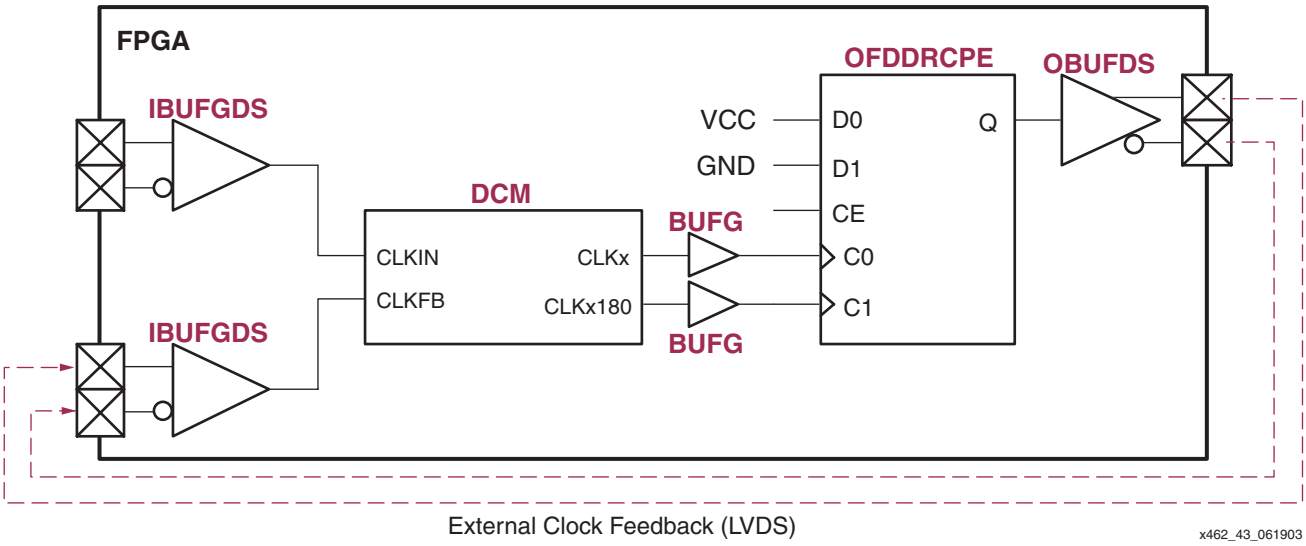



*Figure 43:* **High-Frequency (250+ MHz) LVDS Clock Forwarding Circuit with 50% Duty Cycle**

## Clock Jitter or Phase Noise

All clocks, including the most expensive, high-precision sources, exhibit some amount of clock jitter or phase noise. The Spartan-3 Digital Clock Managers have their own jitter characteristics, as described in this section. When operating at low frequencies—20 MHz, for example—the effects of jitter usually can be ignored. However, when operating at high frequencies—200 MHz, for example, especially in dual-data rate (DDR) applications—clock jitter becomes a relevant design factor. Clock jitter directly subtracts from the time available to the FPGA application by effectively reducing the available time between active clock edges.

### What is Clock Jitter?

Clock jitter is the variation of a clock edge from its ideal position in time, as illustrated in Figure 44. The heavy line shows the ideal position on the clock signal. On each clock edge, there is some amount of variation between the actual clock edge and its ideal location. The difference between the maximum and minimum variations is called peak-to-peak jitter. Jitter is only relevant on the active clock edge. For example, in single-data rate (SDR) applications, data is clocked at each rising clock edge and the specified jitter only subtracts from the total clock period. In dual-data rate (DDR) application, data is clocked at the start of each period and halfway into the period. Therefore, jitter affects each half period.

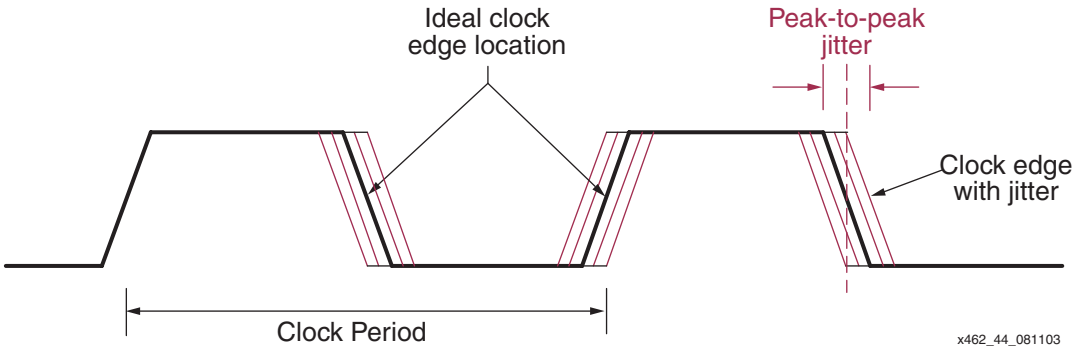



*Figure 44:* **Jitter in Clock Signals**

## What Causes Clock Jitter?

Clock jitter is unavoidable and exists in all systems. Clock jitter is caused by the various sources of noise or by signal imperfections within the system. In fact, jitter is the manifestation of noise in the time domain. The incoming clock source, for example, has its own jitter characteristics due to random thermal or mechanical vibration noise from the crystal. A large number of simultaneous switching outputs (SSOs) adds substrate noise that slightly changes internal switching thresholds and therefore adds jitter. Similarly, an improperly designed power supply or insufficient decoupling also contributes to jitter. Other sources of clock jitter include cross talk from adjacent signals, poor termination, ground bounce, and electromagnetic interference (EMI).

This application note only discusses the jitter behavior of Spartan-3 Digital Clock Managers (DCMs) and how to improve overall jitter performance within the FPGA.

## Understanding Clock Jitter Specifications

Clock jitter is specified in a variety of manners, and the various specifications show different aspects of the same phenomenon.

### Cycle-to-Cycle Jitter

Cycle-to-cycle jitter, also called adjacent cycle jitter, indicates the maximum clock period variance from one clock cycle to the next, as shown in Figure 45. In this simple example, the maximum change from one cycle to the next is +100 ps and –100 ps, or put simply, ±100 ps. Although the clock period may change by larger absolute amounts when measured over millions of clock cycles, the clock period never changes by more than ±100 ps from one clock cycle to the next.

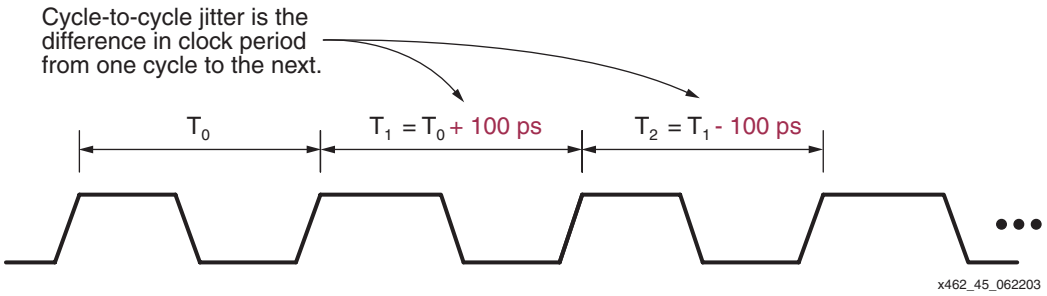



*Figure 45:* **Cycle-to-Cycle Jitter Example**

Cycle-to-cycle is an important measure of the quality of a clock output or oscillator but has little use in analyzing the timing of an application.

### Period Jitter

Period jitter is the summation of all the cycle-to-cycle jitter values over millions of clock cycles. Peak jitter indicates the earliest and the latest transition times compared to the ideal clock transition time over consecutive clocks.

Period jitter for Digital Clock Mangers is random and is expressed as peak-to-peak jitter. Conceptually, the position of the clock transition is a probabilistic distribution or histogram, centered around the ideal, desired clock position, as shown in Figure 46. The actual distribution may not appear purely Gaussian and may be bimodal. Regardless, most actual clock transitions occur near the desired ideal position. However, measured over millions of clock cycles, some clock transitions occur far from the desired position.

The statistical distance from the desired position is measured in standard deviations, also called σ (sigma). Because the DCM is an all-digital design, it is highly stable and Xilinx specifies jitter deviation to ±7σ or peak-to-peak jitter to 14σ. As a point of reference, ±7σ guarantees that

99.99999999974% of the jitter values are less than the specified worst-case jitter value. A 14σ peak-to-peak jitter, ±7σ jitter deviation, equates to a maximum bit error rate (BER) of $1.28 \times 10^{-12}$.

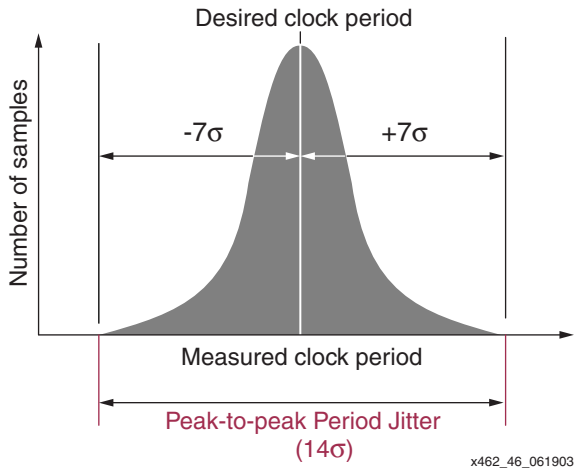



*Figure 46:* **Peak-to-Peak Period Jitter Example**

### Unit Interval (UI)

Another method to specify jitter is as a fraction of the Unit Interval (UI). One UI represents the time equivalent to one bit time, irrespective of frequency. In single-data rate (SDR) applications where either the rising or the falling clock edge captures data, one UI equals one clock period. In dual-data rate (DDR) applications where data is clocked at twice the clock rate, one UI equals half the clock period.

The peak-to-peak jitter amplitude, quantified in UIs, is the fraction of the peak-to-peak jitter value compared to the total bit period time.

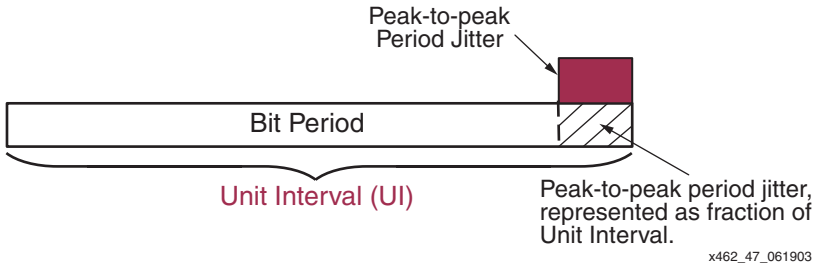



*Figure 47:* **Period Jitter Specified as a Fraction of a Unit Interval**

## Calculating Total Jitter

The Spartan-3 Data Sheet specifies the output jitter from the DCM clock outputs, except for CLKFX/CLKFX180. The Digital Frequency Synthesizer (DFS) jitter is calculated based on the multiplier and divider settings.

The clock outputs from the DLL unit—i.e., every clock output except CLKFX and CLKFX180—have a worst-case specified jitter listed in the data sheet. This specified value includes the jitter added by the DLL unit. The DLL unit does not remove jitter, so the total jitter on the DLL clock output includes the jitter on the input clock, CLKIN, plus the specified value from the data sheet.

The DFS clock outputs, CLKFX and CLKFX180, remove some amount of incoming clock jitter, so the calculated output jitter is the total jitter.

### Adding Input Jitter to DLL Output Jitter

When adding the input jitter and the DLL output jitter, use a root-mean-square (RMS) calculation, similar to noise calculations.

#### *Peak-to-Peak*

$$JITTER_{TOTAL} = \sqrt{(JITTER_{INPUT})^2 + (JITTER_{SPEC})^2} \quad \text{Eq. 10}$$

#### *Peak-to-Peak Deviation*

$$JITTER_{TOTAL} = \pm\left[\sqrt{\frac{(JITTER_{INPUT})^2 + (JITTER_{SPEC})^2}{2}}\right] \quad \text{Eq. 11}$$

where

- $JITTER_{INPUT}$ = The input period jitter, measured at the clock input pin of the FPGA
- $JITTER_{SPEC}$ = The DLL clock output period jitter, as specified in the Spartan-3 Data Sheet for the appropriate output port
- $JITTER_{TOTAL}$ = The expected total output period jitter

#### *Example*

Assume that an input clock has 150 ps peak-to-peak period jitter, optionally expressed as ±75 ps. The incoming clock is duty-cycle corrected, using the same frequency, on the CLK0 DCM output.

In this case, $JITTER_{INPUT}$ = 150 ps. The value for $JITTER_{SPEC}$ is the Spartan-3 Data Sheet specification called CLKOUT_JITT_PER_0, which is estimated here as ±100 ps, or 200 ps peak-to-peak.

$$JITTER_{TOTAL} = \sqrt{(150\ \text{ps})^2 + (200\ \text{ps})^2} = 250\ \text{ps}$$

Consequently, the total jitter on the DCM output is 250 ps peak-to-peak or ±125 ps.

## Calculating Jitter for Cascaded DCMs

Figure 48 shows an example application where multiple DCMs are cascaded together to create various output frequencies. The jitter at any point depends on:

- the incoming jitter from the previous sources and
- which DCM output is used.

Each DCM output has slightly different jitter characteristics, as specified in the data sheet. Also, the CLKFX and CLKFX180 outputs from the DFS unit remove some amount of input jitter and DCM Wizard calculates their jitter values (see "Clock Frequency Synthesizer").

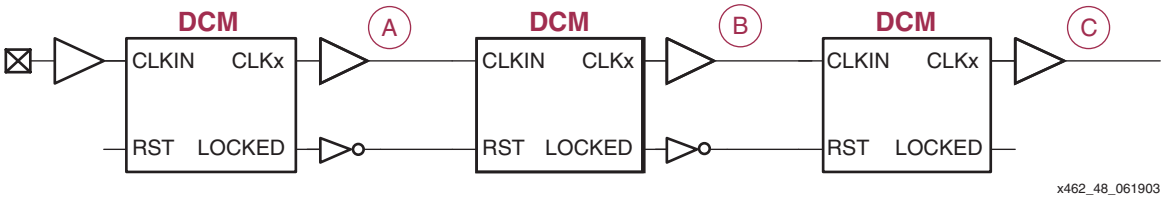



*Figure 48:* **Calculating Jitter for Cascaded DCMs Depends on which DCM Outputs are Used**

Consequently, the jitter at any point in the cascaded DCM chain depends on the factors described above. The following examples illustrate how to calculate total jitter at the various points in the circuit.

**Example 1: All DCMs Use DLL Outputs**

In this example, assume that the input clock has 150 ps (±75 ps) of period jitter.

Assume that DCM (A) uses the CLK2X output. Use the Spartan-3 Data Sheet specification called CLKOUT_PER_JITT_2X for the DCM output jitter, estimated here as 400 ps (±200 ps). Calculate the total period jitter on clock (A) using Equation 10.

$$JITTER_{TOTAL(A)} = \sqrt{(150\text{ps})^2 + (400\text{ps})^2} = 427\text{ps} = \pm 214\text{ps}$$

Assume that DCM (B) uses the CLKDV output with an integer divider value. Use the Spartan-3 Data Sheet specification called CLKOUT_PER_JITT_DV1 for the DCM output jitter, estimated here as 300 ps (±150 ps). Calculate the total period jitter on clock (B) using Equation 10. Because there are now three elements involved—the input jitter, the jitter from DCM (A), and the jitter from DCM (B)—expand the RMS equation appropriately.

$$JITTER_{TOTAL(B)} = \sqrt{(150\text{ps})^2 + (400\text{ps})^2 + (300\text{ps})^2} = 522\text{ps} = \pm 261\text{ps}$$

Finally, assume that DCM (C) phase shifts the output from DCM (B) by 90°. Use the Spartan-3 Data Sheet specification called CLKOUT_PER_JITT_90 for the DCM output jitter, estimated here as 300 ps (±150 ps). Calculate the total period jitter on clock (C) using Equation 10. Because there are now four elements involved—the input jitter, the jitter from DCM (A), the jitter from DCM (B), and the jitter from DCM (C)—expand the RMS equation appropriately.

$$JITTER_{TOTAL(C)} = \sqrt{(150ps)^2 + (400ps)^2 + (300ps)^2 + (300ps)^2} = 602ps = \pm 301ps$$

**Example 2: Some DCMs Use the CLKFX or CLKFX180 Outputs**

This example is similar to Example 1: All DCMs Use DLL Outputs above except that some DCMs use the CLKFX or CLKFX180 outputs from the DCM's DFS unit.

In this example, assume that the 75MHz input clock has 150 ps (±75 ps) of period jitter.

As in Example 1, assume again that DCM (A) uses the CLK2X output. The resulting output jitter is the same as that shown in the following equation.

In this example, assume that DCM (B) synthesizes a 90MHz clock using the 150MHz clock generated by DCM (A). Per the DCM Wizard (see "Clock Frequency Synthesizer"), set the attributes CLKFX_MULTIPLY=3 and CLKFX_DIVIDE=5. DCM Wizard also specifies the worst-case output period jitter as 700 ps.

$$JITTER_{TOTAL(B)} = 700\text{ps} = \pm 350\text{ps}$$

Finally, assume again that DCM (C) phase shifts the output from DCM (B) by 90°. Use the Spartan-3 Data Sheet specification called CLKOUT_PER_JITT_90 for the DCM output jitter,

estimated here as 300 ps (±150 ps). Calculate the total period jitter on clock (C) using the following equation. Because the preceding DCM used the CLKFX output, the total incoming jitter is set at 700 ps, worst-case. Use the RMS equation to calculate the resulting output jitter as shown below.

$$JITTER_{TOTAL(C)} = \sqrt{(700\text{ps})^2 + (300\text{ps})^2} = 762\text{ps} = \pm 381\text{ps}$$

### Cascaded DCM Design Recommendations

When cascading DCMs, be sure that the LOCKED output of the preceding DCM controls the cascaded DCM's RST input, as shown in Figure 48. The cascaded DCM should not attempt to lock to the input clock until the preceding DCM asserts its LOCKED output, indicating that the clock is stable.

When cascading DCMs, place the most jitter-critical clock output on the first DCM in the cascaded chain.

## Jitter Effect on System Performance

Clock jitter, along with other effects, adversely affects system performance by reducing the effective bit period. The bit period available to the FPGA application is the total bit period, $T_{BIT}$, minus the following effects, as shown in the following equation. In single-data rate (SDR) applications, the clock period and the bit period are equal. However, in dual-data-rate (DDR) applications, the bit period is half the clock period.

$$T_{AVAILABLE} = T_{BIT} - t_{TOTAL\_JITTER} - t_{DUTY\_CYCLE\_DISTORTION}$$

where

$T_{BIT}$ = Bit period time

$t_{TOTAL\_JITTER}$ = Total clock jitter. Includes the clock input jitter plus any DCM output jitter or cascaded DCM output jitter.

$t_{DUTY\_CYCLE\_DISTORTION}$ = Duty cycle distortion specification. Only required for dual-data rate (DDR) applications; otherwise zero. Either data sheet specification CLKOUT_DUTY_CYCLE_DLL or CLKOUT_DUTY_CYCLE_FX depending on which DCM clock output is used.

If the total jitter is specified as a positive value instead of a deviation from the clock period—e.g., 200 ps instead of ±100 ps—subtract half the positive value—i.e., 100 ps. The bit period is only shortened by the negative deviation. The positive deviate adds to the bit period, adding more timing slack.

### Example

Assume that an incoming clock signal enters the FPGA at 75 MHz and that the clock source has ±100 ps of jitter. The application clocks data on the rising edge of an internally generated 150 MHz, or a total bit period, $T_{BIT}$, of 6.67 ns. How long is the available bit period, $T_{AVAILABLE}$, after considering the effects of jitter?

The CLK2X output from the Clock Doubler generates a 150MHz clock from the 75MHz clock input. The Clock Doubler output, CLK2X, has ±~200 ps (estimated) of worst-case jitter according to the CLKOUT_PER_JITT_2X specification in the Spartan-3 Data Sheet. Adding

the DCM's ±200 ps of jitter to the clock source's ±100 ps of jitter using root-mean square (RMS), the total jitter, $t_{TOTAL\_JITTER}$, is ±0.223 ns.

$$t_{TOTAL\_JITTER} = \sqrt{(\pm 100\text{ps})^2 + (\pm 200\text{ps})^2} = \pm 223.60\text{ps} = \pm 0.223\text{ns}$$

Because data is only clocked on the rising clock edge, there are no duty-cycle distortion effects and $t_{DUTY\_CYCLE\_DISTORTION} = 0$.

Therefore, the total available clock period, $T_{AVAILABLE}$ is reduced down to 6.444 ns from a total bit period of 6.667 ns. Effectively, this forces the logic to operate at 155.1831 MHz instead of 150 MHz.

$$T_{AVAILABLE} = 6.667\text{ns} - 0.223\text{ns} = 6.444\text{ns}$$

## Recommended Design Practices to Minimize Clock Jitter

In higher-performance applications, clock jitter steals valuable bit period time. Adhere to the following recommendations to minimize the amount of system-wide clock jitter.

### Properly Design the Power Distribution System

A properly designed power distribution system (PDS), including proper power-plane decoupling, reduces system jitter by creating a stable power environment. Application note XAPP623 discusses recommended design practices for PDS design.

- XAPP623: *Power Distribution System (PDS) Design: Using Bypass/Decoupling Capacitors*
  http://www.xilinx.com/xapp/xapp623.pdf

### Properly Design the Printed Circuit Board

Design the printed circuit board for expected operating frequency range and application environment.

- WP174: *Methodologies for Efficient FPGA Integration into PCBs*
  http://www.xilinx.com/publications/whitepapers/wp_pdf/wp174.pdf
- PCB Checklist
  http://support.xilinx.com/xlnx/xil_prodcat_product.jsp?title=si_pcbcheck

### Obey Simultaneous Switching Output (SSO) Recommendations

To avoid signal-related corruption of clock inputs to or clock outputs from a DCM, be sure to follow the Simultaneous Switching Output (SSO) recommendations outlined in the Spartan-3 Data Sheet.

Whenever possible, avoid placing DCM inputs or output near heavily switching I/Os, especially those with large output voltage swings or with high current drive.

### Place Virtual Ground Pins Around DCM Input and Output Connections

On sensitive, high frequency DCM inputs or outputs, use additional user-I/O pin to create extra connections to the PCB ground—i.e., create virtual ground pins. Place these virtual ground pins on the I/O pads adjacent to the sensitive DCM signal. Make sure that the I/O pads are on adjacent pads on the FPGA die level, not just on adjacent pins or balls on the package. Adjacent balls on BGA packages do not necessarily connect to adjacent pads on the FPGA. These techniques reduce the internal voltage drop and improve the jitter.

To create a "virtual ground", configure an IOB as an output driving GND (Low logic level) and connect the IOB externally directly to the ground plane, as shown in Figure 49.

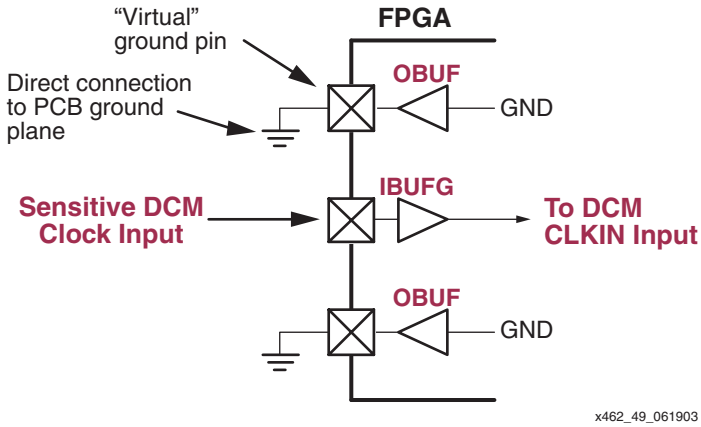



*Figure 49:* **Place Virtual Ground Pins Adjacent to Sensitive DCM Input or Output Clock Signals**

### $V_{CCAUX}$ Considerations for Improving Jitter Performance

The Digital Clock Managers are powered by the $V_{CCAUX}$ supply input. Any excessive noise on the $V_{CCAUX}$ supply input to the FPGA adversely affects the DCM's characteristics, especially its jitter performance. For best DCM performance, please follow these recommendations.

1. Limit changes in on the $V_{CCAUX}$ power supply or ground potentials to less than 10 mV in any 1 ms interval, as shown in Figure 50. This recommendation allows the DCM to properly track out the change.
2. Limit the noise at the power supply to be within 200 mV peak-to-peak, as shown in Figure 50.

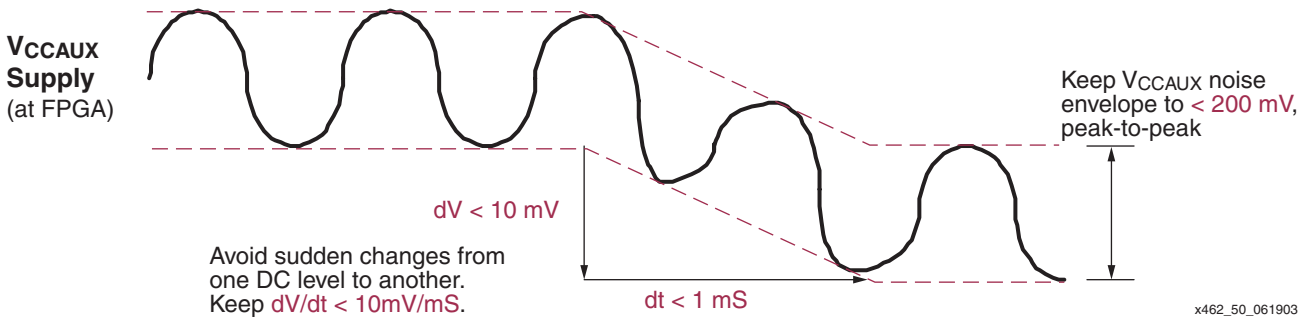



*Figure 50:* **Recommended $V_{CCAUX}$ Supply Considerations Avoid Voltage Droop**

3. If $V_{CCAUX}$ and $V_{CCO}$ are of the same power plane, every $V_{CCAUX}$/$V_{CCO}$ pin must be properly decoupled or bypassed (see "Properly Design the Power Distribution System"). Separate the $V_{CCAUX}$ supply from any $V_{CCO}$ supplies if Guidelines 1 and 2 above cannot be maintained.
4. The CLK2X output is especially affected by the power or ground shift. Consequently, the CLKFX output, using CLKFX_MULTIPLY =2 and CLKFX_DIVIDE=1, may provide a better quality output when all IOBs and CLBs are switching. The CLKFX circuitry updates the tap every three input clocks in the DFS mode, as opposed to the slower update rate for the CLK2X output.

### Adjusting FACTORY_JF Setting

A well-designed, stable, properly decoupled power supply is the best overall solution to reducing clock skew and jitter within the FPGA. However, increasing the FACTORY_JF attribute setting to 0xFFFF may improve jitter performance on a problem board. When

FACTORY_JF=FFFF, the DCM updates its tap settings approximately every twenty input clocks. The frequency-based default settings update the tap settings much more slowly.

Increasing the FACTORY_JF setting may introduce a small amount of jitter (~30 ps) because the DCM frequently updates its delay line, which is why FACTORY_JF is not set to the maximum value by default. If the power supply is unstable, the phase error introduced may be much bigger than the extra jitter introduced; therefore, increasing the FACTORY_JF setting may improve the design.

# Miscellaneous Topics

## Bitstream Generation Settings

There are two bitstream generation (BitGen) options related to the DCM:

- **-g lck_cycle**: This option causes the FPGA configuration startup sequence to wait until all instantiated DCMs assert their LOCKED outputs.
- **-g DCMShutdown**: This option resets the DCM logic if the "SHUTDOWN" configuration command is loaded into the configuration logic, as during either partial reconfiguration or during full reconfiguration via the JTAG port.

### Setting Bitstream Generation Options in Project Navigator

If using the ISE 5.2i Project Navigator graphical interface, set the bitstream generation options by right-mouse clicking **Generate Programming File** in the Processes for Current Source panel, as shown in Figure 51. Select **Properties** from the resulting menu.

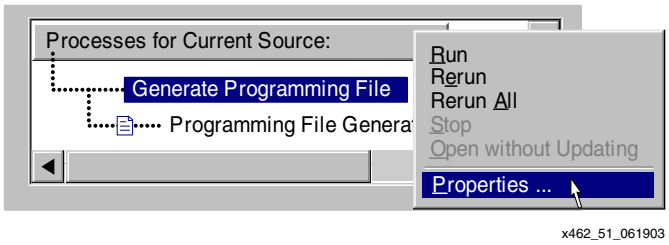



*Figure 51:* **Setting Bitstream Generator (BitGen) Options within Project Navigator**

See "BitGen Switches and Options" for more information.

### Setting Bitstream Generation Options via Command Line or Script

To see the available options, type the following in a command window:

```
bitgen -help spartan3
```

### Setting Configuration Logic to Wait for DCM LOCKED Output

The DCM's STARTUP_WAIT attribute signals the FPGA's configuration start-up logic to wait for the DCM to assert its LOCKED output before the FPGA asserts its DONE output. Two actions are required at design time, however. First, set the STARTUP_WAIT attribute to TRUE on each of the DCMs that must be locked before configuration completes. Then, modify the bitstream generation options so that the events shown in Figure 52 happen within the six-clock start-up cycle. Sufficient configuration clock cycles must be provided after the DCM locks to allow the device to complete the configuration start-up sequence.

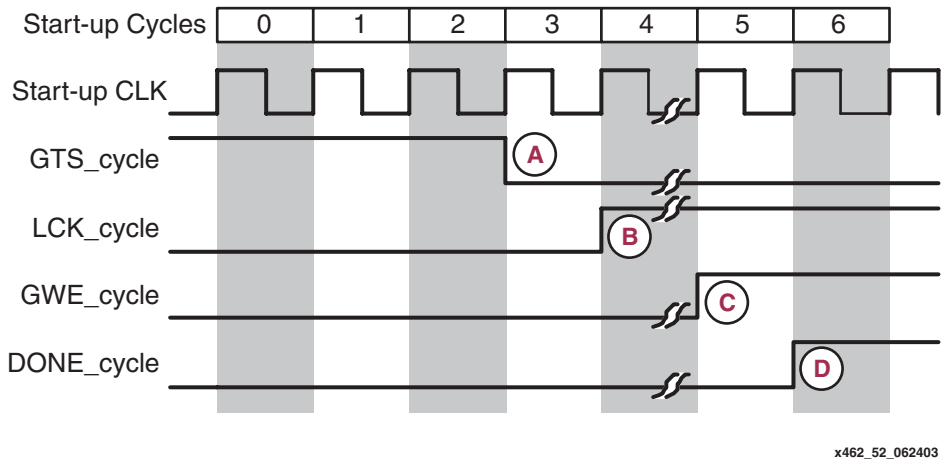



*Figure 52:* **Start-up Logic Interaction with DCM LOCKED Output**

A. Release the FPGA's internal Global Three-State (GTS_cycle) signal, enabling all I/O signals.

B. Set the cycle where the start-up logic waits for the DCM(s) to assert LOCKED after the GTS_cycle. The DCMs require some form of external input—a clock and possibly a feedback signal—before the DCM can lock on the clock signal.

C. After achieving valid DCM lock, assert the FPGA's internal Global Write Enable (GWE_cycle) signal.

D. Finally, assert the internal DONE signal.

Figure 53 shows these same option settings from within Project Navigator.

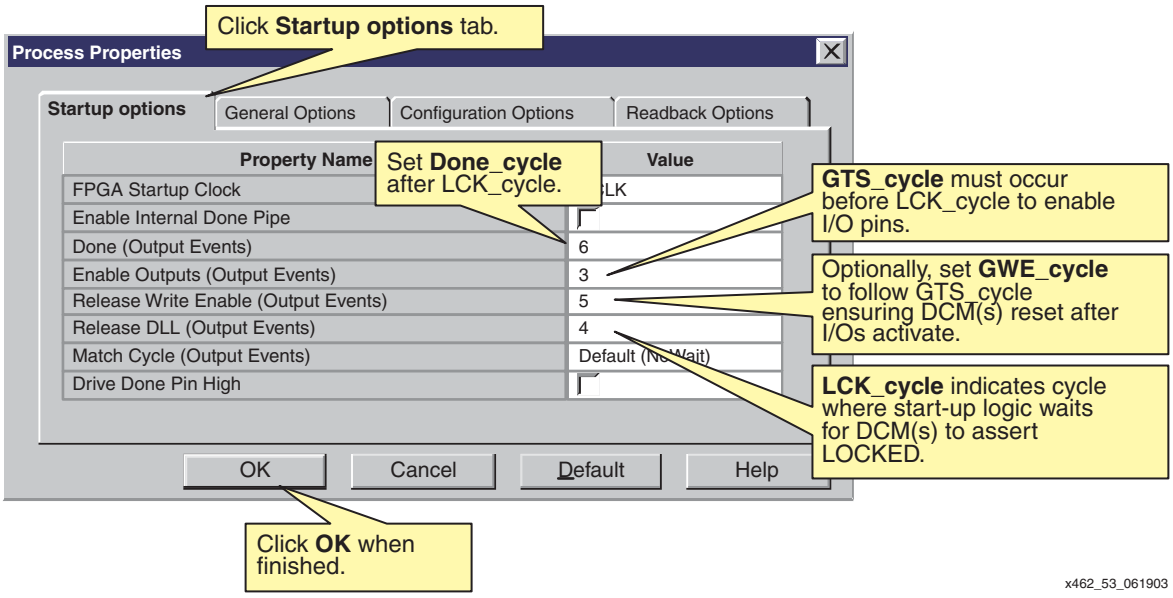



*Figure 53:* **BitGen Options**

The specific start-up phase timing and the timing of both the GWE_cycle and DONE_cycle are flexible. However, if using the STARTUP_WAIT attribute on a DCM, the GTS_cycle must always happen before the LCK_cycle. Otherwise, the DCM never locks and configuration never completes!

### Reset DCM During Partial Reconfiguration or During Full Reconfiguration via JTAG

Another bitstream option resets all the DCMs in the FPGA application during reconfiguration via the SelectMAP interface or during full or partial reconfiguration via the JTAG port. If the option is enabled, the DCMs are reset when the AGHIGH configuration command is issued during the SHUTDOWN command sequence. It is imperative to reset the DCMs when reconfiguring through JTAG. Change the bitstream generator options in Project Navigator (see "Setting Bitstream Generation Options in Project Navigator"). Click **Configuration options**, then check the **Reset DCM if SHUTDOWN & AGHIGH performed** option as shown in Figure 54.

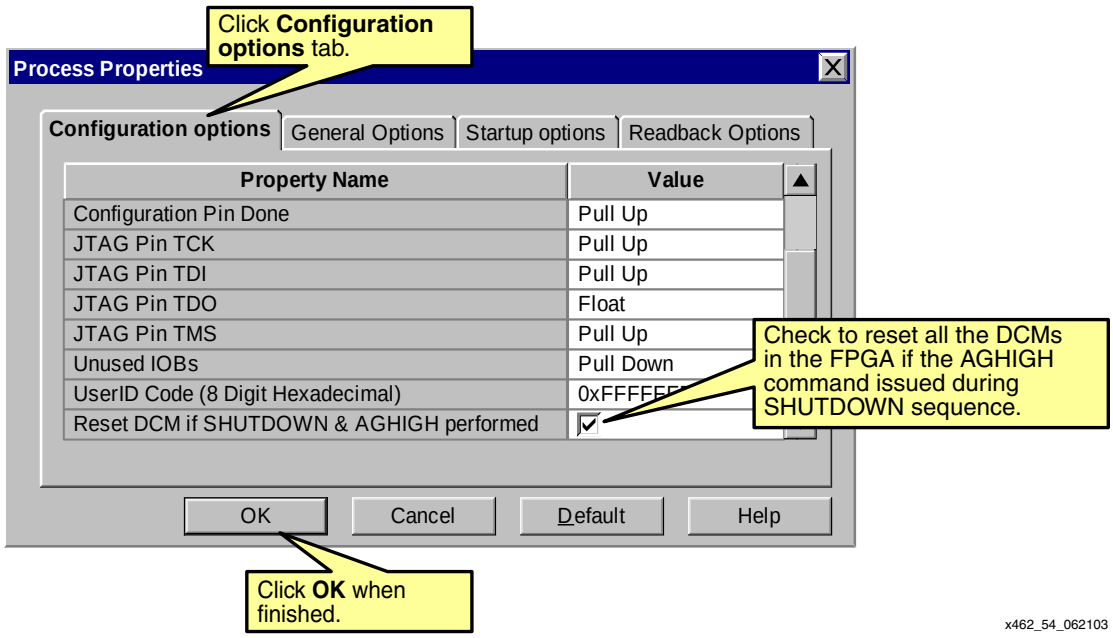



*Figure 54:* **Configuration Option Allows DCM Reset During Reconfiguration Process**

## Momentarily Stopping CLKIN

To reduce overall system noise while taking precision analog measurements, it is possible to momentarily stop the clock inputs to the DCM without adversely affecting the remainder of the FPGA application. This is possible, in part, because the DCM is an all-digital, stable system. The DCM must first lock to the input clock and assert the LOCKED output. If the DCM is not reset, it is possible to momentarily stop the CLKIN input clock with little impact to the deskew circuit, provided that these guidelines are followed:

- The clock must not be stopped for 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.

Although the above conditions do technically violate the clock input jitter specifications, the DCM LOCKED output stays High and remains High when the clock is restored. Consequently, the High on LOCKED does not necessarily mean that a valid clock is available. The above conditions technically do violate the clock input jitter specifications but work within the limits described above.

When CLKIN is stopped, an additional one to eight output clock cycles are still generated as the DCM's digital delay line is flushed. Similarly, once CLKIN is restarted, output clocks are not generated for one to four clocks cycles as the delay line is filled. The delay line usually fills within two or three clocks.

Likewise, it is also possible to phase shift the input clock. This phase shift propagates to the output one to four clocks after the original shift with no disruption to the DCM control.

Figure 55 shows an example where the CLKIN input clock is momentarily stopped. The figure also illustrates the corresponding effect on the CLK2X clock output.

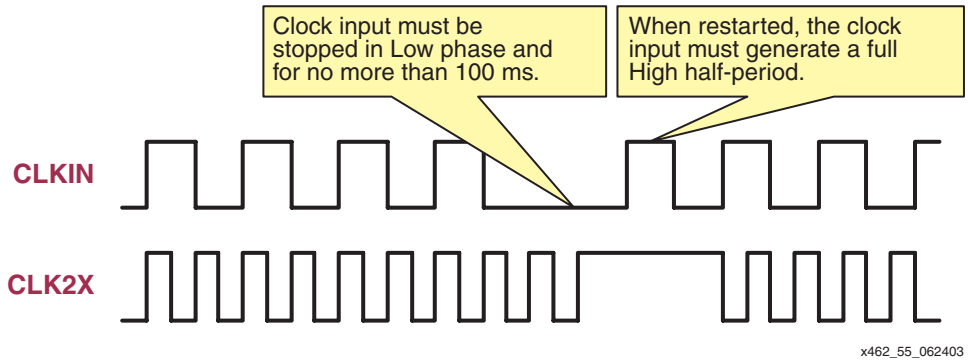



*Figure 55:* **Momentarily Stopping CLKIN Clock Input**

## Related Materials and References

- Spartan-3 Data Sheet (Module 2).
  "Digital Clock Manager". Description of features and capabilities.
  http://www.xilinx.com/bvdocs/publications/ds099-2.pdf
- Spartan-3 Data Sheet (Module 3).
  Timing and Jitter Specifications.
  http://www.xilinx.com/bvdocs/publications/ds099-3.pdf
- Libraries Guide, for ISE 5.2i by Xilinx, Inc.
  DCM Primitive description.
  http://toolbox.xilinx.com/docsan/xilinx5/pdf/docs/lib/lib.pdf
- Architecture Wizard.
  Free recorded E-learning session.
  http://www.xilinx.com/support/training/ise5-wizard.htm
- XAPP259: *System Interface Timing Parameters*.
  http://www.xilinx.com/xapp/xapp259.pdf
- XAPP268: *Dynamic Phase Alignment*.
  http://www.xilinx.com/xapp/xapp268.pdf
- XAPP622: *SDR LVDS Transmitter/Receiver*.
  http://www.xilinx.com/xapp/xapp622.pdf
- Development System Reference Guide. Chapter 15, "BitGen". Description of bitstream generation program and options. Pages 335-367.
  http://toolbox.xilinx.com/docsan/xilinx5/pdf/docs/dev/dev.pdf

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 07/09/03 | 1.0 | Initial Xilinx release. |

# Using Block RAM in Spartan-3 FPGAs

XAPP463 (v1.1.2) July 23, 2003


## Summary

For applications requiring large, on-chip memories, Spartan™-3 FPGAs provides plentiful, efficient SelectRAM™ memory blocks. Using various configuration options, SelectRAM blocks create RAM, ROM, FIFOs, large look-up tables, data width converters, circular buffers, and shift registers, each supporting various data widths and depths. This application note describes the features and capabilities of block SelectRAM and illustrates how to specify the various options using the Xilinx CORE Generator™ system or via VHDL or Verilog instantiation. Various non-obvious block RAM applications are discussed with references to additional tools, application notes, and documentation.


## Introduction

All Spartan-3 devices feature multiple block RAM memories, organized in columns. The total amount of block RAM memory depends on the size of the Spartan-3 device, as shown in Table 1.

*Table 1:* **Block RAM Available in Spartan-3 Devices**

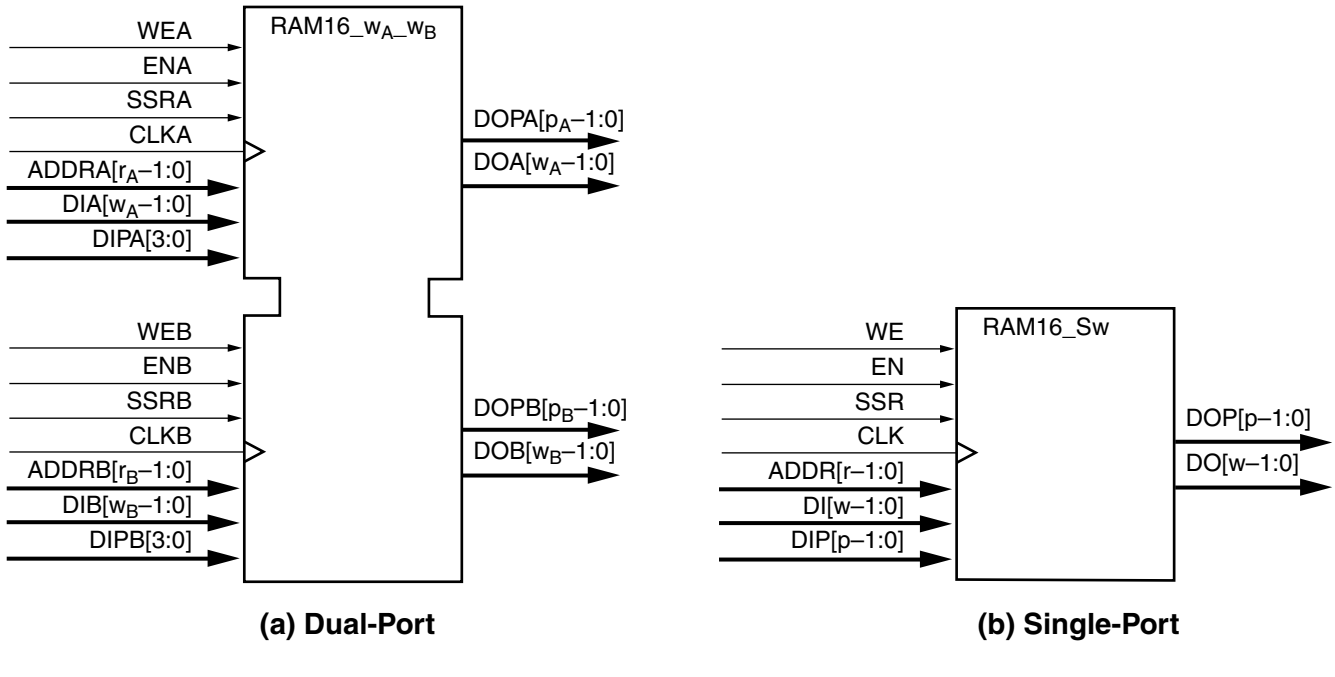
| Spartan-3 Device | RAM Columns | RAM Blocks Per Column | Total RAM Blocks | Total RAM Bits | Total RAM Kbits |
|---|---|---|---|---|---|
| XC3S50 | 1 | 4 | 4 | 73,728 | 72K |
| XC3S200 | 2 | 6 | 12 | 221,184 | 216K |
| XC3S400 | 2 | 8 | 16 | 294,912 | 288K |
| XC3S1000 | 2 | 12 | 24 | 442,368 | 432K |
| XC3S1500 | 2 | 16 | 32 | 589,824 | 576K |
| XC3S2000 | 2 | 20 | 40 | 737,280 | 720K |
| XC3S4000 | 4 | 24 | 96 | 1,769,472 | 1,728K |
| XC3S5000 | 4 | 26 | 104 | 1,916,928 | 1,872K |

**Notes:**
1. 1Kbit = 1,024 bits, per memory conventions.

Each block RAM contains 18,432 bits of fast static RAM, 16K bits of which is allocated to data storage and, in some memory configurations, an additional 2K bits allocated to parity or additional "plus" data bits. Physically, the block RAM memory has two completely independent access ports, labeled Port A and Port B. The structure is fully symmetrical, and both ports are interchangeable and both ports support data read and write operations. Each memory port is synchronous, with its own clock, clock enable, and write enable. Read operations are also synchronous and require a clock edge and clock enable.

Though physically a dual-port memory, block RAM simulates single-port memory in an application, as shown in Figure 1. Furthermore, each block memory supports multiple configurations or aspect ratios. Table 2 summarizes the essential SelectRAM features.

Cascade multiple block RAMs to create deeper and wider memory organizations, with a minimal timing penalty incurred through specialized routing resources.




**Notes:**
1. $w_A$ and $w_B$ are integers representing the total data path width (i.e., data bits plus parity bits) at ports A and B, respectively.
2. $p_A$ and $p_B$ are integers that indicate the number of data path lines serving as parity bits.
3. $r_A$ and $r_B$ are integers representing the address bus width at ports A and B, respectively.
4. The control signals CLK, WE, EN, and SSR on both ports have the option of inverted polarity.

*Figure 1:* **SelectRAM 18K Blocks Perform as Dual-Port (a) and Single-Port (b) Memory**

*Table 2:* **SelectRAM 18K Block Memory Features and Applications**

| | |
|---|---|
| Total RAM bits, including parity | 18,432 (16K data + 2K parity) |
| Memory Organizations | 16Kx1<br>8Kx2<br>4Kx4<br>2Kx8 (no parity)<br>2Kx9 (x8 + parity)<br>1Kx16 (no parity)<br>1Kx18 (x16 + 2 parity)<br>512x32 (no parity)<br>512x36 (x32 + 4 parity)<br>256x72 (single-port only) |
| Parity | Available and optional only for organizations greater than byte-wide. Parity bits optionally available as extra data bits. |
| Performance | 200 MHz (estimated) |

*Table 2:* **SelectRAM 18K Block Memory Features and Applications *(Continued)***

| | |
|---|---|
| Timing Interface | Simple synchronous interface. Similar to reading and writing from a register with a setup time for write operations and clock-to-output delay for read operations. |
| Single-Port | Yes |
| True Dual-Port | Yes |
| ROM, Initial RAM Contents | Yes |
| Mixed Data Port Widths | Yes |
| Power-Up Condition | User-defined data, defaults to zero |
| Potential Applications | Local data storage, FIFOs, elastic stores, register files, buffers, stacks, circular buffers, shift registers, delay lines, waveform storage and generation, direct digital synthesis, CAMs, associative memories, function tables, function generators, wide logic functions, code converters, encoders, decoders, counters, state machines, microsequencers, program storage for embedded processor(s) |

The Xilinx CORE Generator system supports various modules containing block RAM for Spartan-3 devices including:

- Embedded dual- or single-port RAM modules
- ROM modules
- Synchronous and asynchronous FIFO modules
- Content-Addressable Memory (CAM) modules

Furthermore, block RAM can be instantiated in any synthesis-based design using the appropriate “RAMB16” module from the Xilinx design library.

This application note describes the signals and attributes of the Spartan-3 block RAM feature, including details on the various attributes and applications for block RAM.

## Block RAM Location and Surrounding Neighborhood

As mentioned previously, block RAM is organized in columns. Figure 2 shows the Block RAM column arrangement for the XC3S200. The XC3S50 has a single column of block RAM, located two CLB columns from the left edge of the device. Spartan-3 devices larger than the XC3S50 have two columns of block RAM, adjacent to the left and right edges of the die, located two columns of CLBs from the I/Os at the edge. In addition to the block RAM columns at the edge, the XC3S4000 and XC3S5000 have two additional columns—a total of four columns—nearly equally distributed between the two edge columns. Table 1 describes the number of columns and the total amount of block RAM on a specific device. The edge columns make block RAM particularly useful in buffering or resynchronizing buses entering or leaving the Spartan-3 device.

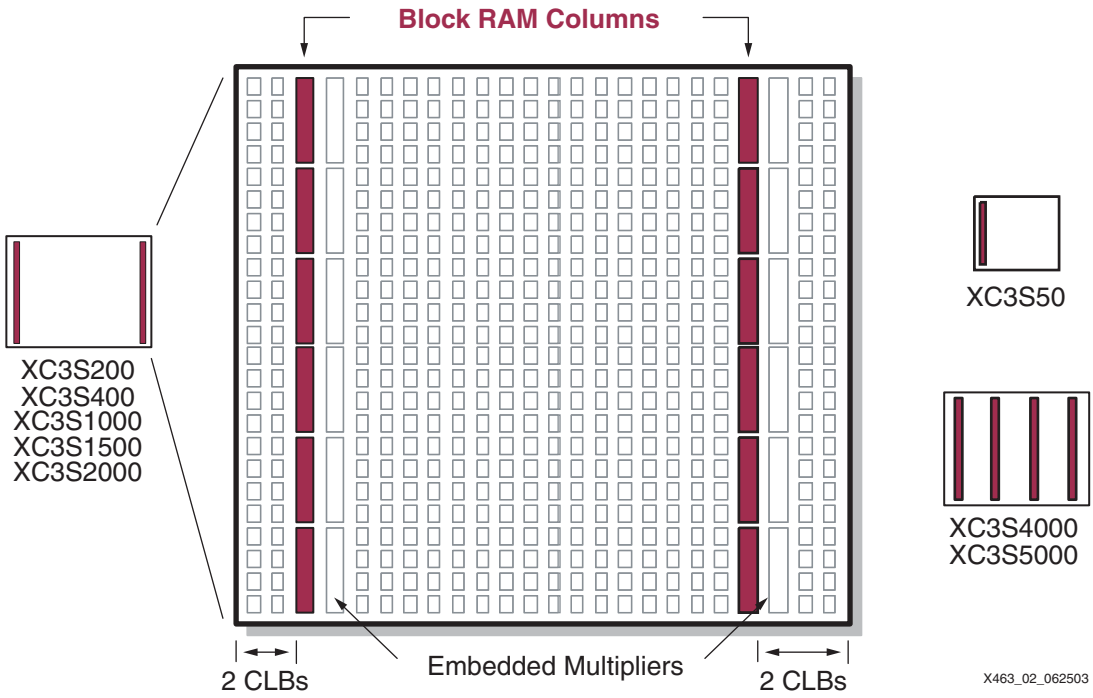



*Figure 2:* **Block RAMs Arranged in Columns with Detailed Floorplan of XC3S200**

Immediately adjacent to each block RAM is an embedded 18x18 hardware multiplier. Co-locating block RAM and the embedded multipliers improves the performance of some digital signal processing functions.

Special interconnect surrounding the block RAM provides efficient signal distribution for address and data. Furthermore, special provisions allow multiple block RAMs to be cascaded to create wider or deeper memories.

## Data Flows

Spartan-3 block RAM is constructed of true dual-port memory and simultaneously supports all the data flows and operations shown in Figure 3. Both ports access the same set of memory bits but with two potentially different address schemes depending on the port's data width.

1. Port A behaves as an independent single-port RAM supporting simultaneous read and write operations using a single set of address lines.
2. Port B behaves as an independent single-port RAM supporting simultaneous read and write operations using a single set of address lines.
3. Port A is the write port with a separate write address and Port B is the read port with a separate read address. The data widths for Port A and Port B can be different also.
4. Port B is the write port with a separate write address and Port A is the read port with a separate read address. The data widths for Port B and Port A can be different also.

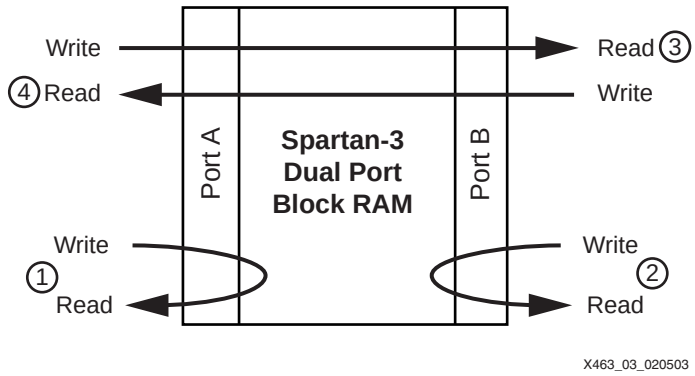



*Figure 3:* **Block RAM Support Single- and Dual-Port Data Transfers**

## Signals

The signals connected to a block RAM primitive divide into four categories, as listed below. Table 3 lists the block RAM interface signals, the signals names for both single-port and dual-port memories, and signal direction.

1. Data Inputs and Outputs
2. Parity Inputs and Outputs, available when a data port is byte-wide or wider
3. Address inputs to select a specific memory location
4. Various control signals that manage read, write, or set/reset operations.

*Table 3:* **Block RAM Interface Signals**

| Signal Description | Single Port | Dual Port | | Direction |
|---|---|---|---|---|
| | | Port A | Port B | |
| Data Input Bus | DI | DIA | DIB | Input |
| Parity Data Input Bus (available only for byte-wide and wider organizations) | DIP | DIPA | DIPB | Input |
| Data Output Bus | DO | DOA | DOB | Output |
| Parity Data Output (available only for byte-wide and wider organizations) | DOP | DOPA | DOPB | Output |
| Address Bus | ADDR | ADDRA | ADDRB | Input |
| Write Enable | WE | WEA | WEB | Input |
| Clock Enable | EN | ENA | ENB | Input |
| Synchronous Set/Reset | SSR | SSRA | SSRB | Input |
| Clock | CLK | CLKA | CLKB | Input |

### Data Inputs and Outputs

The total width of a port's data port includes both the data bus and the parity bus, when applicable, as shown in Figure 4. In the 512x36 organization, for example, the 36-bit data port width includes four parity bits as the more significant bits followed by the 32 data bits as the less significant bits.

The data and parity input and output signals are always buses; that is, in a 1-bit width configuration, the data input signal is DI[0] and the data output signal is DO[0].

**Data Input Bus — DI[#:0] (DIA[#:0], DIB[#:0])**

The Data Input bus is the source of data to be written into RAM.

Data at the DI input bus is written to the RAM location specified by the address input bus, ADDR, during a Low-to-High transition on the CLK input, when the clock enable EN and write enable WE inputs are High.

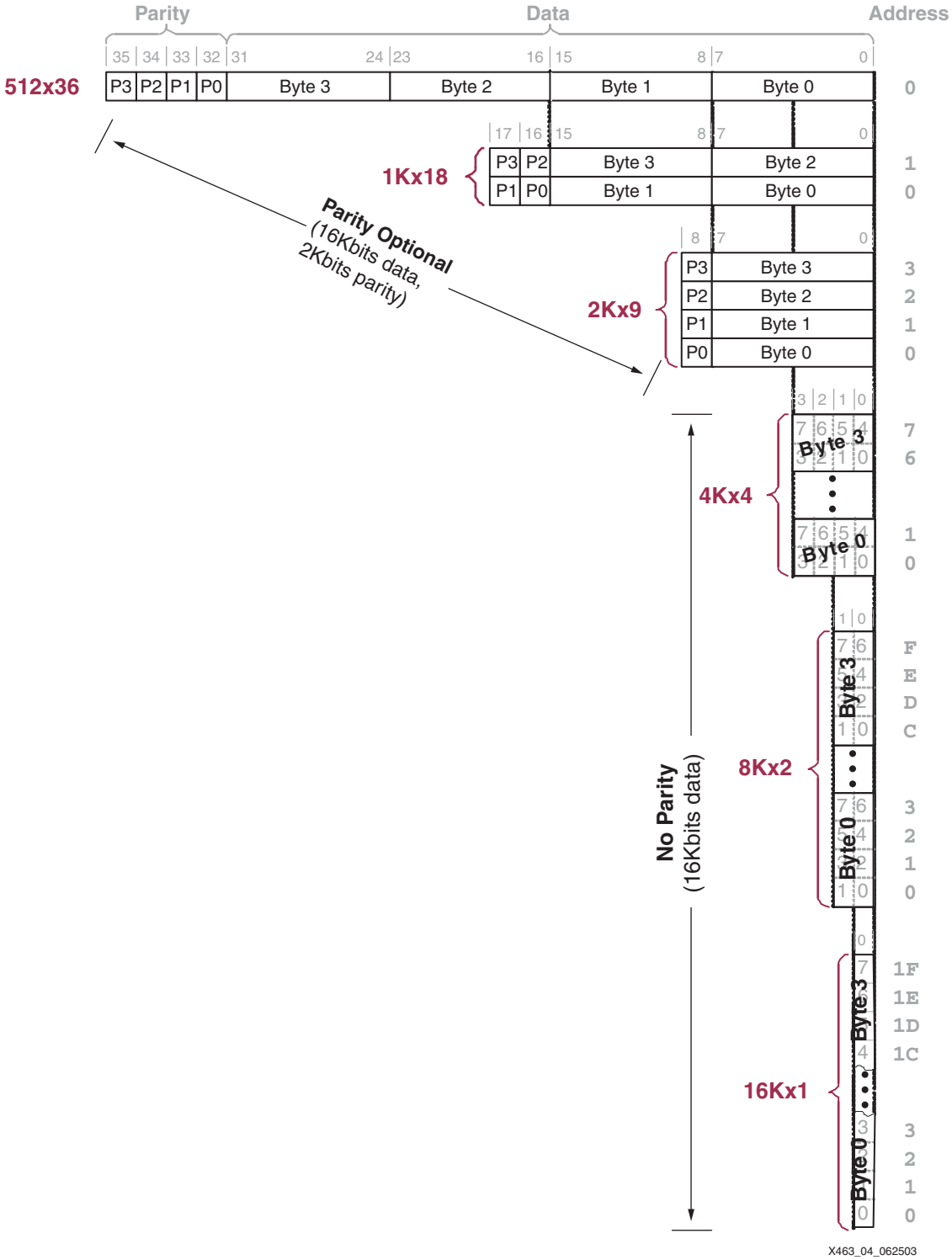



*Figure 4:* **Data Organization and Mapping Between Modes**

#### Data Output Bus — DO[#:0] (DOA[#:0], DOB[#:0])

The data output bus, DO, presents the contents of memory cells referenced by the address bus, ADDR, at the active clock edge during a read operation. During a simultaneous write operation, the behavior of the data output latches is controlled by the WRITE_MODE attribute (see **Read Behavior During Simultaneous Write — WRITE_MODE**, page 190).

## Parity Inputs and Outputs

Parity is only supported for data paths byte wide and wider.

Although referred to herein as “parity” bits, the parity inputs and outputs have no special functionality and can be used as additional data bits. For example, the parity bits could be used to hold additional information about a data word, tagging the data as code or data, positive or negative values, old or new data, *etc*.

Block RAM does not contain any special circuitry for generating or checking parity. These functions, if required by the application, are created using CLB logic resources.

#### Data Input Parity Bus — DIP[#:0] (DIPA[#:0], DIPB[#:0])

Data at the DIP input bus is written to the RAM location specified by the address input bus, ADDR, during a Low-to-High transition on the CLK input, when the clock enable EN and write enable WE inputs are High.

#### Data Output Parity Bus — DOP[#:0] (DOPA[#:0], DOPB[#:0])

The data output bus, DOP, presents the contents of memory cells referenced by the address bus, ADDR, at the active clock edge during a read operation. During a simultaneous write operation, the behavior of the data output latches is controlled by the WRITE_MODE attribute (see **Read Behavior During Simultaneous Write — WRITE_MODE**, page 190).

## Address Input

As dual-port RAM, both ports operate independently while accessing the same set of 18K-bit memory cells.

#### Address Bus — ADDR[#:0] (ADDRA[#:0], ADDRB[#:0])

The address bus selects the memory cells for read or write operations. The width of the address bus input determines the required address bus width, as shown in Table 5.

## Control Inputs

#### Clock — CLK (CLKA, CLKB)

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 and is rising edge triggered by default.

With default polarity, a Low-to-High transition on the clock (CLK) input controls read, write, and reset operations.

#### Enable — EN (ENA, ENB)

The enable input, EN, controls read, write, and set/reset operations. When EN is Low, no data is written and the outputs DO and DOP retain the last state. The polarity of EN is configurable and is active High by default.

When EN is asserted, minus an active synchronous set/reset input or write enable input, block RAM always reads the memory location specified by the address bus, ADDR, at the rising clock edge.

### Write Enable — WE (WEA, WEB)

The write enable input, WE, controls when data is written to RAM. When both EN and WE are asserted at the rising clock edge, the value on the data and parity input buses is written to memory location selected by the address bus.

The data output latches are loaded or not loaded according to the WRITE_MODE attribute.

The polarity of WE is configurable and is active High by default.

### Synchronous Set/Reset — SSR (SSRA, SSRB)

The synchronous set/reset input, SSR, forces the data output latches to value specified by the SRVAL attribute. When SSR and the enable signal, EN, are High, the data output latches for the DO and DOP outputs are synchronously set to a '0' or '1' according to the SRVAL parameter.

A Synchronous Set/Reset operation does not affect RAM memory cells and does not disturb write operations on the other port.

The polarity of SSR is configurable and is active High by default.

### Global Set/Reset — GSR

The global set/reset signal, GSR, is asserted automatically and momentarily at the end of device configuration. By instantiating the STARTUP primitive, the logic application can also assert GSR to restore the initial Spartan-3 state at any time. The GSR signal initializes the output latches to the INIT value. A GSR signal has no impact on internal memory contents.

Because GSR is a global signal and automatically connected throughout the device, the block RAM primitive does not have a GSR input pin.

### 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 without consuming additional logic resources.

### Unused Inputs

Tie any unused data or address inputs to logic '1'. Connecting the unused inputs High saves logic and routing resources compared to connecting the inputs Low.

## Attributes

A block RAM has a number of attributes that control its behavior as shown in Table 4 for VHDL and Verilog. The CORE Generator system uses slightly different values, as described below.

*Table 4:* **Block RAM Attributes and VHDL/Verilog Attribute Names**

| Function | VHDL or Verilog Attribute | Default Value |
|---|---|---|
| Number of Ports | Defined by instantiating the appropriate RAMB16 primitive | N/A |
| Memory Organization | Defined by instantiating the appropriate RAMB16 primitive | N/A |
| Initial Content for Data Memory, Loaded during Configuration | INIT_xx | Initialized to zero |
| Initial Content for Parity Memory, Loaded during Configuration | INITP_xx | Initialized to zero |
| Data Output Latch Initialization | INIT (single-port)<br>INIT_A, INIT_B (dual-port) | Initialized to zero |

*Table 4:* **Block RAM Attributes and VHDL/Verilog Attribute Names *(Continued)***

| Function | VHDL or Verilog Attribute | Default Value |
|---|---|---|
| Data Output Latch Synchronous Set/Reset Value | SRVAL (single-port)<br>SRVAL_A, SRVAL_B (dual-port) | Reset to zero |
| Data Output Latch Behavior during Write | WRITE_MODE | WRITE_FIRST |
| Block RAM Location | LOC | N/A |

## Number of Ports

Although physically dual-port memory, each block RAM performs as either single-port or dual-port memory. The method to specify the number of ports depends on the design entry tool.

### CORE Generator System

As shown in Figure 5, the Xilinx CORE Generator system provides module generators for various types of memory blocks. Choose single- or dual-port block memories, or use the higher-level functions to create FIFOs, content-addressable memories (CAMs), and so forth.

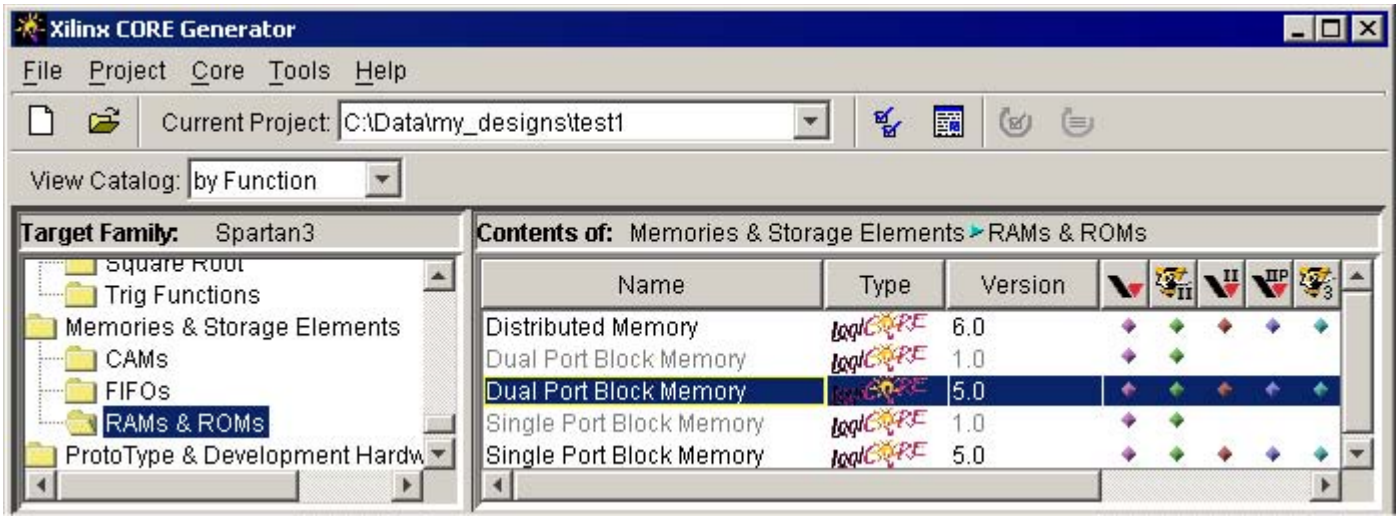



*Figure 5:* **Selecting a Block RAM Function in CORE Generator System**

### VHDL or Verilog Instantiation

The Xilinx design libraries contain single- and dual-port memory primitives similar to those shown in Figure 1. Select among the various primitives to choose single- or dual-port memory, as well as the memory organization or aspect ratio of the memory. See Table 5 and Table 6 for single-port and dual-port block RAM primitives, respectively.

## Memory Organization/Aspect Ratio

The data organization or aspect ratio of a RAM block is configurable, as shown in Table 5. If the data path is byte-wide or wider, then the block RAM also provides additional bits to support parity for each byte. Consequently, a 1Kx18 memory organization is 18 bits wide with 16 bits (two bytes) allocated to data plus two parity bits, one for each byte. Also, the physical amount of memory accessible from a port depends on the memory organization. For memories byte-wide and wider, there are 18K memory bits accessible. For narrower memories, only 16K bits are accessible due to the lack of parity bits in these organizations. Essentially, 16K bits are allocated to data, 2K bits to parity on the 18K-bit block RAM. See Figure 4 for details on data mapping for and between each memory organization.

*Table 5:* **Block RAM Data Organizations/Aspect Ratios**

| Organization | Memory Depth | Data Width | Parity Width | DI/DO | DIP/DOP | ADDR | Single-Port Primitive | Total RAM Kbits |
|---|---|---|---|---|---|---|---|---|
| 512x36 | 512 | 32 | 4 | (31:0) | (3:0) | (8:0) | **RAMB16_S36** | 18K |
| 1Kx18 | 1024 | 16 | 2 | (15:0) | (1:0) | (9:0) | **RAMB16_S18** | 18K |
| 2Kx9 | 2048 | 8 | 1 | (7:0) | (0:0) | (10:0) | **RAMB16_S9** | 18K |
| 4Kx4 | 4096 | 4 | - | (3:0) | - | (11:0) | **RAMB16_S4** | 16K |
| 8Kx2 | 8192 | 2 | - | (1:0) | - | (12:0) | **RAMB16_S2** | 16K |
| 16Kx1 | 16384 | 1 | - | (0:0) | - | (13:0) | **RAMB16_S1** | 16K |

### CORE Generator System — Memory Size

The CORE Generator system creates a wide variety of memories with very flexible aspect ratios. Unlike the actual block RAM primitive, the CORE generator system does not differentiate between data and parity bits and considers all bits data bits. For dual-port memories, each port can have different organizations or aspect ratios.

Within the CORE Generator system, locate the Memory Size group and enter the desired memory organization, as shown in Figure 6.


*Figure 6:* **Selecting Memory Width and Depth in CORE Generator System**

### VHDL or Verilog Instantiation

The aspect ratio is defined at design time by specifying or instantiating the appropriate SelectRAM component. Table 5 indicates the SelectRAM component for single-port RAM. For single-port RAM, the proper component name is `RAMB16_S`***n***, where ***n*** is the data path width including both the data bits plus parity bits. For example, a 1Kx18 single-port RAM uses component `RAMB16_S18`. In this example, ***n***=18 because there are 16 data bits plus 2 parity bits.

Selecting a dual-port memory is slightly more complex because the two memory ports may have different aspect ratios. For dual-port RAM, the proper component name is `RAMB16_S`***m***`_S`***m***, where ***m*** is the data path width for Port A and ***n*** is the width for Port B. For example, using the suffix shown in Table 6, if Port A is organized a 2Kx9 and Port B is organized as 1Kx18, then the proper dual-port RAM component is `RAMB16_S9_S18`. In this example, ***m***=9 and ***n***=18.

*Table 6:* **Dual-Port RAM Component Suffix Appended to "RAMB16"**

| | | Port A | | | | | |
|---|---|---|---|---|---|---|---|
| | | **16Kx1** | **8Kx2** | **4Kx4** | **2Kx9** | **1Kx18** | **512x36** |
| **Port B** | 16Kx1 | **_S1_S1** | | | | | |
| | 8Kx2 | **_S1_S2** | **_S2_S2** | | | | |
| | 4Kx4 | **_S1_S4** | **_S2_S4** | **_S4_S4** | | | |
| | 2Kx9 | **_S1_S9** | **_S2_S9** | **_S4_S9** | **_S9_S9** | | |
| | 1Kx18 | **_S1_S18** | **_S2_S18** | **_S4_S18** | **_S9_S18** | **_S18_S18** | |
| | 512x36 | **_S1_S36** | **_S2_S36** | **_S4_S36** | **_S9_S36** | **_S18_S36** | **_S36_S36** |

### Address and Data Mapping Between Two Ports

In dual-port mode, both ports access the same set of memory cells. However, both ports may have the same or different memory organization or aspect ratio. Figure 4 shows how the same data set may appear with different aspect ratios.

There are extra bits available to store parity for memory organizations that are byte-wide or wider. The extra parity bits are designed to be associated with a particular byte and these parity bits appear as the more-significant bits on the data port. For example, if a x36 data word (32 data, 4 parity) is addressed as two x18 halfwords (16 data, 2 parity), the parity bits associated with each data byte are mapped within the block RAM to appropriate parity bits. The same effect happens when the x36 data word is mapped as four x9 words. The extra parity bits are not available if the data port is configured as x4, x2, or x1.

The following formulas provide the starting and ending address for data when the two ports have different memory organizations. Find the starting and ending address for Port X given the address and port width of Port Y and the port width of Port X.

$$\text{START\_ADDRESS}_X = \text{INTEGER}\left(\frac{\text{ADDRESS}_Y \bullet \text{WIDTH}_Y}{\text{WIDTH}_X}\right)$$

$$\text{END\_ADDRESS}_X = \text{INTEGER}\left(\frac{((\text{ADDRESS}_Y + 1) \bullet \text{WIDTH}_Y) - 1}{\text{WIDTH}_X}\right)$$

If, due the memory organization, one port includes parity bits and the other does not, then the above equations are invalid and the values for width should only include the data bits. The parity bits are not available on any port that is less than 8 bits wide.

## Content Initialization

By default, block RAM memory is initialized with all zeros during the device configuration sequence. However, the contents can also be initialized with user-defined data. Furthermore, the RAM contents are protected against spurious writes during configuration.

### CORE Generator System — Load Init File

To specify the initial RAM contents for a CORE Generator block RAM function, create a coefficients (.coe) file. A simple example of a coefficients file appears in Figure 7. At a minimum, define the radix for the initialization data—*i.e.*, base 2, 10, or 16—and then specify the RAM contents starting with the data at location 0, followed by data at subsequent locations.

```
memory_initialization_radix=16;
memory_initialization_vector= 80, 0F, 00, 0B, 00, 0C, …, 81;
```

*Figure 7:* **A Simple Coefficients File (.coe) Example**

To include the coefficients file, locate the appropriate section in the CORE Generator wizard and check **Load Init File**, as shown in Figure 8. Then, click **Load File** and select the coefficients file.

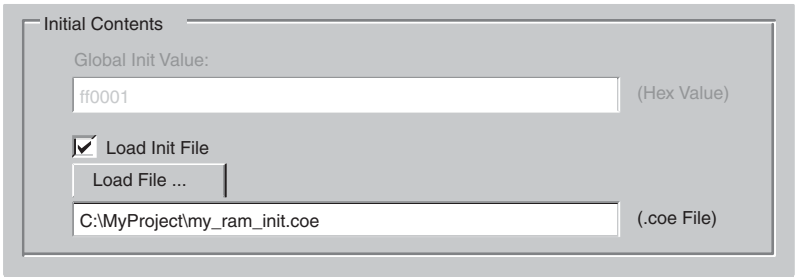



*Figure 8:* **Specifying Initial RAM Contents in CORE Generator System**

### VHDL or Verilog Instantiation — INIT_xx, INITP_xx

For VHDL and Verilog instantiation, there are two different types of initialization attributes. The **INIT_xx** attributes define the initial contents of the data memory locations. The **INITP_xx** attributes define the initial contents of the parity memory locations.

The **INIT_xx** attributes on the instantiated primitive define the initial memory contents. There are 64 initialization attributes, named **INIT_00** through **INIT_3F**. Each **INIT_xx** attribute is a 64-digit (256-bit) hex-encoded bit vector. The memory contents can be partially initialized and any unspecified locations are automatically completed with zeros.

The following formula defines the bit positions for each **INIT_xx** attribute.

Given yy = convert_hex_to_decimal(xx), **INIT_xx** corresponds to the following memory cells.

- Starting Location: [(yy + 1) * 256] –1
- End Location: (yy) * 256

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

- yy = convert_hex_to_decimal(0x1F) = 31
- Starting Location: [(31+1) * 256] –1 = 8191
- End Location: 31 * 256 = 7936

*Table 7:* **VHDL/Verilog RAM Initialization Attributes for Block RAM**

| Attribute | From | To |
|---|---|---|
| **INIT_00** | 255 | 0 |
| **INIT_01** | 511 | 256 |
| **INIT_02** | 767 | 512 |
| **…** | … | … |
| **INIT_3F** | 16383 | 16128 |

The **INITP_xx** attributes define the initial contents of the memory cells corresponding to parity bits, *i.e.*, those bits that connect to the DIP/DOP buses. 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 (256-bit) hex-encoded bit vector and behaves like an **INIT_xx** attribute. The same formula calculates the bit positions initialized by a particular **INITP_xx** attribute.

## Data Output Latch Initialization

The block RAM output latches can be initialized to a user-specified value immediately after configuration or whenever the global set/reset signal, GSR, is asserted. For dual-port memories, there is a separate initialization value for each port.

If no value is specified, the output latch is initialized to zero.

### CORE Generator System — Global Init Value

Figure 9 describes how to specify the initial value for data output latches in the CORE Generator system. The value, specified in hexadecimal, should include one bit per the specified data width. For dual-port memories, there is a separate initialization value for each port.


*Figure 9:* **Specifying Initial Value for Block RAM Data Output Latches**

### VHDL or Verilog Instantiation — INIT (INIT_A and INIT_B)

For VHDL or Verilog, the INIT attribute (or INIT_A and INIT_B for dual-port memories) defines the output latch value after configuration. The INIT (or INIT_A and INIT_B) attribute specifies the initial value for the data and, if applicable, the parity bits. Figure 4 shows the expected bit format for each memory organization with parity bits—if applicable—as the more significant bits followed by the data bits. For example, the initialization value for a 2Kx9 memory would be nine bits wide and would include one parity bit followed by eight data bits. These attributes are hex-encoded bit vectors and the default value is 0.

## Data Output Latch Synchronous Set/Reset Value

When the synchronous set/reset input, SSR, is asserted, the data output latches are set or reset according to the set/reset value attribute. For dual-port memories, there is a separate initialization value for each port.

If no value is specified, the output latch is reset to zero during a valid Synchronous Set/Reset operation.

### CORE Generator System — Init Value (SINIT)

Figure 10 describes how to specify the synchronous set/reset value for data output latches in the CORE Generator system. Check the **SINIT pin** and then specify the synchronous set/reset value in hexadecimal, with one bit per the specified data width. For dual-port memories, there is a separate value for each port.

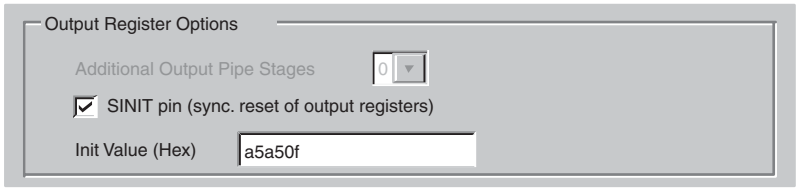



*Figure 10:* **Specifying the Output Data Latch Set/Reset Value**

### VHDL or Verilog Instantiation — SRVAL (SRVAL_A and SRVAL_B)

For VHDL or Verilog, the SRVAL attribute (or SRVAL_A and SRVAL_B for dual-port memories) defines the output latch value after configuration. The SRVAL (or SRVAL_A and SRVAL_B) attribute specifies the initial value for the data and, if applicable, the parity bits. Figure 4 shows the expected bit format for each memory organization with parity bits—if applicable—as the more significant bits followed by the data bits. These attributes are hex-encoded bit vectors and the default value is 0.

## Read Behavior During Simultaneous Write — WRITE_MODE

To maximize data throughput and utilization of the dual-port memory at each clock edge, block RAM memory supports one of three write modes for each memory port. These different modes determine which data is available on the output latches after a valid write clock edge to the same port. The default mode, WRITE_FIRST, provides backwards compatibility with the older Virtex™/E and Spartan-IIE FPGA architectures and is also the default behavior for Virtex-II/Pro devices. However, READ_FIRST mode is the most useful as it increases the efficiency of block RAM memory at each clock cycle, allowing designs to use maximum bandwidth. In READ_FIRST mode, a memory port supports simultaneous read and write operations to the same address on the same clock edge, free of any timing complications.

Table 8 outlines how the WRITE_MODE setting affects the output data latches on the same port, and how it affects the output latches on the opposite port during a simultaneous access to the same address.

*Table 8:* **WRITE_MODE Affects Data Output Latches During Write Operations**

| Write Mode | Effect on Same Port | Effect on Opposite Port (dual-port mode only, same address) |
|---|---|---|
| **WRITE_FIRST** Read After Write (Default) | Data on DI, DIP inputs written into specified RAM location and simultaneously appears on DO, DOP outputs. | Invalidates data on DO, DOP outputs. |
| **READ_FIRST** Read Before Write (Recommended) | Data from specified RAM location appears on DO, DOP outputs.<br>Data on DI, DIP inputs written into specified location. | Data from specified RAM location appears on DO, DOP outputs. |
| **NO_CHANGE** No Read on Write | Data on DO, DOP outputs remains unchanged.<br>Data on DI, DIP inputs written into specified location. | Invalidates data on DO, DOP outputs. |

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

### WRITE_FIRST or Transparent Mode (Default)

The WRITE_FIRST mode is the default operating mode for backward compatibility reasons. For new designs, READ_FIRST mode is recommended.

In this mode, the input data is written into the addressed RAM location memory and simultaneously stored in the data output latches, resulting in a transparent write operation, as shown in Figure 11. The WRITE_FIRST mode provides backwards compatibility with the 4K-bit blocks RAMs on Virtex/E and Spartan-II/E FPGAs and is also the default mode for Virtex-II/Pro block RAMs.

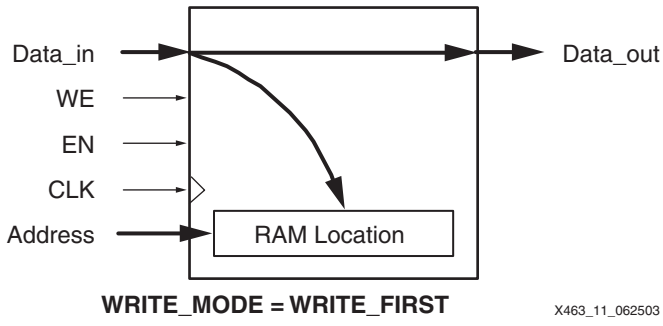



*Figure 11:* **Data Flow during a WRITE_FIRST Write Operation**

Figure 12 demonstrates that a valid write operation during a valid read operation results in the write data appearing on the data output.

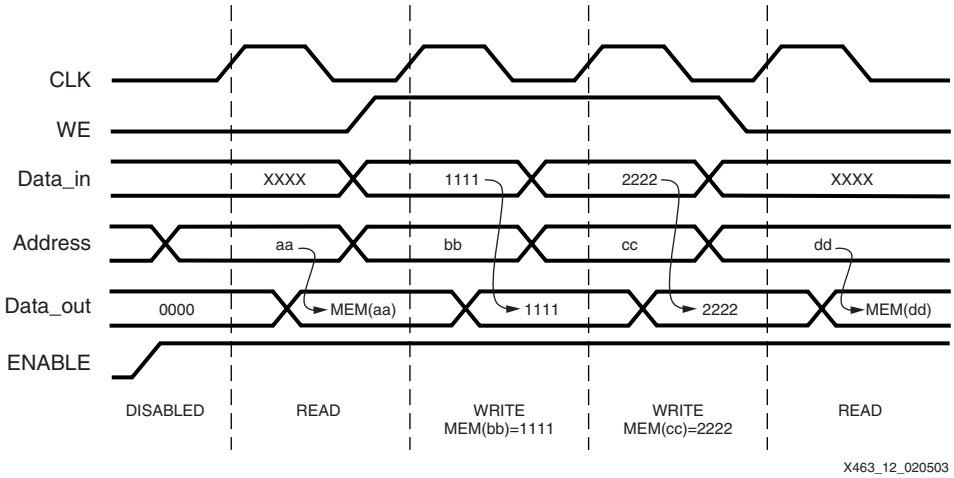



*Figure 12:* **WRITE_FIRST Mode Waveforms**

### 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 new input data is stored in memory, resulting in a read-before-write operation shown in Figure 13. The older RAM data appears on the data output while the new RAM data is stored in the specified RAM location. READ_FIRST mode is the recommended operating mode.

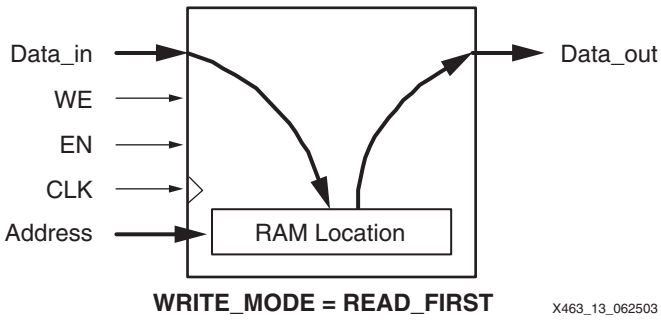



*Figure 13:* **Data Flow during a READ_FIRST Write Operation**

Figure 14 demonstrates that the older RAM data always appears on the data output, regardless of a simultaneous write operation.

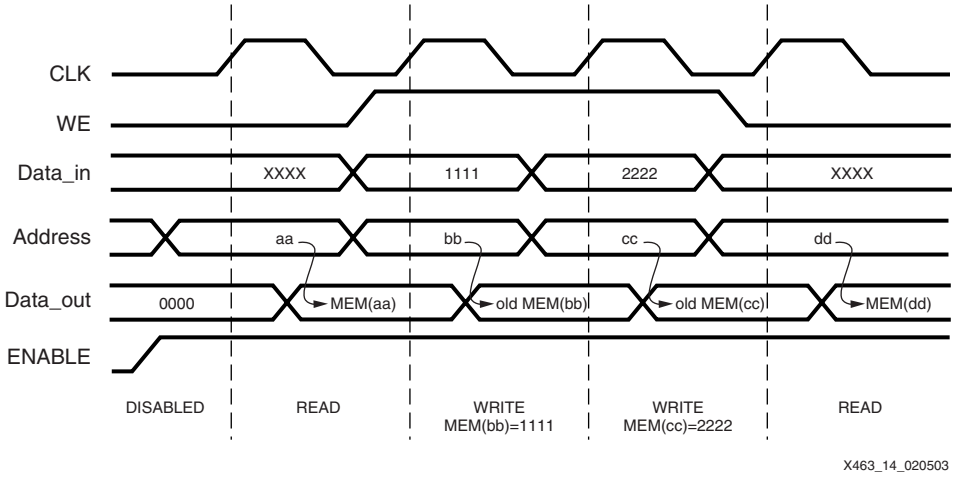



*Figure 14:* **READ_FIRST Mode Waveforms**

This mode is particularly useful for building circular buffers and large, block-RAM-based shift registers. Similarly, this mode is useful when storing FIR filter taps in digital signal processing applications. Old data is copied out from RAM while new data is written into RAM.

### NO_CHANGE Mode

In NO_CHANGE mode, the output latches are disabled and remain unchanged during a simultaneous write operation, as shown in Figure 15. This behavior mimics that of simple synchronous memory where a memory location is either read or written during a clock cycle, but not both.

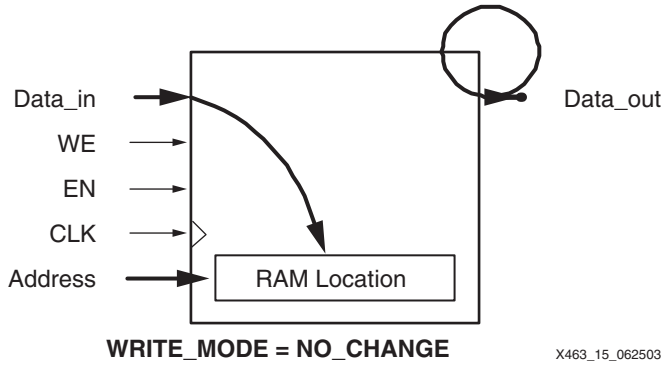



*Figure 15:* **Data Flow during a NO_CHANGE Write Operation**

The NO_CHANGE mode is useful in a variety of applications, including those where the block RAM contains waveforms, function tables, coefficients, and so forth. The memory can be updated without affecting the memory output.

Figure 16 shows that the data output retains the last read data if there is a simultaneous write operation on the same port.

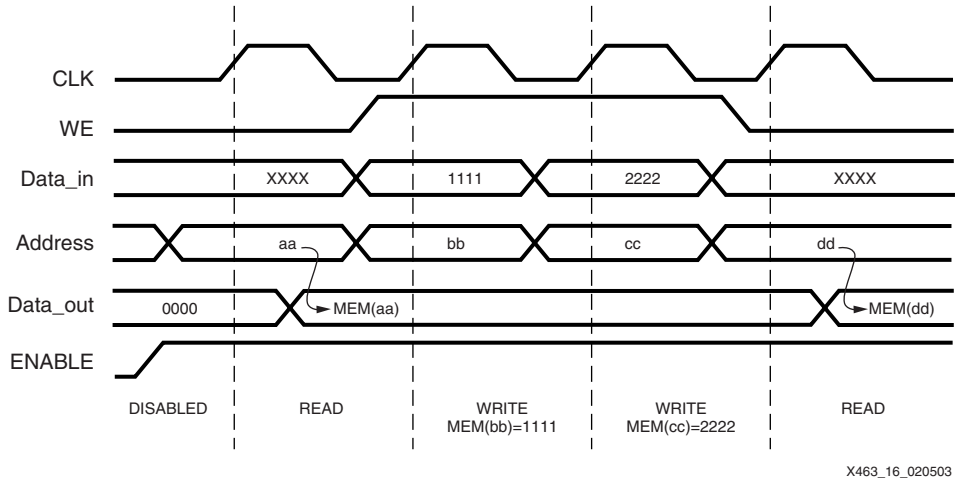



*Figure 16:* **NO_CHANGE Mode Waveforms**

### CORE Generator System — Write Mode

To specify the WRITE_MODE in the CORE Generator system, locate the settings for Write Mode as shown in Figure 17. Select between Read After Write (WRITE_FIRST), Read Before Write (READ_FIRST) or No Read On Write (NO_CHANGE).

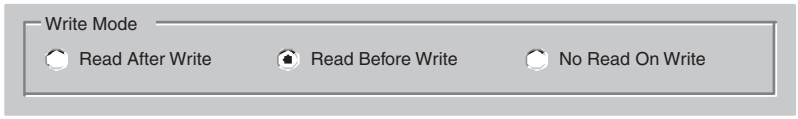



*Figure 17:* **Selecting the Write Mode in CORE Generator System**

### VHDL or Verilog Instantiation — WRITE_MODE

When instantiating block RAM, specify the write mode via the WRITE_MODE attribute. Acceptable values include WRITE_FIRST, READ_FIRST, and NO_CHANGE, as demonstrated in the examples in the appendices.

## Location Constraints (LOC)

In general, it is best to allow the Xilinx ISE software to assign a block RAM location. However, block RAMs can be constrained to specific locations on a Spartan-3 device using an attached LOC property. Block RAM placement locations are device specific and 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 lower-left block RAM location on the device, as shown in Figure 18. The upper-right block RAM location depends on **n**, the number of block RAM columns, and **m**, the number of block RAM rows, as provided in Table 1.

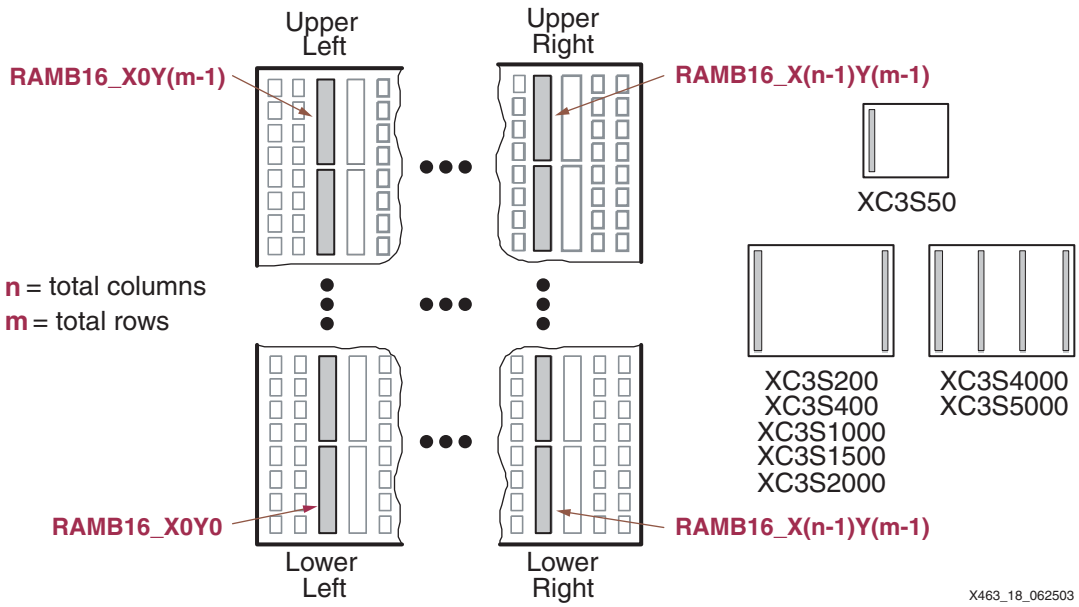



*Figure 18:* **Block RAM LOC Coordinates**

Location attributes cannot be specified directly in the CORE Generator system. However, location constraints can be added to VHDL or Verilog instantiations.

## Block RAM Operation

Table 9 describes the behavior of block RAM and assumes that all control signals use their default, active-High behavior. However, the control signals can be inverted in the design if necessary. The table and following text describes the behavior for a single memory port. In dual-port mode, both ports perform as independent single-port memories.

All read and write operations to block RAM are synchronous. All inputs have a set-up time relative to clock and all outputs have a clock-to-output time.

*Table 9:* **Block RAM Function Table**

| Input Signals | | | | | | | | Output Signals | | RAM Contents | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **GSR** | **EN** | **SSR** | **WE** | **CLK** | **ADDR** | **DIP** | **DI** | **DOP** | **DO** | **Parity** | **Data** |
| **Immediately After Configuration** | | | | | | | | | | | |
| **Loaded During Configuration** | | | | | | | | X | X | INITP_xx[2] | INIT_xx[2] |
| **Global Set/Reset Immediately after Configuration** | | | | | | | | | | | |
| 1 | X | X | X | X | X | X | X | INIT[3] | INIT | No Chg | No Chg |
| **RAM Disabled** | | | | | | | | | | | |
| 0 | 0 | X | X | X | X | X | X | No Chg | No Chg | No Chg | No Chg |
| **Synchronous Set/Reset** | | | | | | | | | | | |
| 0 | 1 | 1 | 0 | ↑ | X | X | X | SRVAL[4] | SRVAL | No Chg | No Chg |
| **Synchronous Set/Reset during Write RAM** | | | | | | | | | | | |
| 0 | 1 | 1 | 1 | ↑ | addr | pdata | Data | SRVAL | SRVAL | RAM(addr) ←pdata | RAM(addr) ← data |

*Table 9:* **Block RAM Function Table *(Continued)***

| Input Signals | | | | | | | | Output Signals | | RAM Contents | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| GSR | EN | SSR | WE | CLK | ADDR | DIP | DI | DOP | DO | Parity | Data |
| **Read RAM, no Write Operation** | | | | | | | | | | | |
| 0 | 1 | 0 | 0 | ↑ | addr | X | X | RAM(pdata) | RAM(data) | No Chg | No Chg |
| **Write RAM, Simultaneous Read Operation** | | | | | | | | | | | |
| 0 | 1 | 0 | 1 | ↑ | addr | pdata | Data | **WRITE_MODE =** WRITE_FIRST[5] **(default)** | | | |
| | | | | | | | | pdata | data | RAM(addr) ←pdata | RAM(addr) ← data |
| | | | | | | | | **WRITE_MODE =** READ_FIRST[6] **(recommended)** | | | |
| | | | | | | | | RAM(data) | RAM(data) | RAM(addr) ←pdata | RAM(addr) ←pdata |
| | | | | | | | | **WRITE_MODE =** NO_CHANGE[7] | | | |
| | | | | | | | | No Chg | No Chg | RAM(addr) ←pdata | RAM(addr) ←pdata |

**Notes:**
1. No Chg = No Change, addr = address to RAM, data = RAM data, pdata = RAM parity data.
2. Refer to **Content Initialization**, page 187.
3. Refer to **Data Output Latch Initialization**, page 188.
4. Refer to **Data Output Latch Synchronous Set/Reset Value**, page 189.
5. Refer to **WRITE_FIRST or Transparent Mode (Default)**, page 190.
6. Refer to **READ_FIRST or Read-Before-Write Mode**, page 191.
7. Refer to **NO_CHANGE Mode**, page 192.

## RAM Contents Initialized During Configuration

The initial RAM contents, if specified, are loaded during the Spartan-3 configuration process. If no contents are specified, the RAM cells are loaded with zero. The RAM contents are protected against spurious writes during configuration.

## Global Set/Reset Initializes Data Output Latches Immediately After Configuration or Global Reset

Immediately following configuration, the Spartan-3 device begins its start-up procedure and asserts the global set/reset signal, GSR, to initialize the state of all flip-flops and registers. The initial contents of the block RAM output latches, INIT, are asynchronously loaded at this time. The GSR signal does not change or re-initialize the RAM contents.

## Enable Input Activates or Disables RAM

If the block RAM is disabled—*i.e.*, EN is Low—then the block RAM retains its present state. The enable input must be High for any other operations to proceed.

## Synchronous Set/Reset Initializes Data Output Latches

If the block RAM is enabled (EN is High) and the Synchronous Set/Reset signal is asserted High, then the data output latches are initialized at the next rising clock edge. The SRVAL attribute defines the synchronous set/reset state for the data output latches. This operation is different the operation caused by the global set/reset signal, GSR, immediately after configuration. The synchronous set/reset input affects the specific RAM block whereas the GSR signal affects the entire device.

## Simultaneous Write and Synchronous Set/Reset Operations

If a simultaneous write operation occurs during the synchronous set/reset operation, then the data on the DI and DIP inputs is stored at the RAM location specified by the ADDR input. However, the data output latches are initialized to the SRVAL attribute value as described immediately above.

## Read Operations Occur on Every Clock Edge When Enable is Asserted

Read operations are synchronous and require a clock edge and an asserted clock enable. The data output behavior depends on whether or not a simultaneous write operation occurs during the read cycle.

If no simultaneous write cycle occurs during a valid read cycle, then the read address is registered on the read port and the data stored in RAM at that address is simply loaded into the output latches after the RAM access interval passes.

However, if there is a simultaneous write cycle during the read cycle, then the output behavior depends on which of the three write modes is selected, as described immediately below.

## Write Operations Always Have Simultaneous Read Operation, Data Output Latches Affected

During a Write operation, a simultaneous Read operation occurs. The WRITE_MODE attribute determines the behavior of the data output latches during the Write operation (refer to **Read Behavior During Simultaneous Write — WRITE_MODE**, page 190). By default, WRITE_MODE is WRITE_FIRST and the data output latches and the addressed RAM locations are updated with the input data during a simultaneous Write operation. When WRITE_MODE is READ_FIRST, the output latches are updated with the data previously stored in the addressed RAM location and the new data on the DI and DIP inputs is stored at the address RAM location. When WRITE_MODE is NO_CHANGE, the data output latches are unaffected by a simultaneous Write operation and retain their present state.

## General 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 RAM 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 without arbitration. Each port has its own clocking, control, address, read/write functions, initialization, and data width.
- Output ports are latched with a self-timed circuit, guaranteeing glitch-free read operations. The state of the output port does not change until the port executes another read or write operation.

## Functional Compatibility with Other Xilinx FPGA Families

The block RAM on Spartan-3 FPGAs is functionally identical to block RAM on the Xilinx Virtex-II/Pro FPGA families. Consequently, design tools that support Virtex-II and Virtex-II Pro block RAM also support with Spartan-3 FPGAs.

## Dual-Port RAM Conflicts and Resolution

As a dual-port RAM, the block RAM allows both ports to simultaneously access the same memory cell. Potentially, conflicts arise under the following conditions.

1. If the clock inputs to the two ports are asynchronous, then conflicts occur if clock-to-clock setup time requirements are violated.
2. Both memory ports write different data to the same RAM location during a valid write cycle.
3. If a port uses WRITE_MODE=NO_CHANGE or WRITE_FIRST, a write to the port invalidates the read data output latches on the opposite port.

If Port A and Port B different memory organizations and consequently different widths, only the overlapping bits are invalid when conflicts occur.

### Timing Violation Conflicts

When one port writes to a given memory cell, the other port must not address that memory cell—either for a write or a read operation—within the clock-to-clock setup window specified in the Spartan-3 data sheet. Figure 19 describes this situation where both ports operate from asynchronous clock inputs.

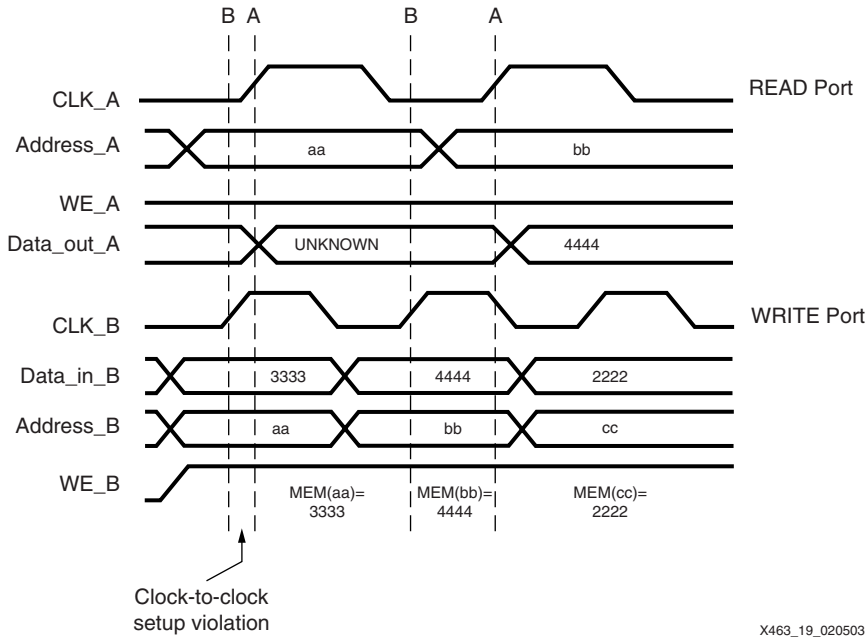



*Figure 19:* **Clock-to-Clock Timing Conflicts**

The first rising edge on CLK_A 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 because Data_in_B, Address_B, and WE_B all had sufficient set-up time before the rising edge on CLK_B. Unfortunately, the read operation on port A is invalid because it depends on the RAM contents being written to Address_B and the read clock, CLK_A, happened too soon after the write clock, CLK_B.

On the second rising edge of CLK_B, there is another valid write operation to port B. The memory location at address (bb) contains 4444. Data on the Data_out_A port is still invalid because there has not been another rising clock edge on CLK_A. The second rising edge of CLK_A reads the new data at the in location (bb), which now contains 4444. This time, the read operating is valid because there has been sufficient setup time between CLK_B and CLK_A.

## Simultaneous Writes to Both Ports with Different Data Conflicts

If both ports write simultaneously into the same memory cell with different data, then the data stored in that cell becomes invalid, as outlined in Table 10.

*Table 10:* **RAM Conflicts During Simultaneous Writes to Same Address**

| Input Signals | | | | | | | | RAM Contents | |
|---|---|---|---|---|---|---|---|---|---|
| Port A | | | | Port B | | | | | |
| WEA | CLKB | DIPA | DIA | WEB | CLKA | DIPB | DIB | Parity | Data |
| 1 | ↑ | DIPA | DIA | 1 | ↑ | DIPB | DIB | ? | ? |

**Notes:**
1. ADDRA=ADDRB, ENA=1,ENB=1, DIPA ≠ DIPB, DIA ≠ DIB, ?=Unknown or invalid data.

## Write Mode Conflicts on Output Latches

Potential conflicts occur when one port writes to memory and the opposite port reads from memory. Write operations always succeed and the write port's output data latches behave as described by the port's WRITE_MODE attribute. If the write port is configured with WRITE_MODE set to NO_CHANGE or WRITE_FIRST, then a write operation to the port invalidates the data output latches on the opposite port, as shown in Table 11.

Using the READ_FIRST mode does not cause conflicts on the opposite port.

*Table 11:* **Conflicts to Output Latches Based on WRITE_MODE**

| Input Signals | | | | | | | | Output Signals | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Port A | | | | Port B | | | | Port A | | Port B | |
| WEA | CLKB | DIPA | DIA | WEB | CLKA | DIPB | DIB | DOPA | DOA | DOPB | DOB |
| **WRITE_MODE_A=NO_CHANGE** | | | | | | | | | | | |
| 1 | ↑ | DIPA | DIA | 0 | ↑ | DIPB | DIB | No Chg | No Chg | ? | ? |
| **WRITE_MODE_B=NO_CHANGE** | | | | | | | | | | | |
| 0 | ↑ | DIPA | DIA | 1 | ↑ | DIPB | DIB | ? | ? | No Chg | No Chg |
| **WRITE_MODE_A=WRITE_FIRST** | | | | | | | | | | | |
| 1 | ↑ | DIPA | DIA | 0 | ↑ | DIPB | DIB | DIPA | DOA | ? | ? |
| **WRITE_MODE_B=WRITE_FIRST** | | | | | | | | | | | |
| 0 | ↑ | DIPA | DIA | 1 | ↑ | DIPB | DIB | ? | ? | DIPB | DIB |
| **WRITE_MODE_A=WRITE_FIRST, WRITE_MODE_B=WRITE_FIRST** | | | | | | | | | | | |
| 0 | ↑ | DIPA | DIA | 1 | ↑ | DIPB | DIB | ? | ? | ? | ? |

**Notes:**
1. ADDRA=ADDRB, ENA=1, ENB=1, ?=Unknown or invalid data

## Conflict Resolution

There is no dedicated monitor to arbitrate the result of identical addresses on both ports. The application must time the two clocks appropriately. However, conflicting simultaneous writes to the same location never cause any physical damage.

# Block RAM Design Entry

Various tools help create Spartan-3 block RAM designs, two of which are the Xilinx CORE Generator system and VHDL or Verilog instantiation of the appropriate Xilinx library primitives.

## Xilinx CORE Generator System

The Xilinx CORE Generator system provides both a Single Port Block Memory and a Dual Port Block Memory module generator, as shown in Figure 5. Both module generators support RAM, ROM, and Write Only functions, according to the control signals that are selected. Any size memory that can be created in the architecture is supported.

Both modules are parameterizable as with most CORE Generator modules. To create a module, specify the component name and choose to include or exclude control inputs, and choose the active polarity for the control inputs. For the Dual-Port Block Memory, once the organization or aspect ratio for Port A is selected, only the valid options for Port B are displayed.

Optionally, specify the initial memory contents. Unless otherwise specified, each memory location initializes to zero. Enter user-specified initial values via a Memory Initialization File, consisting of one line of binary data for every memory location. A default file is generated by the CORE Generator system. Alternatively, create a coefficients file (.coe), which not only defines the initial contents in a radix of 2, 10, or 16, but also defines all the other control parameters for the CORE Generator system.

The output from the CORE Generator system includes a report on the options selected and the device resources required. If a very deep memory is generated, some external multiplexing may be required, and these resources are reported as the number of logic slices required. In addition, the software reports the number of bits available in block RAM that are less than 100% utilized. For simulation purposes, the CORE Generator system creates VHDL or Verilog behavioral models.

- **CORE Generator**: Single-Port Block Memory module (RAM or ROM)
- **CORE Generator**: Dual-Port Block Memory module (RAM or ROM)

## VHDL and Verilog Instantiation

VHDL and Verilog synthesis-based designs can either infer or directly instantiate block RAM, depending on the specific logic synthesis tool used to create the design.

### Inferring Block RAM

Some VHDL and Verilog logic synthesis tools, such as the Xilinx Synthesis Tool (XST) and Synplicity Synplify both infer block RAM based on the hardware described. The Xilinx ISE Project Navigator includes templates for inferring block RAM in your design. To use the templates within Project Navigator, select **Edit → Language Templates** from the menu, and then select **VHDL** or **Verilog**, followed by **Synthesis Templates → RAM** from the selection tree. Finally, select the preferred block RAM template.

It is still possible to directly instantiate block RAM, even if portions of the design infer block RAM.

### Instantiation Templates

For VHDL- and Verilog-based designs, various instantiation templates are available to speed development. Within the Xilinx ISE Project Navigator, select **Edit → Language Templates** from the menu, and then select **VHDL** or **Verilog**, followed by **Component Instantiation → Block RAM** from the selection tree.

The appendices include example code showing how to instantiate block RAM in both VHDL and Verilog.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template must be inserted within the VHDL design file. The port map of the architecture section must include the signal names used in the application.

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.

#### Initialization in VHDL or Verilog Codes

Block RAM memory structures can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the block RAM instantiation and are copied within the EDIF output file 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 VHDL and Verilog examples in the appendices illustrate these techniques.

## Block RAM Applications

Typically, block RAM is used for a variety of local storage applications. However, the following section describes additional, perhaps less obvious block RAM capabilities, illustrating some powerful capabilities to spur the imagination.

### Creating Larger RAM Structures

Block SelectRAM columns have specialized routing to allow cascading blocks with minimal routing delays. Wider or deeper RAM structures incur a small delay penalty.

### Block RAM as Read-Only Memory (ROM)

By tying the write enable input Low, block RAM optionally functions as registered block ROM. The ROM outputs are synchronous and require a clock input and perform exactly like a block RAM read operation. The ROM contents are defined by the initial contents at design time.

After design compilation, the ROM contents can also be updated using the Data2BRAM utility described below.

### FIFOs

First-In, First-Out (FIFO) memories, also known as elastic stores, are perhaps the most common application of block RAM, other than for random data storage. FIFOs typically resynchronize data, either between two different clock domains, or between two parts of a system that have different data rates, even though they operate from a single clock. The Xilinx CORE Generator system provides two parameterizable FIFO modules, one a synchronous FIFO where both the read and write clocks are synchronous to one another and the other an asynchronous FIFO where the read and write clocks are different.

Application note XAPP261 demonstrates that the FIFO read and write ports can be different data widths, integrating the data width converter into the FIFO.

Application note XAPP291 describes a self-addressing FIFO that is useful for throttling data in a continuous data stream.

- **CORE Generator**: **Synchronous FIFO** module
- **CORE Generator**: **Asynchronous FIFO** module
- **XAPP258**: *FIFOs Using Block RAM*, includes reference design
- **XAPP261**: *Data-Width Conversion FIFOs Using Block RAM Memory*, includes reference design
- **XAPP291**: *Self-Addressing FIFO*

### Storage for Embedded Processors

Block RAM also enables efficient embedded processor applications. RAM performs a variety functions in an embedded processor such as those listed below.

- Register file for processor register set, although for some processors, distributed RAM may be a preferred solution.
- Stack or LIFO for stack-based architectures and for call stacks.
- Fast, local code storage. The fast access time to internal block RAM significantly boosts the performance of embedded processors. However, on-chip storage is limited by the number of available block RAMs.
- Large dual-ported mailbox memory shared with external processor or DSP device.
- Temporary trace buffers (see **Circular Buffers, Shift Registers, and Delay Lines**) to ease and enhance application debugging.

## Updating Block RAM/ROM Content by Directly Modifying Device Bitstream

In a typical design flow, the initial contents of block RAM/ROM is defined at design time and compiled into the device bitstream that is downloaded to and configures a Spartan-3 FPGA.

However, for some applications, the actual memory contents may not be known when the bitstream is created or may change later. One example is if a processor embedded with the Spartan-3 FPGA uses block RAM to store program code. To avoid re-compiling the FPGA design just to incorporate a code change, Xilinx provides a utility called Data2BRAM that updates an existing FPGA bitstream with new block RAM/ROM contents.

As shown in Figure 20, the inputs to Data2BRAM include the new RAM contents—typically the output from the embedded processor compiler/linker, the present FPGA bitstream, and a file that describes both the mapping between the system address space and the addressing used on the individual block RAMs and the physical location of each block RAM.

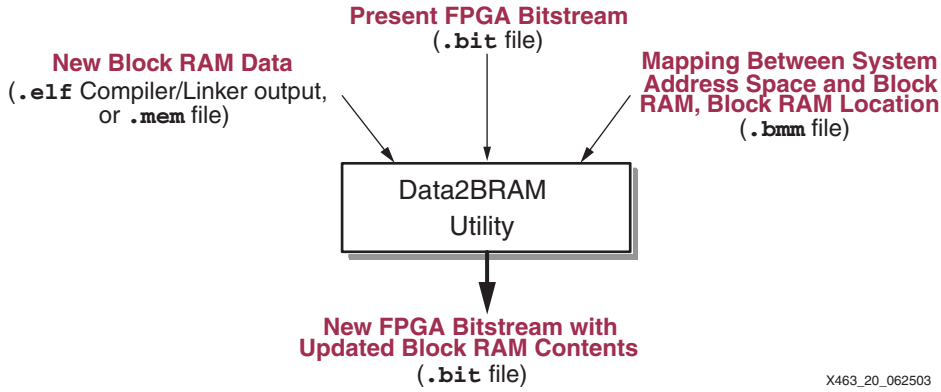



*Figure 20:* **The Data2BRAM Utility Updates Block RAM Contents in a Bitstream**

## Two Independent Single-port RAMs Using One Block RAM

Some applications may require more single-port RAMs than there are RAM blocks on the device. However, a simple trick allows a single block RAM to behave as if it were two, completely independent single-port memories, effectively doubling the number of RAM blocks on the device. The penalty is that each RAM block is only half the size of the original block, up to 9K bits total.

Figure 21 shows how to create two independent single-port RAMs from one block RAM. Tie the most-significant address bit of one port High and the most-significant address bit of the other port Low. Both ports evenly split the available RAM between them.

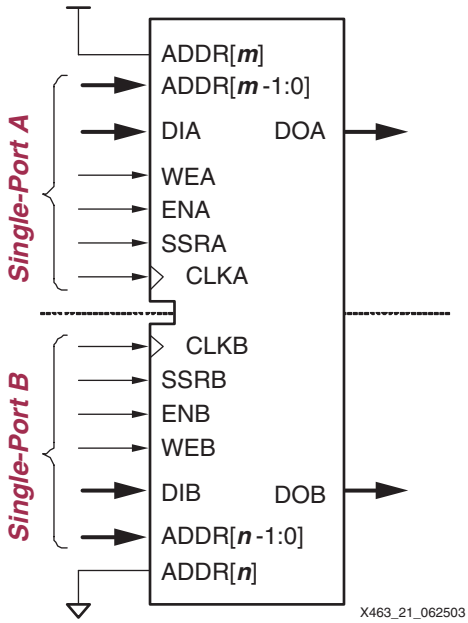



*Figure 21:* **One Block RAM Becomes Two Independent Single-Port RAMs**

Both ports are independent, each with its own memory organization, data inputs and outputs, clock input and control signals. For example, Port A could be 256x36 while Port B is 2Kx4.

Figure 21 splits the available memory evenly between the two ports. With additional logic on the upper address lines, the memory can be split into other ratios.

## A 256x72 Single-Port RAM Using One Block RAM

Figure 22 illustrates how to create a 256-deep by 72-bit wide single-port RAM using a single block RAM. As in the previous example, the memory array is split into halves. One half contains the lower 36 bits and the upper half stores the upper 36 bits, effectively creating a 72-bit wide memory.

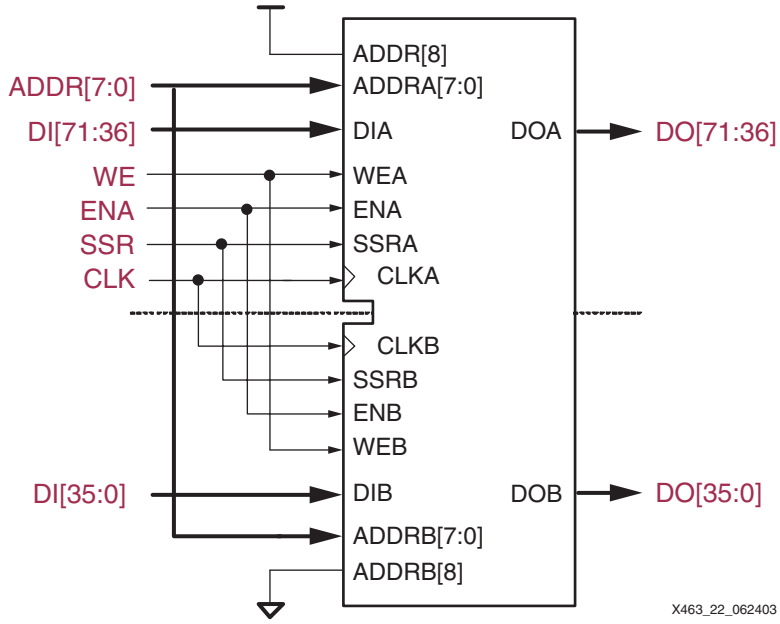



*Figure 22:* **A 256x72 Single-Port RAM Using a Single Block RAM**

The most-significant address line, ADDR[8] is tied High on one port and Low on the other. Both ports share the same the address inputs, control inputs, and clock input.

## Circular Buffers, Shift Registers, and Delay Lines

Circular buffers are used in a variety of digital signal processing applications, such as finite impulse response (FIR) filters, multi-channel filtering, plus correlation and cross-correlation functions. Circular buffers are also useful simply for delaying data to resynchronize it with other parts of a data path.

Figure 23 conceptually describes how a circular buffer operates. Data is written into the buffer. After ***n*** clock cycles, that same data is clocked out of the buffer while new data is written to the same location.

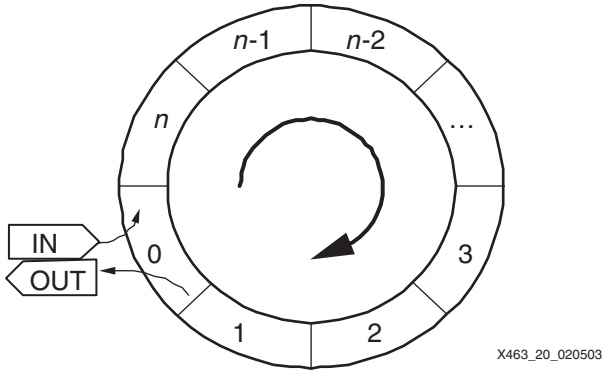



*Figure 23:* **Circular Buffer**

Figure 24 describes the hardware implementation to create a circular buffer using block RAM. A modulo-***n*** counter drives the address inputs to a single-port block RAM. For simple data delay lines, the block RAM writes new data on every clock cycle.

The circular buffer also reads the delayed data value on every clock edge. Using block RAM's READ_FIRST write mode, both the incoming write data and the outgoing read data use the same clock input and the same clock edge, both simplifying the design and improving overall performance. The actual write and read behavior is described in Figure 17.

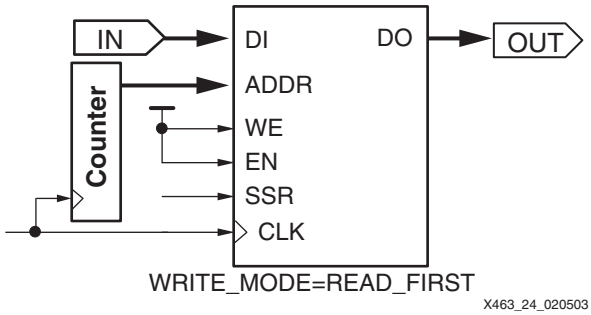



*Figure 24:* **Circular Buffer Implementation Using Block RAM and Counter**

In Figure 24, the width of the IN and OUT data ports is identical, although they do not need be. Using dual-port mode, the ports can be different widths. Figure 25 shows an example where byte-wide data enters the block RAM and a 32-bit word exits the block RAM. Furthermore, the data can be delayed up to 2,048 byte-clock cycles.

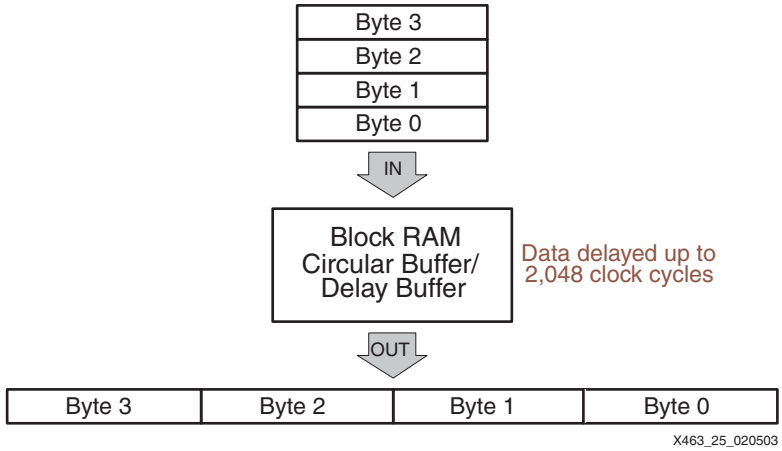



*Figure 25:* **Merge Circular Buffer and Port-Width Converter into a Single Block RAM**

A single block RAM is configured as dual-port memory. The incoming byte-wide data feeds Port B, which is configured as a 2Kx9 memory. The outgoing 32-bit data appears on Port A and consequently, Port A is configured as a 512x36 memory.

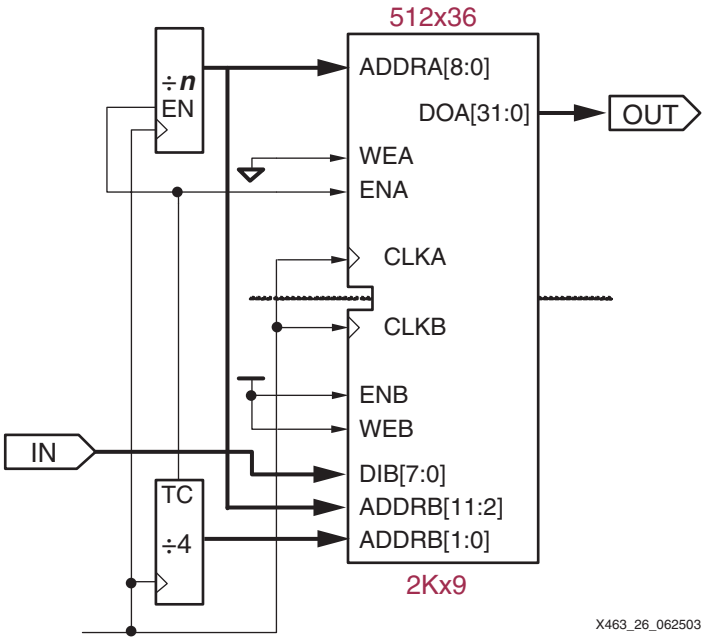



*Figure 26:* **Incoming Byte-Wide Data is Delayed 4n Clock Cycles, Converted to 32-Bit Data**

Manipulating the addresses that feeds both ports creates the 4***n***-byte clock delay. Every 32-bit output word requires four incoming bytes. Consequently, a divide-by-4 counter feeds the two lower address bits, ADDRB[1:0]. After four bytes are stored, a terminal count, TC, from the lower counter enables Port A plus a separate divide-by-***n*** counter. The enable signal latches the 32-bit output data on Port B and increments the upper counter. The combination of the divide-by-4 counter and the divide-by-***n*** counter effectively create a divide-by-4***n*** counter. The output from the divide-by-***n*** counter forms the more-significant address bits to Port B, ADDRB[11:2] and the entire address to Port A, ADDRA[9:0].

## Fast Complex State Machines and Microsequencers

Because block RAMs can be configured with any set of initial values, they also make excellent dual-ported registered ROMs that can be used as state machines. For example, a 128-state, 8-way branch finite state machine with 38 total state outputs, fits in a single block RAM, as shown in Figure 27.

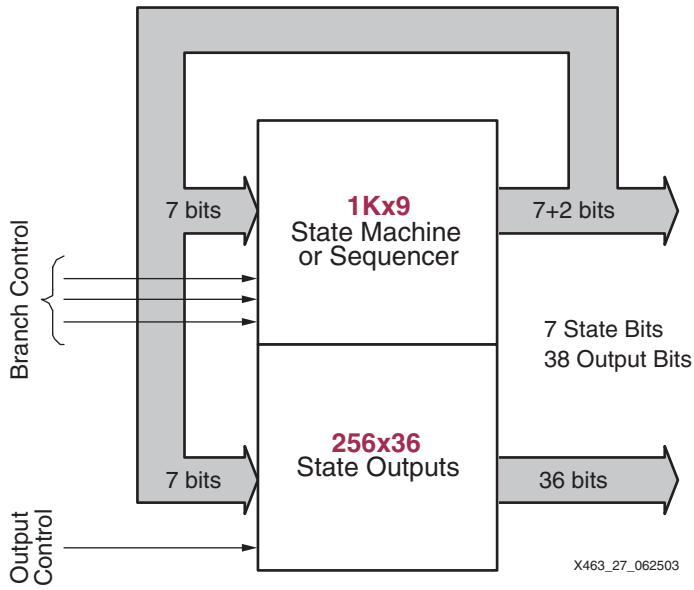



*Figure 27:* **128-State Finite State Machine with 38 Outputs in a Single Block RAM**

A dual-port block RAM memory is divided into two completely independent half-size, single-port memories by tying the most-significant address bit of one port High and the other one Low, similar to Figure 21. Port A is configured as 2Kx9 but used as a 1K x 9 single-port ROM. Seven outputs feed back as address inputs, stepping through the 128 states. The 1Kx9 ROM has ten total address lines, seven of which are the current-state inputs and the remaining three address inputs determine the eight-way branch. Any of the 128 states can conditionally branch to any set of eight new states, under the control of these three address inputs.

Port B is configured as 512 x 36 and used as a 256 x 36 single-port ROM. It receives the same 7-bit current-state value from Port A, and drives 36 outputs that can be arbitrarily defined for each state. However, due to the synchronous nature of block ROM, the 36 outputs from the 256x36 ROM are delayed by one clock cycle. The eighth address input can invoke an alternate definition of the 36 outputs. Two additional state bits are available from the 1Kx9 block, but are not delayed by one clock.

This same basic architecture can be modified to form a 256-state finite state machine with four-way branch, or a 64-state state machine with 16-way branch.

If additional branch-control inputs are needed, they can be combined using an input multiplexer. The advantages of this design are its low cost (a single block RAM), its high performance (125+ MHz), the absence of lay-out or routing issues, and complete design freedom.

## Fast, Long Counters Using RAM

A counter is an example of a simple state machine, where the next state depends only on the current state. A binary up counter, for example, simply increments the current state to create the next state. Figure 28 shows a 20-bit binary up counter, with clock enable and synchronous reset, implemented in a single block RAM.

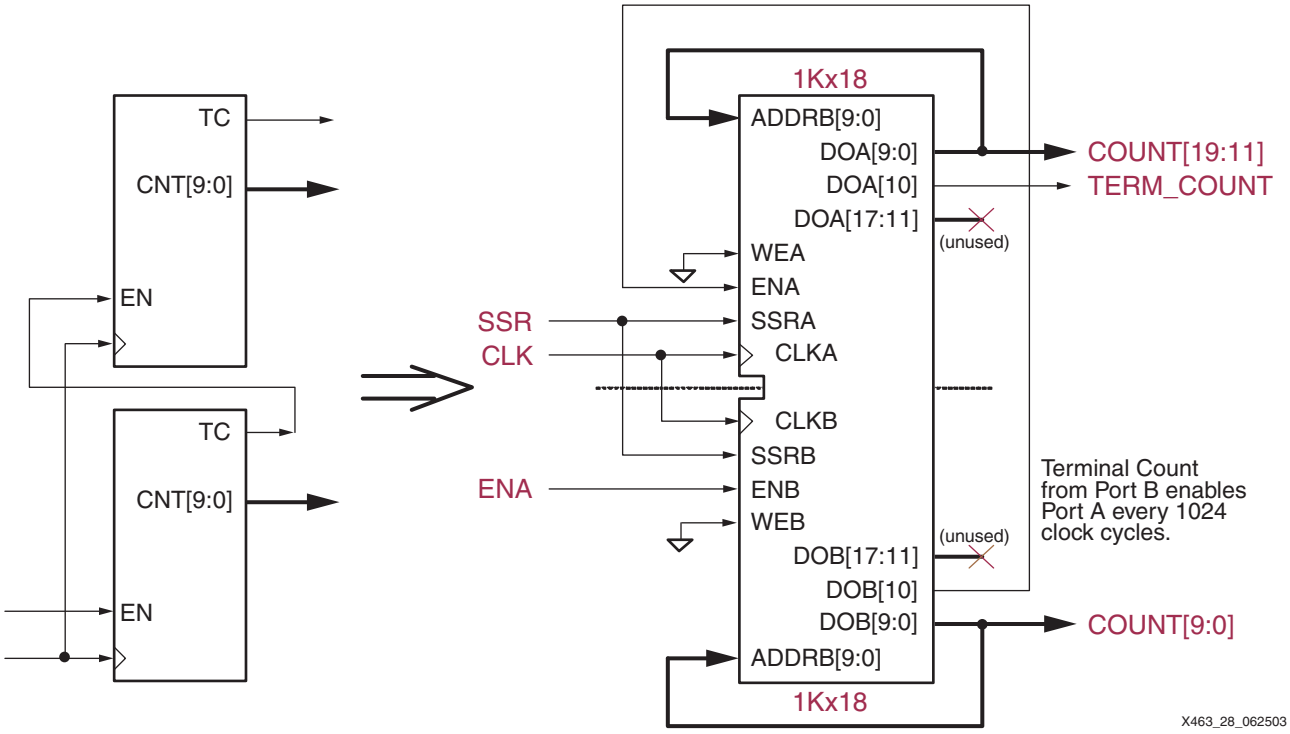


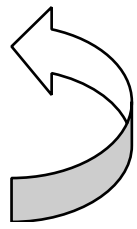

*Figure 28:* **Two 10-Bit Counters Create a 20-Bit Binary Counter Using a Single Block RAM**

A 20-bit binary counter can be constructed from two identical 10-bit binary counters, with the lower 10-bit counter enabling the upper 10-bit counter every 1024 clock cycles. In this example, Port B is a 1Kx18 ROM (WEB is Low) that forms the lower 10-bit counter. The ten less-significant data outputs, representing the current state, connect directly to the ten address inputs, ADDRB[9:0]. The next state is looked up in the ROM using the current state applied to the address pins. The eleventh data bit, D[10], forms the terminal-count output from the counter. In this example, the upper seven data bits, DOB[17:11] are unused.

The next-state logic for a binary counter appears in Table 12. The counter starts at state 0—or the value specified by the INIT or SRVAL attributes—and counts through to 0x3FF (1023 decimal) at which time the terminal count, D[10], is active and the counter rolls over back to 0.

*Table 12:* **Next-State Logic for Binary Up Counter**

| Current State | State Outputs | Next State |
|---|---|---|
| | TC | COUNT |
| ADDR[9:0] (Hex) | D[10] | D[9:0] (Hex) |
| 0 | 0 | 1 |
| 1 | 0 | 2 |
| 2 | 0 | 3 |
| … | … | … |
| 3FFF | 1 | 0 |

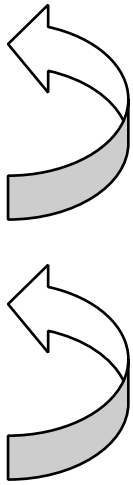


Port A is configured nearly identically to Port B, except that Port A is enabled by the terminal count output from Port B. The 10-bit counter in Port A has the identical counting pattern as Port B, except that it increments at 1/1024$^{th}$ the rate of Port B.

With a simple modification, the 20-bit up counter becomes an 18-bit up/down counter. Using the most-significant address input as a direction control, the same basic counter architecture either increments or decrements its count, as shown in Table 13. In this example, the counter increments when the Up/Down control is Low and decrements when High. The ROM memory is split between the incrementing and decrementing next-state logic.

*Table 13:* **Next-State Logic for Binary Up/Down Counter**

| Up/Down Control | Present State | State Outputs | Next State |
|---|---|---|---|
| | | TC | COUNT |
| ADDR[9] | ADDR[8:0] (Hex) | D[10] | D[9:0] (Hex) |
| 0 (Up) | 0 | 0 | 1 |
| | 1 | 0 | 2 |
| | 2 | 0 | 3 |
| | … | … | … |
| | 1FFF | 1 | 0 |
| | | | |
| 1 (Down) | 1FFF | 0 | 1FFE |
| | 1FFE | 0 | 1FFD |
| | 1FFD | 0 | 1FFC |
| | … | … | … |
| | 0 | 1 | 1FFF |



Various other counter implementations are possible including the following.

- Binary up and up/down counters of various modulos determined by the combinations of the modulos of the counters implemented in Port A and Port B.
- Counters with other incrementing and decrementing patterns including fast gray-code counters.
- A six-digit BCD counter in one block ROM, configured as 512x36, plus one CLB.

## Four-Port Memory

Each block RAM is physically a dual-port memory. However, due to the block RAM's fast access performance, it is possible to create multi-port memories by time-division multiplexing the signals in and out of the memory. A block RAM with some additional logic easily supports up to four ports but at the cost of additional access latency for each port. The following application note provides additional details and a reference design.

- **XAPP228**: *Quad-Port Memories in Virtex Devices*, includes reference design

## Content-Addressable Memory (CAM)

Content-Addressable Memory (CAM), sometimes known as associative memory, is used in a variety of networking and data processing applications. In most memory applications, content is referenced by an address. In CAM applications, the content is the driving input and the output indicates whether or not the content exists in memory and, if so, provides a reference to its location.

An easy way to envision how a CAM operates is to think of an index to a book. Looking up an item, *i.e.*, the content, first determines whether the item exists in the index and if it does, provides a reference to its location, *i.e.*, the page number of where the item can be found.

- **CORE Generator**: Content-Addressable Memory module
- XAPP260: *Using Block RAM for High-Performance Read/Write CAMs*
- XAPP201: *An Overview of Multiple CAM Designs*, written for Virtex/E and Spartan-II/E architectures but provides a useful overview to the techniques involved

## Implementing Logic Functions Using Block RAM

Inside every Spartan-3 logic cell, there is a four-input RAM/ROM called a look-up table or LUT. The LUT performs any possible logic function of its four inputs and forms the basis of the Spartan-3 logic architecture.

Another possible application for block RAM is as a much larger look-up table. In one of its organizations, a block RAM—used as ROM in this case—has 14 inputs and a single output. Consequently, block RAM is capable of implementing any possible arbitrary logic function of up to 14 inputs, regardless of the complexity and regardless of inversions. There are a few restrictions, however.

- There cannot be any asynchronous feedback paths in the logic, such as those that create latches.
- The logic output must be synchronized to a clock input. Block RAM does not support asynchronous read outputs.

If the logic function meets these requirements, then a single block RAM implements the following functions.

- Any possible Boolean logic function of up to 14 inputs
- Nine separate arbitrary Boolean logic functions of 11 inputs, as long as the inputs are shared.
- Various other combinations are possible, but may have restrictions to the number of inputs, the number of shared inputs, or the complexity of the logic function.

Due to the flexibility and speed of CLB logic, block RAM may not be faster or more efficient for simple wide functions like an address decoder, where multiple inputs are ANDed together. Block RAM will be faster and more efficient for complex logic functions, such as majority decoders, pattern matching, correlators.

### Fuzzy Pattern Matching Circuit Example

For example, Figure 29 illustrates a fuzzy pattern matching circuit that detects both exact matches and those patterns that are close enough. Each incoming bit is matched against the required MATCH pattern. Then, any “don’t care” bits are masked off, indicating that the specific bit should always match. Then, the number of matching bits is counted and compared against an activation threshold. If the number of matching bits is greater than the activation threshold, then the input data mostly matches the required pattern and the MATCH output goes High.

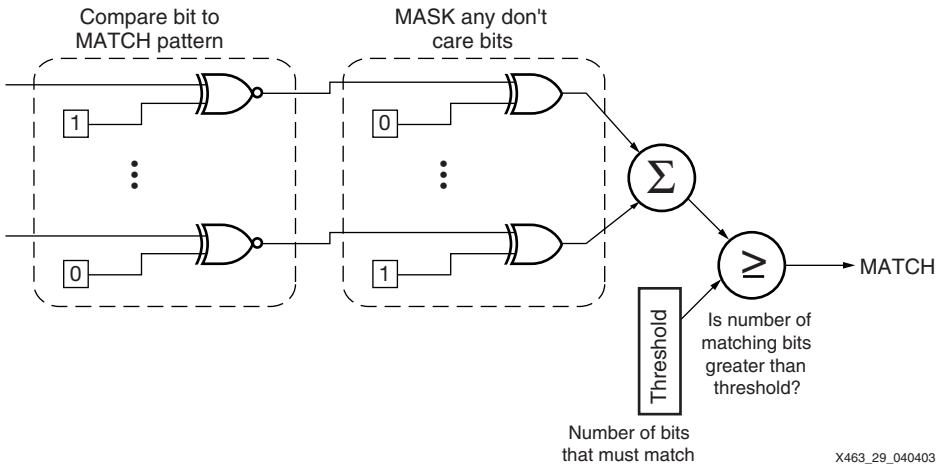



*Figure 29:* **A 14-Input Fuzzy Pattern Matching Circuit Implemented in a Single Block RAM**

If the application requires a new matching pattern or different logic function, it could be loaded via the second memory port.

Implemented in CLB logic, this function would require numerous logic cells and multiple layers of logic. However, because the MATCH, MASK, and Threshold values are known in advance, the function can be pre-computed and then stored in block RAM. For each input condition, *i.e.*, starting at address 0 and incremented through the entire memory, the output condition can be pre-computed. A 14-input fuzzy pattern matching circuit requires a single block RAM and performs the operation in a single clock cycle.

### Mapping Logic into Block RAM Using `MAP -bp` Option

The Xilinx ISE software does not automatically attempt to map logic functions into block RAM. However, there is a mapping option to aid the process.

The block RAM mapping option is enabled when using the MAP –bp option. If so enabled, the Xilinx ISE logic mapping software attempts to place LUTs and attached flip-flops into an unused single-output, single-port block RAM. The final flip-flop output is required as block RAMs have a synchronous, registered output. The mapping software packs the flip-flop with whatever LUT logic is driving it. No register will be packed into block RAM without LUT logic, and *vice versa*.

To specify which register outputs will be converted to block RAM outputs, create a file containing a list of the net names connected to the register output(s). Set the environment variable `XIL_MAP_BRAM_FILE` to the file name, which instructs the mapping software to use this file. The MAP program looks for this environment variable whenever the –bp option is specified. Only those output nets listed in the file are converted into block RAM outputs.

- **PCs**:
  `set XIL_MAP_BRAM_FILE=file_name`
- **Workstations**:
  `setenv XIL_MAP_BRAM_FILE file_name`

## Waveform Storage, Function Tables, Direct Digital Synthesis (DDS) Using Block RAM

Another powerful block RAM application is waveform storage, including function tables such as trigonometric functions like sine and cosine. Sine and cosine form the backbone of other functions such as direct digital synthesis (DDS) to generate output waveforms. The Xilinx CORE Generator system provides parameterizable modules for both:

- **CORE Generator**: **Sine/Cosine Look-Up Table** module
- **CORE Generator**: **Direct Digital Synthesizer (DDS)** module

Another potential application of waveform storage is in various signal companders (compressors/expanders) and normalization circuits used to boost important parts of a signal within the available bandwidth. Examples include converters between linear data, u-Law encoded data, and A-Law encoded data commonly used in telecommunications.

The dual-port nature of block RAM not only facilitates waveform storage, it also enables an application to update the waveform, either with a completely new waveform or with corrected or normalized waveform data. In the example shown in Figure 30, Port A initially contains the currently active waveform. The application can load a new waveform on Port B.

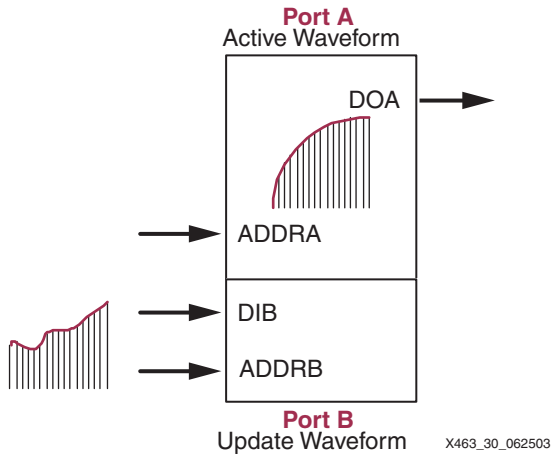



*Figure 30:* **Dual-Port Block RAM Facilitates Waveform Storage and Updates**

As in real-world engineering, sometimes it is faster to look up an answer than deriving it. The same is true in digital designs. Block RAM is also useful for storing pre-computed function tables where the output, y, is a function of the input, x, or y=f(x).

For example, instead of creating the CLB logic that implements the following polynomial equation, the function can be pre-computed and stored in a block RAM.

$$Y = Ax^3 - Bx^2 + Cx + D$$

The values A, B, C, and D are all constants. The output, y, depends only on the input, x. The output value can be pre-computed for each input value of x and stored in memory. There are obvious limitations as the function may not fit in a single logic block either because of the range of values for x, or the magnitude of the output, y. For example, a 512x36 block ROM implements the above equation for input values between 0 and 511. The range of x is limited by its exponential effect on y. With x at its maximum value for this specific example, y requires at least 28 output bits.

Some other look-up functions possible in a single block RAM/ROM include the following.

- Various complex arithmetic functions of a single input, including mixtures of functions such as log(x), square-root(x). Multipliers of two values are possible but are typically limited by the number of block RAM inputs. The Spartan-3 embedded 18x18 multipliers are a better solution for pure multiplication functions.
- Two independent 11-bit binary to 4-digit BCD converters, with the block ROM configured as 1Kx18. The least-significant bit (LSB) of each converter bypasses the ROM as the converted result is the same as the original value, *i.e.* the LSB indicates whether the value is odd or even.
- Two independent 3-digit BCD to 10-bit binary converters, with the block ROM configured as 2Kx9 and the LSBs bypass the converters.
- Sine-cosine look-up tables using one port for sine, the other one for cosine, with 90 degree-shifted addresses, 18-bit amplitude, 10-bit angular resolution.
- Two independent 10-bit binary to three-digit, seven-segment LED output converter with the block ROM configured as 1Kx18. Leading zeros are displayed as blanks. Because input values are limited to 1023, the LED digits display from “0” to “3FF”. Consequently, the logic for the most-significant digit requires only four inputs (segment a=d=g, segment f is always High).

## Related Materials and References

- "**Using Leftover Multipliers and Block RAM in Your Design**" by Peter Alfke, Xilinx, Inc.
  http://www.xilinx.com/support/techxclusives/leftover-techX11.htm
- "**The Myriad Uses of Block RAM**" by Jan Gray, Gray Research, LLC.
  http://www.fpgacpu.org/usenet/bb.html
- **Libraries Guide**, for Xilinx ISE 5.2i by Xilinx, Inc.
  - *Adobe Acrobat [PDF]*
    http://toolbox.xilinx.com/docsan/xilinx5/pdf/docs/lib/lib.pdf, pp. 1593–1640.

    If ISE 5.2i is installed in the default directory, this document is also located in the following path or within Project Navigator by selecting **Help→Online Documentation**. When the Acrobat document appears, click **Libraries Guide** from the table of contents on the left.
    `C:\Xilinx\doc\usenglish\docs\lib\lib.pdf`
  - `RAMB16_Sn` *Primitive [HTML]*
    http://toolbox.xilinx.com/docsan/xilinx5/data/docs/lib/lib0371_355.html
  - `RAMB16_Sm_Sn` *Primitive [HTML]*
    http://toolbox.xilinx.com/docsan/xilinx5/data/docs/lib/lib0372_356.html

## Conclusion

The Spartan-3 FPGA's abundant, fast, and flexible block RAMs provide invaluable on-chip local storage for scratchpad memories, FIFOs, buffers, look-up tables, and much more. Using unique capabilities, block RAM implements such functions as shift registers, delay lines, counters, and wide, complex logic functions.

Block RAM is supported in applications using the broad spectrum of Xilinx ISE development software, including the CORE Generator system and can be inferred or directly instantiated in VHDL or Verilog synthesis designs.

## Appendix A: VHDL Instantiation Example

The following VHDL instantiation example is for the Synopsys FPGA Express system. The example XC3S_RAMB_1_PORT module uses the **SelectRAM_A36.vhd** VHDL template. This and other templates are available for download from the following Web link. The following example is a VHDL code snippet and will not compile as is.

- ftp://ftp.xilinx.com/pub/applications/xapp/xapp463_vhdl.zip

```
-- Module: XC3S_RAMB_1_PORT
-- Description: 18Kb Block SelectRAM example
-- Single Port 512 x 36 bits
-- Use template "SelectRAM_A36.vhd"
--
-- Device: Spartan-3 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 XC3S_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 XC3S_RAMB_1_PORT;
--
architecture XC3S_RAMB_1_PORT_arch of XC3S_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";
-- Initialize parity memory content
INITP_00 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
INITP_01 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
... (snip)
INITP_07 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
-- Initialize data memory content
INIT_00 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
INIT_01 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
... (snip)
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;
-- Parity memory initialization attributes
attribute INITP_00: string;
attribute INITP_01: string;
... (snip)
attribute INITP_07: string;
-- Data memory initialization attributes
attribute INIT_00: string;
attribute INIT_01: string;
```

```
... (snip)
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";
... (snip)
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";
... (snip)
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 XC3S_RAMB_1_PORT_arch;
```

# Appendix B: Verilog Instantiation Example

The following Verilog instantiation example is for the Synopsys FPGA Express system. The example XC3S_RAMB_1_PORT module uses the **SelectRAM_A36.v** Verilog template. This and other templates are available for download from the following Web link. The following example is a Verilog code snippet and will not compile as is.

- ftp://ftp.xilinx.com/pub/applications/xapp/xapp463_verilog.zip

```
// Module: XC3S_RAMB_1_PORT
// Description: 18Kb Block SelectRAM-II example
// Single Port 512 x 36 bits
// Use template "SelectRAM_A36.v"
//
// Device: Spartan-3 Family
//-----------------------------------------------------------------
module XC3S_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,
//Initialize parity memory content
U_RAMB16_S36.INITP_00 =
256'h0123456789ABCDEF0000000000000000000000000000000000000000000000000,
U_RAMB16_S36.INITP_01 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
... (snip)
U_RAMB16_S36.INITP_07 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
//Initialize data memory content
U_RAMB16_S36.INIT_00 =
256'h0123456789ABCDEF0000000000000000000000000000000000000000000000000,
U_RAMB16_S36.INIT_01 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
... (snip)
U_RAMB16_S36.INIT_3F =
256'h0000000000000000000000000000000000000000000000000000000000000000;
// 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
"0000000000000000000000000000000000000000000000000000000000000000000000"
... (snip)
INITP_07
"0000000000000000000000000000000000000000000000000000000000000000000000"
INIT_00
"0123456789ABCDEF0000000000000000000000000000000000000000000000000000"
INIT_01
"0000000000000000000000000000000000000000000000000000000000000000000000"
... (snip)
INIT_3F
"0000000000000000000000000000000000000000000000000000000000000000000000"
*/
endmodule
```

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 04/05/03 | 1.0 | Initial Xilinx release. |
| 05/12/03 | 1.1 | Updated block RAM information for the XC3S50 device.<br>Updated Table 1, Figure 2, Figure 18, Figure 26, and the text associated with the **Introduction** and **Block RAM Location and Surrounding Neighborhood** sections.<br>Added 256x72 single-port mode to Table 2.<br>Updated hypertext links in **Related Materials and References**, **Appendix A: VHDL Instantiation Example**, and **Appendix B: Verilog Instantiation Example**.<br>Corrected Figure 22. DIB and ADDRB connections were swapped. |
| 06/25/03 | 1.1.1 | Minor editing changes. |
| 07/23/03 | 1.1.2 | Improved display quality for Figure 5. |

XAPP464 (v1.0) July 8, 2003

# Using Look-Up Tables as Distributed RAM in Spartan-3 FPGAs


## Summary

Each Spartan™-3 Configurable Logic Block (CLB) contains up to 64 bits of single-port RAM or 32 bits of dual-port RAM. This RAM is distributed throughout the FPGA and is commonly called “distributed RAM” to distinguish it from block RAM. Distributed RAM is fast, localized, and ideal for small data buffers, FIFOs, or register files. This application note describes the features and capabilities of distributed RAM and illustrates how to specify the various options using the Xilinx CORE Generator™ system or via VHDL or Verilog instantiation.


## Introduction

In addition to the embedded 18Kbit block RAMs, Spartan-3 FPGAs feature distributed RAM within each Configurable Logic Block (CLB). Each SLICEM function generator or LUT within a CLB resource optionally implements a 16-deep x 1-bit synchronous RAM. The LUTs within a SLICEL slice do not have distributed RAM.

Distributed RAM writes synchronously and reads asynchronously. However, if required by the application, use the register associated with each LUT to implement a synchronous read function. Each 16 x 1-bit RAM is cascadable for deeper and/or wider memory applications, with a minimal timing penalty incurred through specialized logic resources.

Spartan-3 CLBs support various RAM primitives up to 64-deep by 1-bit-wide. Two LUTs within a SLICEM slice combine to create a dual-port 16x1 RAM—one LUT with a read/write port, and a second LUT with a read-only port. One port writes into both 16x1 LUT RAMs simultaneously, but the second port reads independently.

Distributed RAM is crucial to many high-performance applications that require relatively small embedded RAM blocks, such as FIFOs or small register files. The Xilinx CORE Generator™ software automatically generates optimized distributed RAMs for the Spartan-3 architecture. Similarly, CORE Generator creates Asynchronous and Synchronous FIFOs using distributed RAMs.

### Single-Port and Dual-Port RAMs

#### Data Flow

Distributed RAM supports the following memory types:

- Single-port RAM with synchronous write and asynchronous read. Synchronous reads are possible using the flip-flop associated with distributed RAM.
- Dual-port RAM with one synchronous write and two asynchronous read ports. As above, synchronous reads are possible.

As illustrated in Figure 1, dual-port distributed RAM has one read/write port and an independent read port.

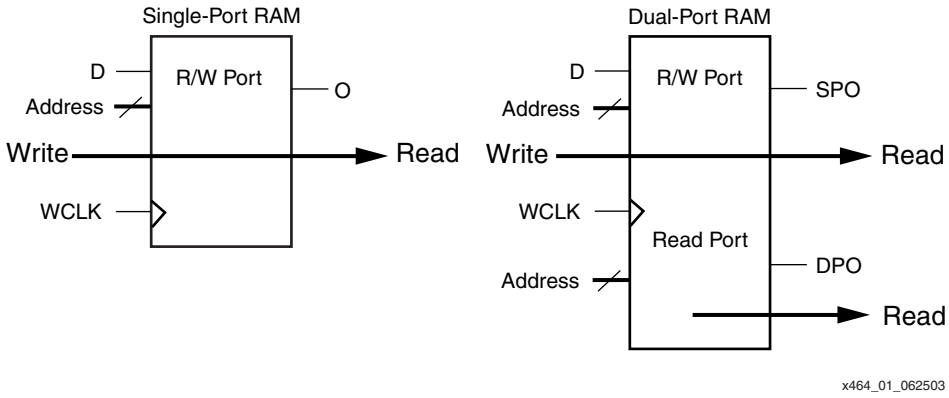



*Figure 1:* **Single-Port and Dual-Port Distributed RAM**

Any write operation on the D input and any read operation on the SPO output can occur simultaneously with and independently from a read operation on the second read-only port, DPO.

## Write Operations

The write operation is a single clock-edge operation, controlled by the write-enable input, WE. By default, WE is active High, although it can be inverted within the distributed RAM. When the write enable is High, the clock edge latches the write address and writes the data on the D input into the selected RAM location.

When the write enable is Low, no data is written into the RAM.

## Read Operation

A read operation is purely combinatorial. The address port—either for single- or dual-port modes—is asynchronous with an access time equivalent to a LUT logic delay.

## Read During Write

When synchronously writing new data, the output reflects the data being written to the addressed memory cell, which is similar to the WRITE_MODE=WRITE_FIRST mode on the Spartan-3 block RAMs. The timing diagram in Figure 2 illustrates a write operation with the previous data read on the output port, before the clock edge, followed by the new data.

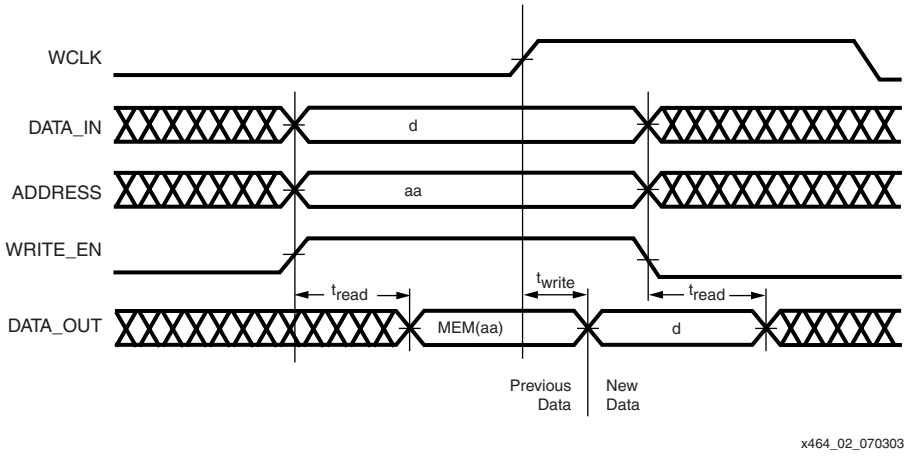



*Figure 2:* **Write Timing Diagram**

## 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 LUT 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, the A[#:0] port is the write and read address, and the DPRA[#:0] port is an independent read-only address.

## Compatibility with Other Xilinx FPGA Families

Each Spartan-3 distributed RAM operates identically to the distributed RAM found in Virtex™, Virtex-E, Spartan-II, Spartan-IIE, Virtex-II, and Virtex-II Pro FPGAs.

Table 1 shows the basic memory capabilities embedded within the CLBs on various Xilinx FPGA families. Like Virtex-II/Pro FPGAs, Spartan-3 CLBs have eight LUTs and implement 128 bits of ROM memory. Like the Virtex/E and Spartan-II/IIE FPGAs, Spartan-3 CLBs have 64 bits of distributed RAM. Although the Spartan-3 and Virtex-II/Pro CLBs are identical for logic functions, the Spartan-3 CLBs have half the amount of distributed RAM within each CLB.

*Table 1:* **Distributed Memory Features by FPGA Family**

| Feature | Spartan-3 Family | Virtex/Virtex-E, Spartan-II/Spartan-IIE Families | Virtex-II, Virtex-II Pro Families |
|---|---|---|---|
| LUTs per CLB | 8 | 4 | 8 |
| ROM bits per CLB | 128 | 64 | 128 |
| Single-port RAM bits per CLB | 64 | 64 | 128 |
| Dual-port RAM bits per CLB | 32 | 32 | 64 |

Table 2 lists the various single- and dual-port distributed RAM primitives supported by the different Xilinx FPGA families. For each type of RAM, the table indicates how many instances of a particular primitive fit within a single CLB. For example, two 32x1 single-port RAM primitives fit in a single Spartan-3 CLB. Similarly, two 16x1 dual-port RAM primitives fit in a Spartan-3 CLB but a single 32x1 dual-port RAM primitive does not.

*Table 2:* **Single- and Dual-port RAM Primitives Supported in a CLB by Family**

| Family | Single-Port RAM | | | | Dual-Port RAM | | |
|---|---|---|---|---|---|---|---|
| | 16x1 | 32x1 | 64x1 | 128x1 | 16x1 | 32x1 | 64x1 |
| Spartan-3 | 4 | 2 | 1 | | 2 | | |
| Spartan-II/IIE Virtex/E | 4 | 2 | 1 | | 2 | | |
| Virtex-II/Pro | 8 | 4 | 2 | 1 | 4 | 2 | 1 |

## Library Primitives

There are four library primitives that support Spartan-3 distributed RAM, ranging from 16 bits deep to 64 bits deep. All the primitives are one bit wide. Three primitives are single-port RAMs and one primitive is dual-port RAM, as shown in Table 3.

*Table 3:* **Single-Port and Dual-Port Distributed RAMs**

| Primitive | RAM Size (Depth x Width) | Type | Address Inputs |
|---|---|---|---|
| RAM16X1S | 16 x 1 | Single-port | A3, A2, A1, A0 |
| RAM32X1S | 32 x 1 | Single-port | A4, A3, A2, A1, A0 |
| RAM64X1S | 64 x 1 | Single-port | A5, A4, A3, A2, A1, A0 |
| RAM16X1D | 16 x 1 | Dual-port | A3, A2, A1, A0 |

The input and output data are one bit wide. However, several distributed RAMs, connected in parallel, easily implement wider memory functions.

Figure 3 shows generic single-port and dual-port distributed RAM primitives. The A[#:0] and DPRA[#:0] signals are address buses.

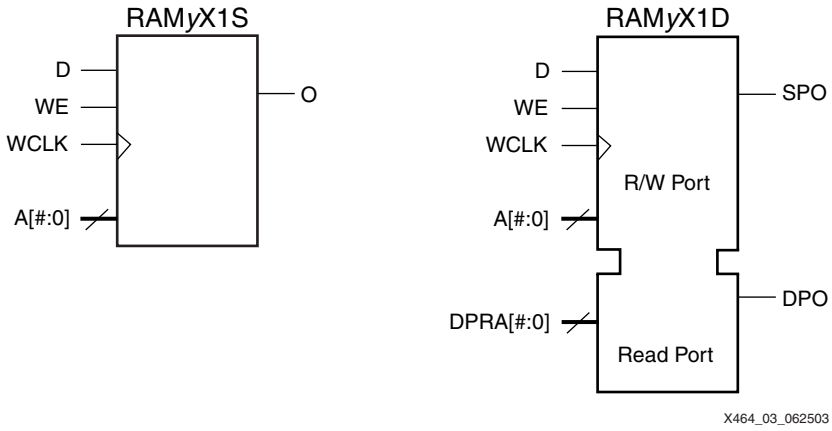



*Figure 3:* **Single-Port and Dual-Port Distributed RAM Primitives**

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

*Table 4:* **Wider Library Primitives**

| Primitive | RAM Size (Depth x Width) | Data Inputs | Address Inputs | Data Outputs |
|---|---|---|---|---|
| RAM16x2S | 16 x 2 | D1, D0 | A3, A2, A1, A0 | O1, O0 |
| RAM32X2S | 32 x 2 | D1, D0 | A4, A3, A2, A1, A0 | O1, O0 |
| RAM16X4S | 16 x 4 | D3, D2, D1, D0 | A3, A2, A1, A0 | O3, O2, O1, O0 |

## Signal Ports

Each distributed RAM port operates independently of the other while reading the same set of memory cells.

### Clock — WCLK

The clock is used for synchronous writes. The data and the address input pins have setup times referenced to the WCLK pin.

### Enable — WE

The enable pin affects the write functionality of the port. An inactive Write Enable prevents any writing to memory cells. An active Write Enable causes the clock edge to write the data input signal to the memory location pointed to by the address inputs.

### Address — A0, A1, A2, A3 (A4, A5)

The address inputs select the memory cells for read or write. The width of the port determines the required address inputs.

***Note:*** 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 output O on single-port RAM or the SPO and DPO outputs on dual-port RAM 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 logic level 0 (negative edge for the clock) or at logic level 1 (positive edge for the clock) without requiring other logic resources.

### Global Set/Reset — GSR

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

### Global Write Enable — GWE

The global write enable signal, GWE, is asserted automatically at the end of device configuration to enable all writable elements. The GWE signal guarantees that the initialized distributed-RAM contents are not disturbed during the configuration process.

Because GWE is a global signal and automatically connected throughout the device, the distributed RAM primitive does not have a GWE input pin.

# Attributes

## Content Initialization — INIT

By default, distributed RAM is initialized with all zeros during the device configuration sequence. To specify [non-zero] initial memory contents after configuration, use the INIT attributes. Each INIT is a hexadecimal-encoded bit vector, arranged from most-significant to least-significant bit. In other words, the right-most hexadecimal character represents RAM locations 3, 2, 1, and 0. Table 5 shows the length of the INIT attribute for each primitive.

*Table 5:* **INIT Attributes Length**

| Primitive | Template | INIT Attribute Length |
|---|---|---|
| RAM16X1S | RAM_16S | 4 digits |
| RAM32X1S | RAM_32S | 8 digits |
| RAM64X1S | RAM_64S | 16 digits |
| RAM16X1D | RAM_16D | 4 digits |

## Placement Location — LOC

Each Spartan-3 CLB contains four slices, each with its own location coordinate, as shown in Figure 4. Distributed RAM fits only in SLICEMs slices. The 'M' in SLICEM indicates that the slice supports memory-related functions and distinguishes SLICEMs from SLICELs. The 'L' indicates that the slice supports logic only.

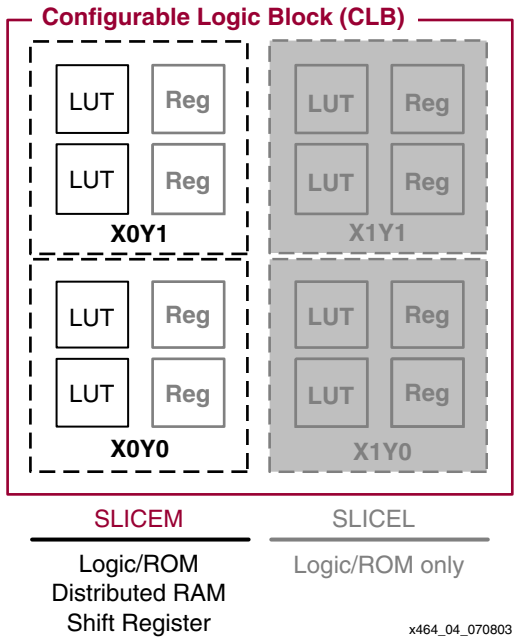



*Figure 4:* **SLICEM slices within Spartan-3 CLB**

When a LOC property is assigned to a distributed RAM instance, the Xilinx ISE software places the instance in the specified location. Figure 4 shows the X,Y coordinates for the slices in a Spartan-3 CLB. Again, only SLICEM slices support memory.

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

For example, the single-port RAM16X1S primitive fits in any LUT within any SLICEM. To place the instance `U_RAM16` in slice X0Y0, use the following LOC assignment:

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

The 16x1 dual-port RAM16X1D primitive requires both 16x1 LUT RAMs within a single SLICEM slice, as shown in Figure 5. The first 16x1 LUT RAM, with output SPO, implements the read/write port controlled by address A[3:0] for read and write. The second LUT RAM implements the independent read-only port controlled by address DPRA[3:0]. Data is presented simultaneously to both LUT RAMs, again controlled by address A[3:0], WE, and WCLK.

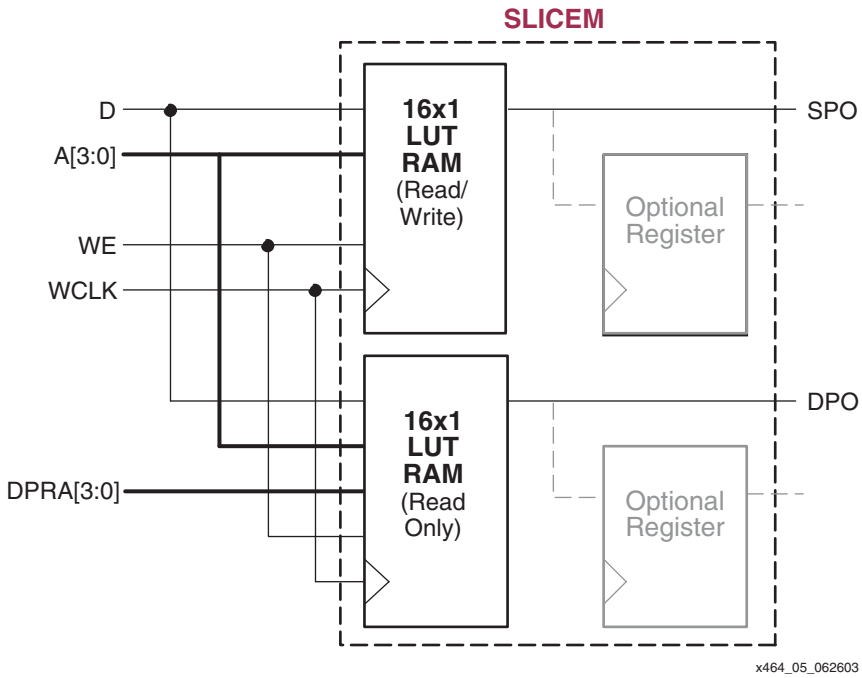



*Figure 5:* **RAM16X1D Placement**

A 32x1 single-port RAM32X1S primitive fits in one slice, as shown in Figure 6. The 32 bits of RAM are split between two 16x1 LUT RAMs within the SLICEM slice. The A4 address line selects the active LUT RAM via the F5MUX multiplexer within the slice.

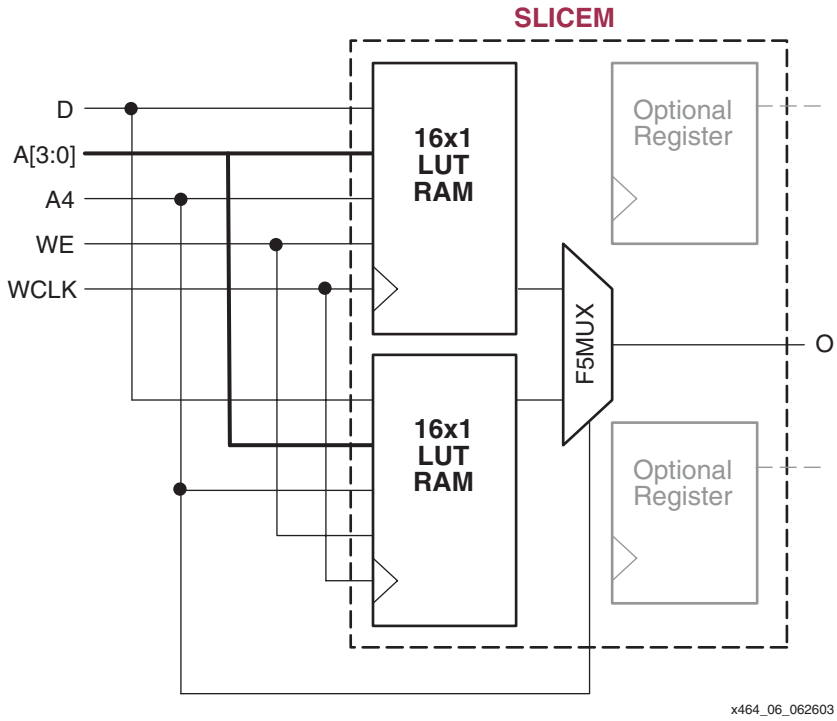



*Figure 6:* **RAM32X1S Placement**

The 64x1 single-port RAM64X1S primitive occupies both SLICEM slices in the CLB. The read path uses both F5MUX and F6MUX multiplexers within the CLB.

## Distributed RAM Design Entry

To specify distributed RAM in an application, use one of the various design entry tools, including the Xilinx CORE Generator software or VHDL or Verilog.

### Xilinx CORE Generator System

The Xilinx CORE Generator system creates distributed memory designs for both single-port and dual-port RAMs, ROMs, and even SRL16 shift-register functions.

The Distributed Memory module is parameterizable. To create a module, specify the component name and choose to include or exclude control inputs, then choose the active polarity for the control inputs.

Optionally, specify the initial memory contents. Unless otherwise specified, each memory location initializes to zero. Enter user-specified initial values via a Memory Initialization File, consisting of one line of binary data for every memory location. A default file is generated by the CORE Generator system. Alternatively, create a coefficients file (`.coe`) as shown in Figure 7, which not only defines the initial contents in a radix of 2, 10, or 16, but also defines all the other control parameters for the CORE Generator system.

```
memory_initialization_radix=16;
memory_initialization_vector= 80, 0F, 00, 0B, 00, 0C, …, 81;
```

*Figure 7:* **A Simple Coefficients File (`.coe`) Example for a Byte-Wide Memory**

The output from the CORE Generator system includes a report on the options selected and the device resources required. If a very deep memory is generated, then some external multiplexing may be required; these resources are reported as the number of logic slices required. For simulation purposes, the CORE Generator system creates VHDL or Verilog behavioral models.

The CORE Generator synchronous and asynchronous FIFO modules support both distributed and block RAMs.

- **CORE Generator**: Distributed Memory module
  http://www.xilinx.com/ipcenter/catalog/logicore/docs/dist_mem.pdf
- **CORE Generator**: Synchronous FIFO module
  http://www.xilinx.com/ipcenter/catalog/logicore/docs/sync_fifo.pdf
- **CORE Generator**: Asynchronous FIFO module
  http://www.xilinx.com/ipcenter/catalog/logicore/docs/async_fifo.pdf

## VHDL and Verilog

VHDL and Verilog synthesis-based designs can either infer or directly instantiate block RAM, depending on the specific logic synthesis tool used to create the design.

### Inferring Block RAM

Some VHDL and Verilog logic synthesis tools, such as the Xilinx Synthesis Tool (XST) and Synplicity Synplify, infer block RAM based on the hardware described. The Xilinx ISE Project Navigator includes templates for inferring block RAM in your design. To use the templates within Project Navigator, select **Edit → Language Templates** from the menu, and then select **VHDL** or **Verilog**, followed by **Synthesis Templates → RAM** from the selection tree. Finally, select the preferred distributed RAM template. Cut and paste the template into the source code for the application and modify it as appropriate.

It is still possible to directly instantiate distributed RAM, even if portions of the design infer distributed RAM.

### Instantiation Templates

For VHDL- and Verilog-based designs, various instantiation templates are available to speed development. Within the Xilinx ISE Project Navigator, select **Edit → Language Templates** from the menu, and then select **VHDL** or **Verilog**, followed by **Component Instantiation → Distributed RAM** from the selection tree. Cut and paste the template into the source code for the application and modify it as appropriate.

There are also downloadable VHDL and Verilog templates available for all single-port and dual-port primitives. The RAM_*x*S templates (where $x$ = 16, 32, or 64) are single-port modules and instantiate the corresponding RAM*x*X1S primitive. The ‘S’ indicates single-port RAM. The RAM_16D template is a dual-port module and instantiates the corresponding RAM16X1D primitive. The ‘D’ indicates dual-port RAM.

- VHDL Distributed RAM Templates
  ftp://ftp.xilinx.com/pub/applications/xapp/xapp464_vhdl.zip
- Verilog Distributed RAM Templates
  ftp://ftp.xilinx.com/pub/applications/xapp/xapp464_verilog.zip

The following are single-port templates:

- RAM_16S
- RAM_32S
- RAM_64S

The following is a dual-port template:

- RAM_16D

In VHDL, each template has a component declaration section and an architecture section. Insert both sections of the template within the VHDL design file. The port map of the architecture section must include the design signal names.

Templates for the RAM_16S module are provided below as examples in both VHDL and Verilog code.

### VHDL Template Example

```
--
-- Module: RAM_16S
--
-- Description: VHDL instantiation template
-- Distributed RAM
-- Single Port 16 x 1
-- Can also be used for RAM16X1S_1
--
-- Device: Spartan-3 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 RAM Instantiation
U_RAM16X1S: RAM16X1S
port map (
    D    => , -- insert Data 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 Data output signal
);
--
------------------------------------------------------------------------
```

### Verilog Template Example

```
//
// Module: RAM_16S
//
// Description: Verilog instantiation template
// Distributed RAM
// Single Port 16 x 1
// Can also be used for RAM16X1S_1
//
// Device: Spartan-3 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 RAM Instantiation
RAM16X1S U_RAM16X1S (
    .D(),     // insert input signal
    .WE(),    // insert Write Enable signal
    .WCLK(),  // insert Write Clock signal
    .A0(),    // insert Address 0 signal
    .A1(),    // insert Address 1 signal
    .A2(),    // insert Address 2 signal
    .A3(),    // insert Address 3 signal
    .O()      // insert output signal
);
// synthesis attribute declarations
/* synopsys attribute
INIT "0000"
*/
```

### Wider Distributed RAM Modules

Table 6 shows the VHDL and Verilog distributed RAM examples that implement *n*-bit-wide memories.

*Table 6:* **VHDL and Verilog Submodules**

| Submodules | Primitive | Size | Type |
|---|---|---|---|
| XC3S_RAM16XN_S_SUBM | RAM16X1S | 16 words x n-bit | Single-port |
| XC3S_RAM32XN_S_SUBM | RAM32X1S | 32 words x n-bit | Single-port |
| XC3S_RAM64XN_S_SUBM | RAM64X1S | 64 words x n-bit | Single-port |
| XC3S_RAM16XN_D_SUBM | RAM16X1D | 16 words x n-bit | Dual-port |

### Initialization in VHDL or Verilog Codes

Distributed RAM structures can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the distributed RAM instantiation and are copied in the EDIF output file to be compiled by Xilinx ISE 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.

## Related Materials and References

Refer to the following documents for additional information:

- "Elements within a Slice" and "Function Generator" sections, *Spartan-3 Data Sheet (Module 2)*. Describes the CLB slice structure and distributed RAM function. http://www.xilinx.com/bvdocs/publications/ds099-2.pdf
- *Libraries Guide*, for ISE 5.2i by Xilinx, Inc. Distributed RAM primitives. Pages 1491-1571. http://toolbox.xilinx.com/docsan/xilinx5/pdf/docs/lib/lib.pdf

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 07/08/03 | 1.0 | Initial Xilinx release. |

# Using Look-Up Tables as Shift Registers (SRL16) in Spartan-3 FPGAs

XAPP465 (v1.0.1) July 5, 2003


## Summary

The SRL16 is an alternative mode for the look-up tables where they are used as 16-bit shift registers. Using this Shift Register LUT (SRL) mode can improve performance and rapidly lead to cost savings of an order of magnitude. Although the SRL16 can be automatically inferred by the software tools, considering their effective use can lead to more cost-effective designs.


## Introduction

Spartan™-3 FPGAs can configure the Look-Up Table (LUT) in a SLICEM slice 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 four of the eight LUTs as a 64-bit shift register.

This document provides generic VHDL and Verilog submodules and reference code examples for implementing from 16-bit up to 64-bit shift registers. These submodules are built from 16-bit shift-register primitives and from dedicated MUXF5, MUXF6, and MUXF7 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 Spartan-3 shift register without using flip-flops (i.e., using the SRL16 element(s)), use the CORE Generator RAM-based Shift Register module.

## Shift Register Architecture

The structure of the SRL16 will be described from the bottom up, starting with the shift register and then building up to the surrounding FPGA structure.

### LUT Structure

The Look-Up Table can be described as a 16:1 multiplexer with the four inputs serving as binary select lines, and the values programmed into the Look-Up Table serving as the data being selected (see Figure 1).

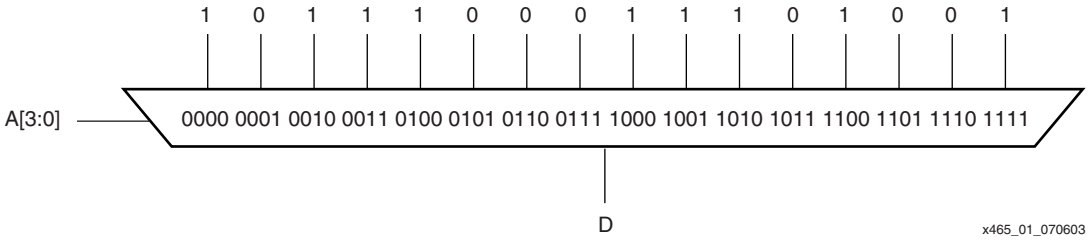



*Figure 1:* **LUT Modeled as a 16:1 Multiplexer**

With the SRL16 configuration, the fixed LUT values are configured instead as an addressable shift register (see Figure 2). The shift register inputs are the same as those for the synchronous RAM configuration of the LUT: a data input, clock, and clock enable (not shown). A special output for the shift register is provided from the last flip-flop, called Q15 on the library primitives or MC15 in the FPGA Editor. The LUT inputs asynchronously (or dynamically) select one of the 16 storage elements in the shift register.

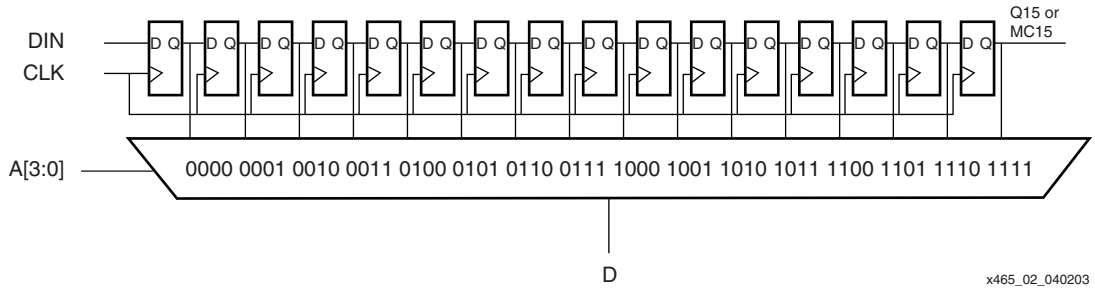



*Figure 2:* **LUT Configured as an Addressable Shift Register**

### Dynamic Length Adjustment

The address can be thought of as dynamically changing the length of the shift register. If D is used as the shift register output instead of Q15, setting the address to 7 (0111) selects Q7 as the output, emulating an 8-bit shift register. Note that since the address lines control the mux, they provide an asynchronous path to the output.

## Logic Cell Structure

Each SRL16 LUT has an associated flip-flop that makes up the overall logic cell. The addressable bit of the shift register can be stored in the flip-flop for a synchronous output or can be fed directly to a combinatorial output of the CLB. When using the register, it is best to have fixed address lines selecting a static shift register length. Since the clock-to-output delay of the flip-flop is faster than the shift register, performance can be improved by addressing the second-to-last bit and then using the flip-flop as the last stage of the shift register. Using the flip-flop also allows for asynchronous or synchronous set or reset of the output.

The shift register input can come from a dedicated SHIFTIN signal, and the Q15/MC15 signal from the last stage of the shift register can drive a SHIFTOUT output. The addressable D output is available in all SRL primitives, while the Q15/MC15 signal that can drive SHIFTOUT is only available in the cascadable SRLC16 primitive.

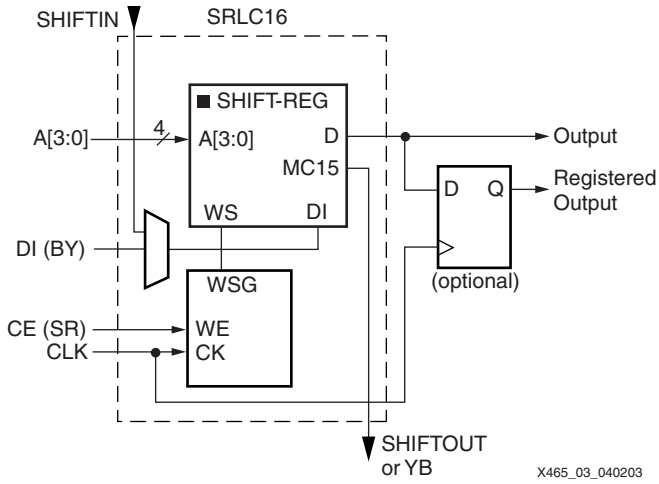



*Figure 3:* **Logic Cell SRL Structure**

## Slice Structure

The two logic cells within a slice are connected via the SHIFTOUT and SHIFTIN signals for cascading a shift register up to 32 bits (see Figure 4). These connect the Q15/MC15 of the first shift register to the DI (or Q0 flip-flop) of the second shift register.

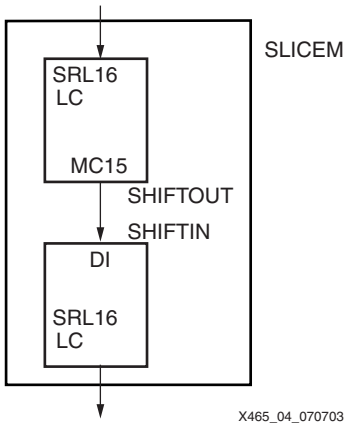



*Figure 4:* **Shift Register Connections Between Logic Cells in a Slice**

If dynamic addressing (or "dynamic length adjustment") is desired, the two separate data outputs from each SRL16 must be multiplexed together. One of the two SRL16 bits can be selected by using the F5MUX to make the selection (see Figure 5).

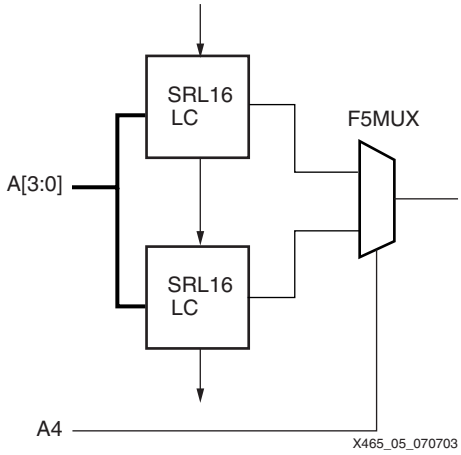



*Figure 5:* **Using F5MUX for Addressing Multiple SRL16 Components**

## CLB Structure

The Spartan-3 CLB contains four slices, each with two Look-Up Tables, but only two allow LUTs to be used as SRL16 components or distributed RAM. The two left-hand SLICEM components allow their two LUTs to be configured as a 16-bit shift register. The same cascading of SHIFTOUT to SHIFTIN available between the LUTs in the SLICEM is also available to connect the two SLICEM components. The four left-hand LUTs of a single CLB can be combined to produce delays up to 64 clock cycles (see Figure 6).

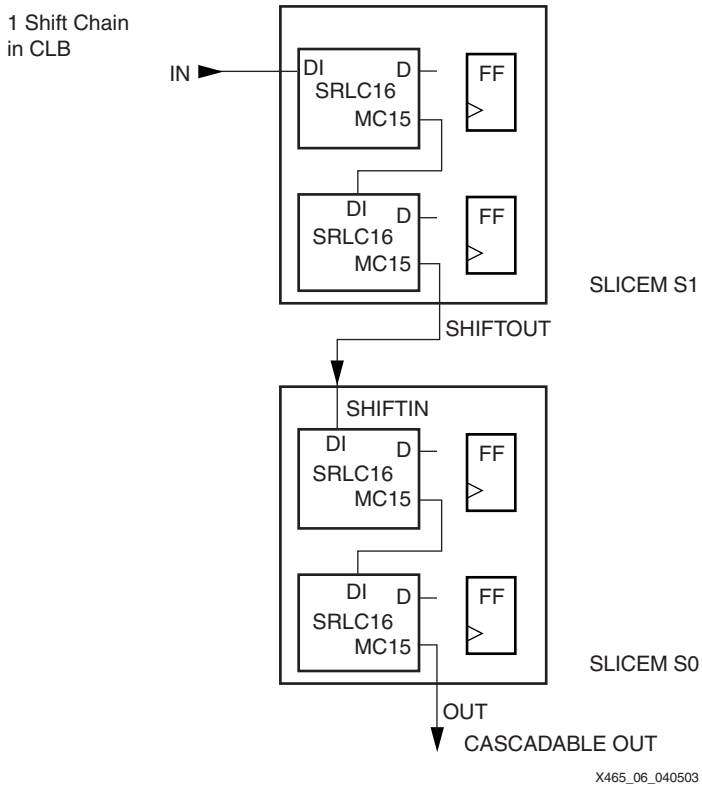



*Figure 6:* **Cascading Shift Register LUTs in a CLB**

The multiplexers can be used to address multiple SLICEMs similar to the description for combining the two LUTs within a SLICEM. The F6MUX can be used to select from three or four SRL16 components in a CLB, providing up to 64 bits of addressable shift register (see Figure 7).

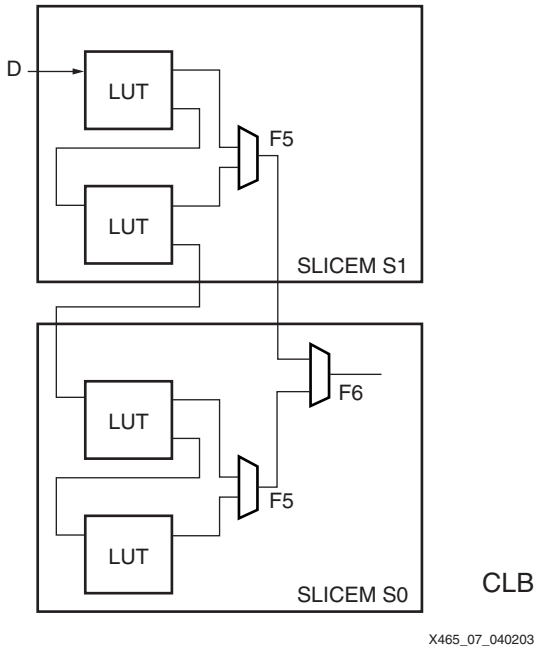



*Figure 7:* **Using F6MUX to Address a 64-Bit Shift Register**

## Library Primitives

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

Table 1 lists all of the available primitives for synthesis and simulation.

*Table 1:* **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 | 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 | CLK | A3, A2, A1, A0 | Q, Q15 |
| SRLC16E_1 | 16 bits | $\overline{CLK}$, CE | A3, A2, A1, A0 | Q, Q15 |

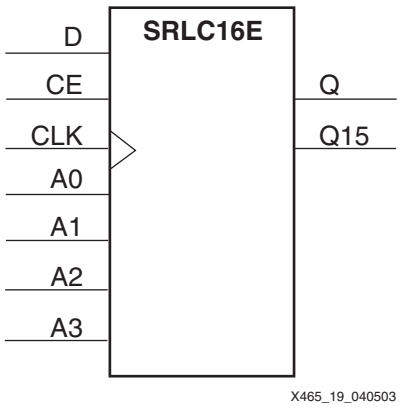



*Figure 8:* **SRLC16E Primitive**

### 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 INIT 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 S3_SRL16E shift register instantiation code examples (in VHDL and Verilog) illustrate these techniques (see "VHDL and Verilog Templates," page 240). `S3_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 — A3, A2, A1, A0

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

Figure 9 shows how the slices are arranged within a CLB. Each CLB has four slices, but only the two at the bottom-left of the CLB can be used as shift registers. These are both designated SLICEM in CLB positions S0 and S1. The relative position coordinates are X0Y0 and X0Y1. To constrain placement, these coordinates can be used in a LOC property attached to the SRL primitive. Note that the dedicated CLB shift chain runs from the top to the bottom, but the start and end of the shift register can be in any of the four SLICEM LUTs.

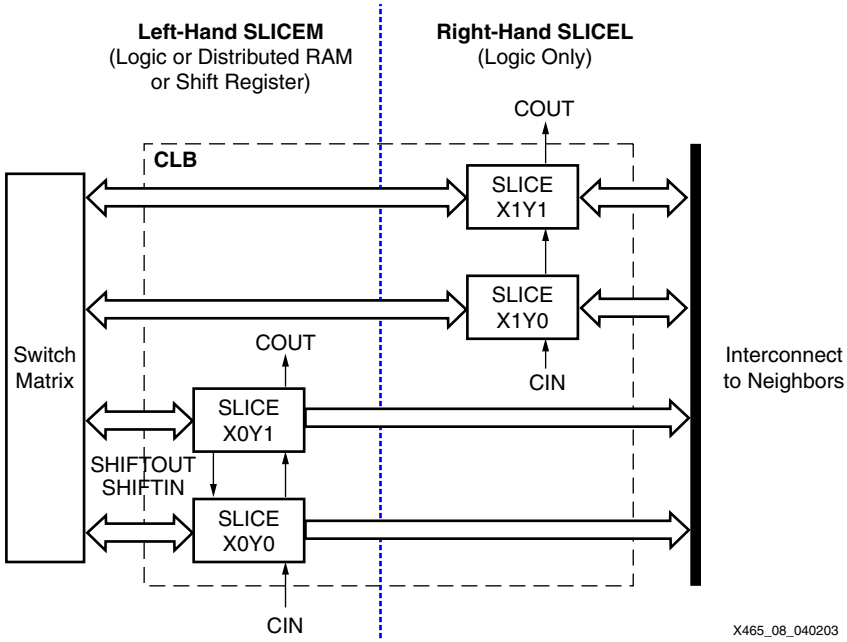



*Figure 9:* **Arrangement of Slices within the CLB**

## 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 10.

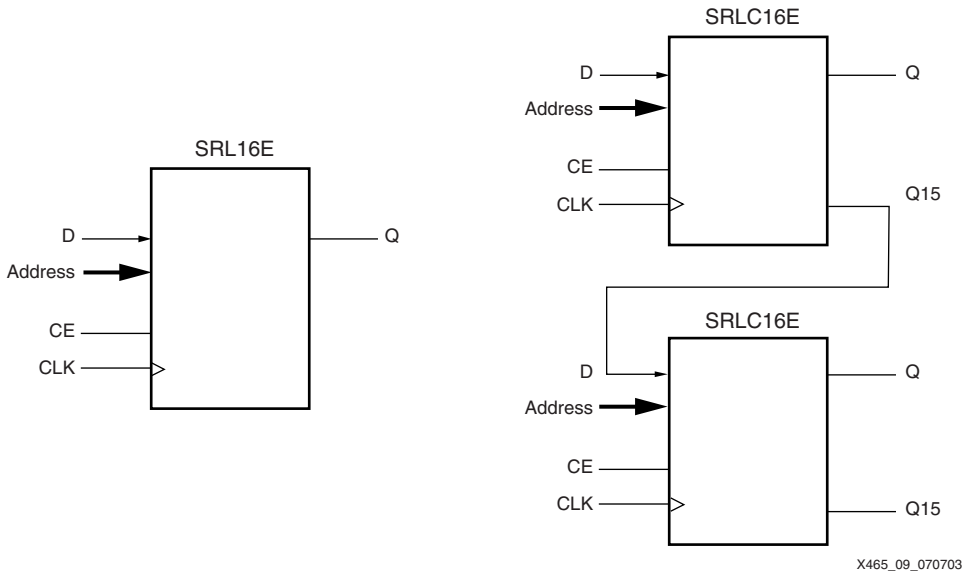



*Figure 10:* **Shift Register and Cascadable Shift Register**

### 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 11 illustrates the shift and dynamic read operations.

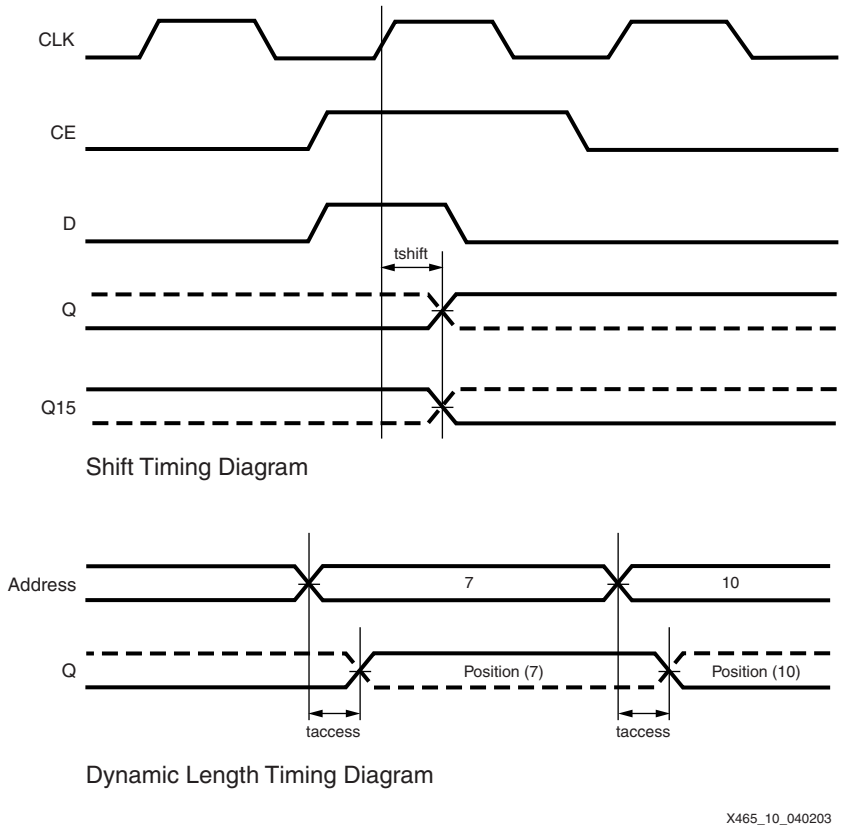



*Figure 11:* **Shift- and Dynamic-Length Timing Diagrams**

### 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.

## Shift Register Inference

When a shift register is described in generic HDL code, synthesis tools will infer the use of the SRL16 component. Note that since the SRL16 does not have either synchronous or asynchronous set or reset inputs, and does not have access to all bits at the same time, using such capabilities will preclude the use of the SRL16, and the function will be implemented in flip-flops. The cascadable shift register (SRLC16) may be inferred if the shift register is larger than 16 bits or if only the Q15 is used.

Although the SRL16 shift register does not have a parallel load capability, an equivalent function can be implemented simply by anticipating the load requirement and shifting in the proper data. This requires predictable timing for the load command.

### VHDL Inference Code

The following code infers an SRL16 in VHDL.

```
architecture Behavioral of srl16 is

signal Q_INT: std_logic_vector(15 downto 0);

begin

process(C)
begin
  if (C'event and C='1') then
    Q_INT <= Q_INT(14 downto 0) & D;
  end if;
end process;

Q <= Q_INT(15);

end Behavioral;
```

An inverted clock (SRL16_1) is inferred by replacing `C='1'` with `C='0'`. A clock enable (SRL16E) is inferred by inserting `if (CE='1') then` after the first if-then statement.

### Verilog Inference Code

The following code infers an SRL16 in Verilog.

```
always @ (posedge C)
begin
  Q_INT <= {Q_INT[14:0],D};
end

always @(Q_INT)
begin
  Q <= Q_INT[15];
end
```

An inverted clock (SRL16_1) is inferred by replacing (`posedge C`) with (`negedge C`). A clock enable (SRL16E) is inferred by inserting `if(CE)` after the begin statement.

## Shift Register Submodules

In addition to the 16-bit primitives, two submodules that implement 32-bit and 64-bit cascadable shift registers are provided in VHDL and Verilog code. Table 2 lists available submodules.

*Table 2:* **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 |

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 12 represents the cascadable shift registers (32-bit and 64-bit) implemented by the submodules in Table 2.

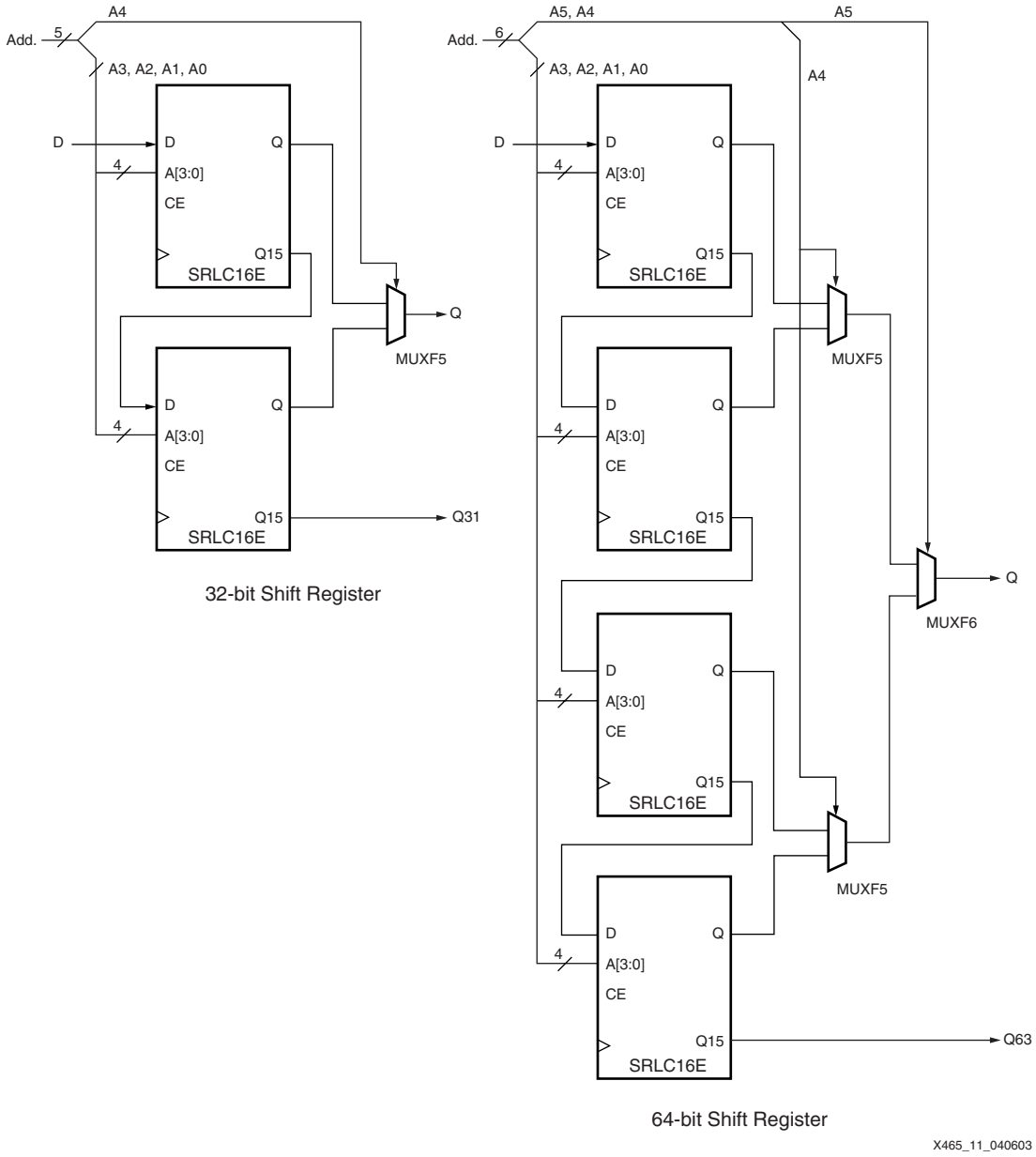



*Figure 12:* **Shift-Register Submodules (32-bit, 64-bit)**

All clock enable (CE) and clock (CLK) inputs are connected to one global clock enable and one clock signal per submodule. If a global static- or dynamic-length mode is not required, the SRLC16E primitive can be cascaded without multiplexers.

## 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 13.

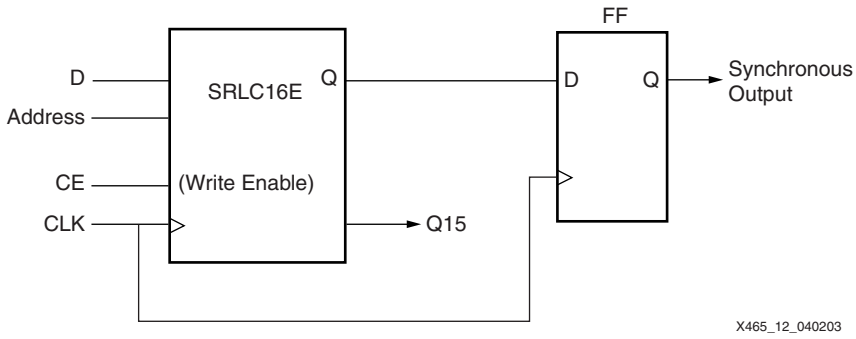



*Figure 13:* **Fully Synchronous Shift Register**

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 cascadable 16-bit shift register implements any static length mode shift register without the dedicated multiplexers (MUXF5, MUXF6, and so on). Figure 14 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.)

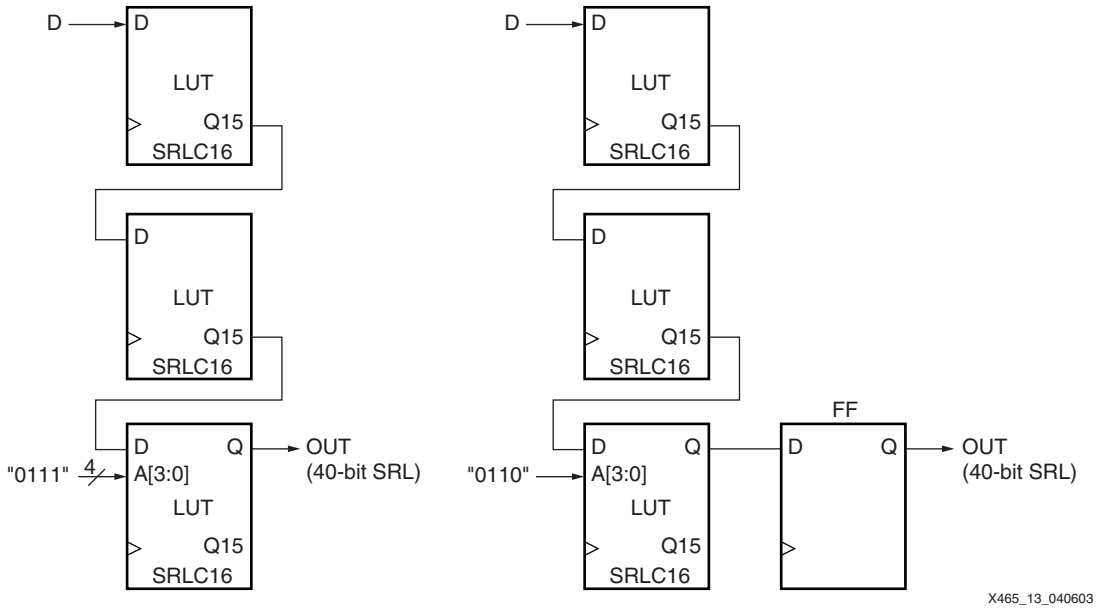



*Figure 14:* **40-bit Static-Length Shift Register**

## VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available for all primitives and submodules.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

The ShiftRegister_C_x (with x = 16, 32, or 64) templates are cascadable modules and instantiate the corresponding SRLCxE primitive (16) or submodule (32 or 64).

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

### VHDL and Verilog Templates

In template 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)

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: Spartan-3 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: Spartan-3 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"
  */
```

## CORE Generator System

The Xilinx CORE Generator™ system generates fast, compact, FIFO-style shift registers, delay lines, or time-skew buffers using the SRL16. The RAM-based Shift Register module shown in Figure 15 provides a very efficient multibit wide shift for widths up to 256 and depths to 1024. Fixed-length shift registers and variable-length shift registers can be created. An option is also provided to register the outputs of the module. If output registering is selected, there are additional options for Clock Enable, Asynchronous Set, Clear, and Init, and Synchronous Set, Clear and Init of the output register. The module can optionally be generated as a relationally placed macro (RPM) or as unplaced logic.

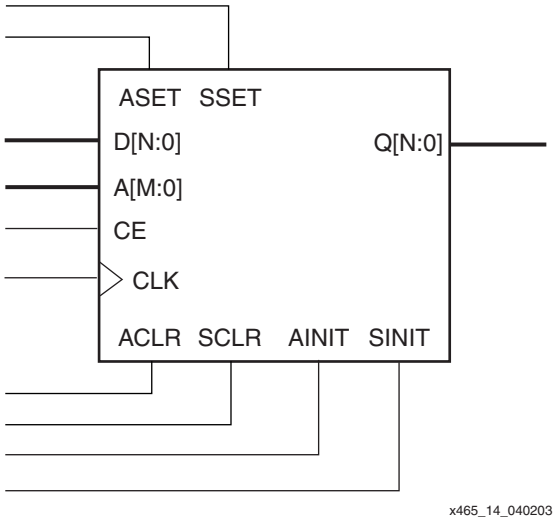



*Figure 15:* **CORE Generator RAM-Based Shift Register Module**

# Applications

## Delay Lines

The register-rich nature of the Xilinx FPGA architecture allows for the addition of pipeline stages to increase throughput. Data paths must be balanced to keep the desired functionality. The SRL16 can be used when additional clock cycles of delay are needed anywhere in the design (see Figure 16).

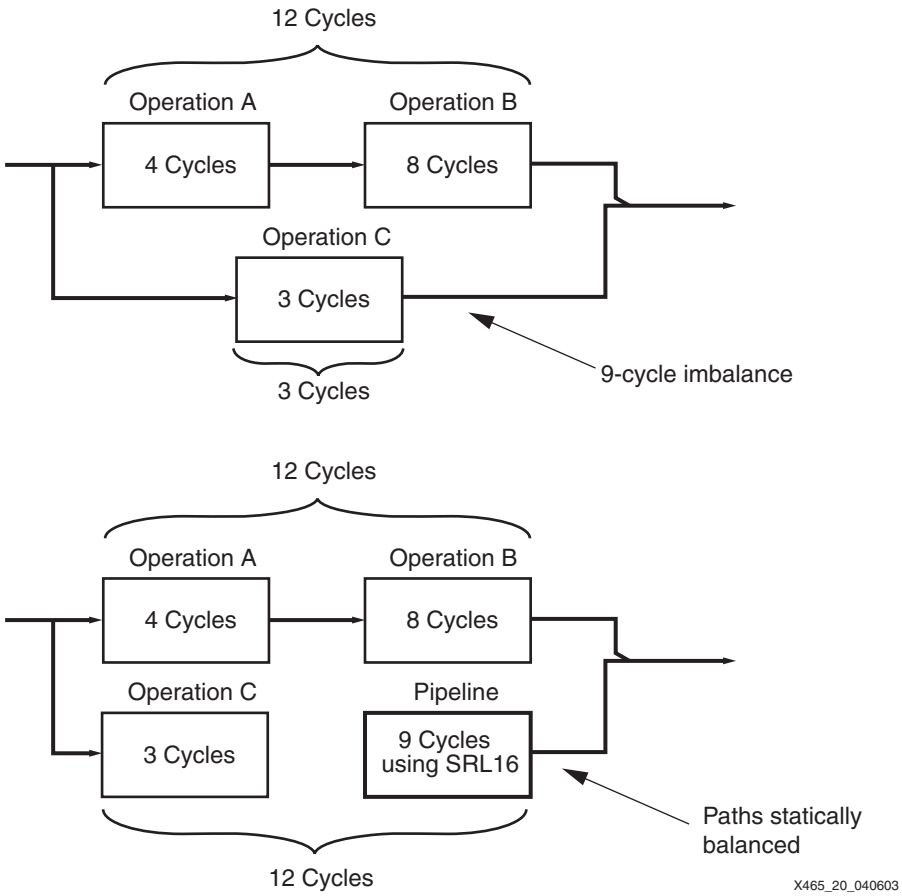



*Figure 16:* **Using SRL16 as a Delay Line**

## Linear Feedback Shift Registers

Linear Feedback Shift Registers (LFSRs) sequence through $2^n$-1 states, where n is the number of flip-flops. The sequence is created by feeding specific bits back through an XOR or XNOR gate. LFSRs can replace conventional binary counters in performance critical applications where the count sequence is not important (e.g., FIFOs). LFSRs are also used as pseudo-random number generators. They are important building blocks in encryption and decryption algorithms.

Maximal-length LFSRs need taps taken from specific positions within the shift register. There are multiple ways these taps can be made available in the SRL16 configuration. One is by addressing the necessary bit in a given SRL16 while allowing the Q15 to cascade to the next SRL16. Another is to use flip-flops to "extend" the SRL16 where necessary to access the tap points. For example, Figure 17 shows how a 52-bit LFSR can be implemented in one CLB with the feedback coming from bits 49 and 52. A third method is to duplicate the LFSR in multiple SRLs and address different bits from each one. Yet another is to generate multiple addresses in one SRL clock cycle to capture multiple bit positions. The XNOR gate required for any LFSR can be conveniently located in the SLICEL part of the CLB. More detail is available in XAPP210.

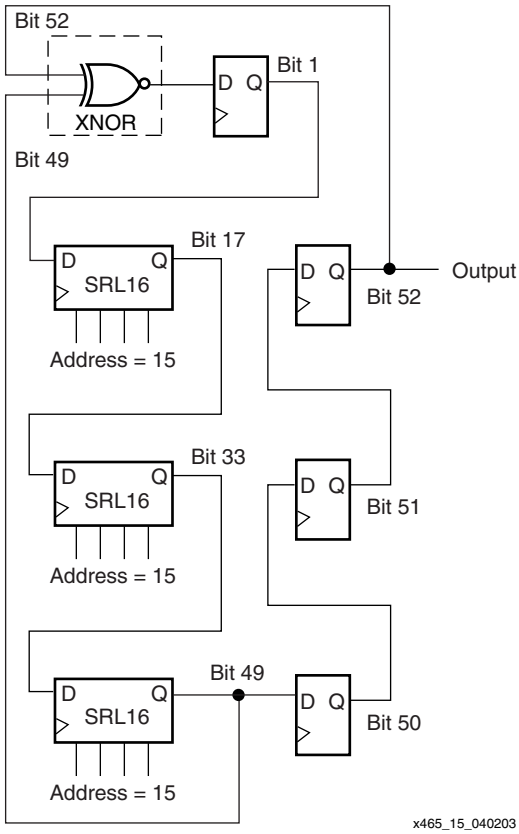



*Figure 17:* **52-bit LFSR in One CLB**

## Gold Code Generator

Gold code generators are used in CDMA systems to generate code sequences with good correlation properties (see Figure 18). R. Gold suggested that sets of small correlation codes could be generated by modulo 2 addition of the results of two LFSRs, primed with factor codes. The result is a set of codes ideally suited to distinguish one code from another in a spectrum full of coded signals. Figure 18 shows an implementation of a Gold code generator. The logic

required to initially fill the LFSR and provide the feedback can be located in the SLICEL parts of the CLB. See XAPP217 for more details.

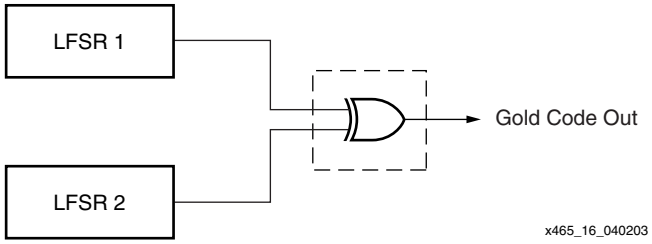



*Figure 18:* **Gold Code Generator**

## FIFOs

Synchronous FIFOs can be built out of the SRL16 components. These are useful when other resources become scarce, providing up to 64 bits per CLB. For larger FIFOs, the block RAM is the most efficient resource to use. See XAPP256 for more detail.

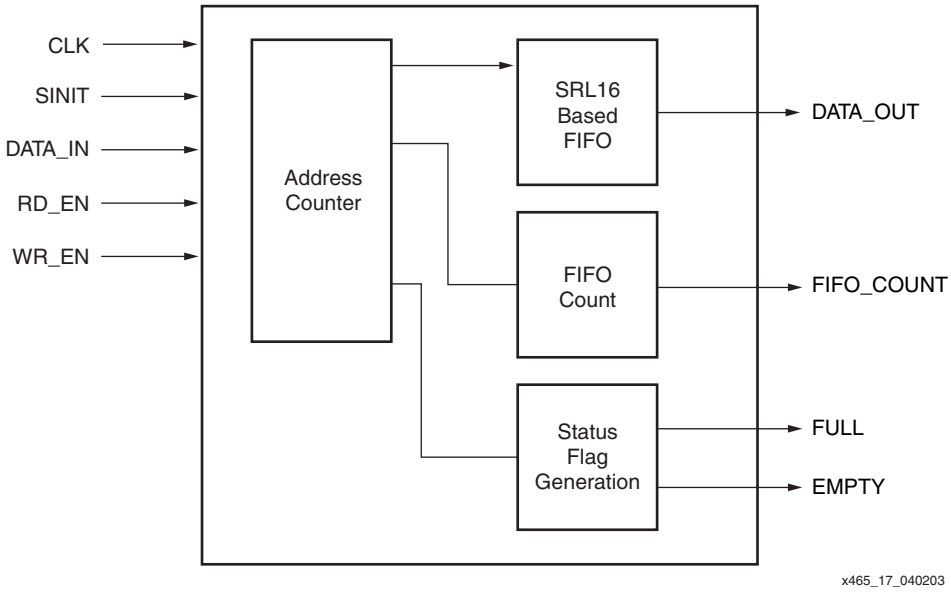



*Figure 19:* **Synchronous FIFO Using SRLC16 Shift Registers**

## Counters

Any desired repeated sequence of 16 states can be achieved by feeding each output with an SRL16. Cascading the SRL16 allows even longer arbitrary count sequences. A terminal count can be generated by using the standard carry chain (see Figure 20).

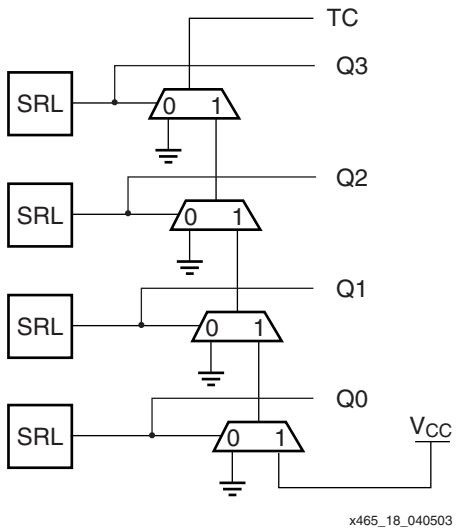



*Figure 20:* **SRL-Based Counter with Terminal Count**

## Related Materials and References

- XAPP210: *Linear Feedback Shift Registers in Virtex Devices*
  Linear Feedback Shift Registers are very efficient counters in the FPGA architecture. Using the SRL16 as the basis of the shift register, a 15-bit counter can fit in one slice and a 52-bit counter in two slices.
- XAPP211: *PN Generators Using the SRL Macro*
  Pseudo-random Noise sequences are used to code and spread signals across a wide band of transmission frequencies for spread spectrum modulation. PN generators are based upon LFSRs, which can be effectively built from the SRL16 components.
- XAPP217: *Gold Code Generators in Virtex Devices*
  A special type of PN sequence is a Gold code generator, which can be created from SRL16-based LFSRs.
- XAPP220: *LFSRs as Functional Blocks in Wireless Applications*
  Further discussion of the usage of LFSRs such as Gold Code Generators in applications such as CDMA.
- XAPP256: *FIFOs Using Virtex-II Shift Registers*
  The SRL16 is ideal for building smaller synchronous FIFOs. FIFOs can be built in any width while producing a 1-bit resolution. With cascaded SRL16 shift registers, a flexible depth in multiples of 16 is available. These techniques are useful for even larger FIFOs when block RAM resources are not available.
- TechXclusive: "The SRL16E: How Using this Exciting Mode Can Lead to Cost Saving of an Order of Magnitude"
  Describes the SRL16 function and its application in pipeline compensation, pseudo random noise generators, serial frame synchronizers, running averages, pulse generation and clock division, pattern generation, state machines, dynamically addressable shift registers, FIFOs, and an RS232 receiver.
- DS228: *RAM-Based Shift Register LogiCORE Module*
  Generates fast, compact, FIFO-style shift registers, delay lines or time-skew buffers using the SRL16.
- SRL16 Primitives in Libraries Guide
  Describes the usage and functionality of the SRL16 primitive and its variations.

## Conclusion

The SRL16 configuration of the Spartan-3 look-up table provides a space-efficient shift register that would otherwise require 16 flip-flops. This feature will be automatically used when a small shift register is described in HDL code. However, creative consideration of the uses of the SRL16 as described here can provide even more significant advantages in many applications.

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 04/10/03 | 1.0 | Initial Xilinx release. |
| 07/05/03 | 1.0.1 | Changed title. |

XAPP466 (v1.0.1) July 5, 2003

# Using Dedicated Multiplexers in Spartan-3 FPGAs


## Summary

The Spartan™-3 architecture includes dedicated multiplexers within the Configurable Logic Blocks (CLBs). These specialized multiplexers improve the performance and density of not just wide multiplexers but almost any wide-input function. Using these resources, a 32:1 multiplexer fits in just one level of logic, as do some Boolean logic functions of up to 79 inputs.


## Introduction

A multiplexer, or mux, is a common building block of almost every logic design, selecting one of several possible input signals. Spartan-3 FPGAs are very efficient at implementing multiplexers: small ones in the look-up tables and larger ones using dedicated multiplexer resources. Any Spartan-3 slice easily implements a 4:1 mux, any CLB implements up to a 16:1 mux, and two CLBs implement up to a 32:1 mux. The same logic resources also can be used for wide, general-purpose logic functions. For applications like comparators, encoder-decoders, or case statements, these resources provide an optimal solution. These resources are used automatically by the Xilinx development system.

This document describes the dedicated multiplexer resources in the Spartan-3 architecture. The signals and parameters associated with the multiplexers are defined. The many methods to include multiplexers in a design are described along with recommendations and guidelines for their use.

### Advantages of Dedicated Multiplexers

Spartan-3 FPGAs are based on four-input Look-Up Tables (LUTs) that can provide any possible function of the four inputs. The largest mux that a single LUT supports is a 2:1 mux, with the fourth input available as a possible enable. One method to construct larger muxes would be to cascade multiple LUTs. For example, a 4:1 mux could be built by combining the outputs of two LUTs into a third LUT. However, this method adds two full levels of logic delays plus an additional routing delay between the LUTs. Without special resources, an 8:1 mux would consume seven LUTs as well as add three levels of logic delays plus two levels of routing delays, as shown in Figure 1.

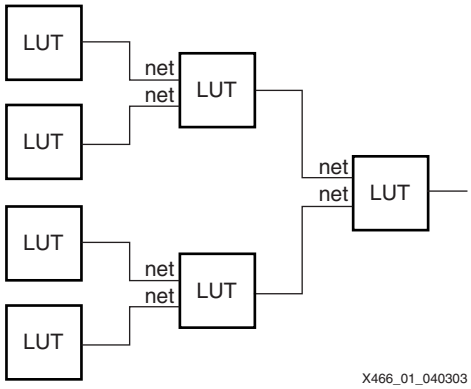



*Figure 1:* **8:1 Mux, 7 LUTs, 3 Levels of Logic**

To increase multiplexer speed and density, Spartan-3 FPGAs provide a dedicated 2:1 mux following every LUT, which replaces additional levels of LUT-based logic. One of these, called the F5MUX, combines adjacent LUTs to create a 4:1 mux. The other mux, following every pair of LUTs, combines muxes into even wider functions, with different capabilities depending on its location in the CLB. This mux is called the FiMUX, where the index "i" equals 6, 7, or 8. For example, the F6MUX combines the results of two F5MUX elements to create an 8:1 mux as shown in Figure 2. The connections from the LUTs to the muxes and between the muxes are dedicated and have zero connection delay. The combination of LUTs and dedicated multiplexers allows very efficient implementation of even large multiplexers.

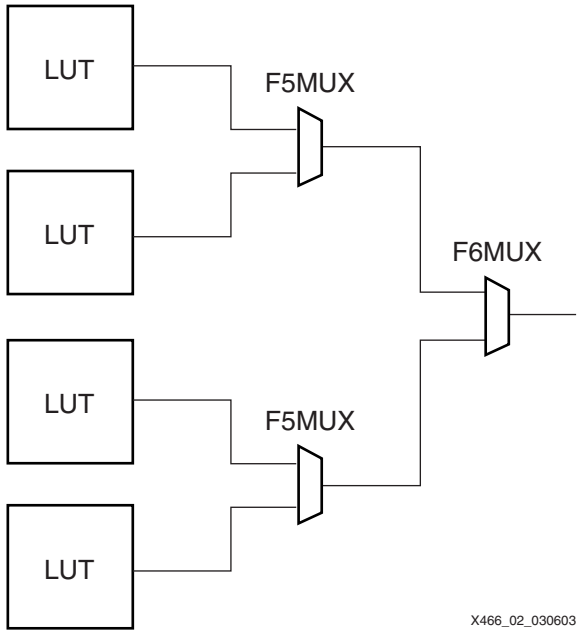



*Figure 2:* **8:1 Mux, 4 LUTs, 1 Level of Logic**

## Spartan-3 CLB Multiplexer Resources

The Spartan-3 architecture consists of an array of identical Configurable Logic Blocks, or CLBs. Each CLB is made up of four slices: two SLICEMs with memory capability and two SLICELs with logic-only capability. Each slice is identical with respect to logic and mux resources. Each slice has two LUTs, an F5MUX, and a second expansion mux (see Figure 3).

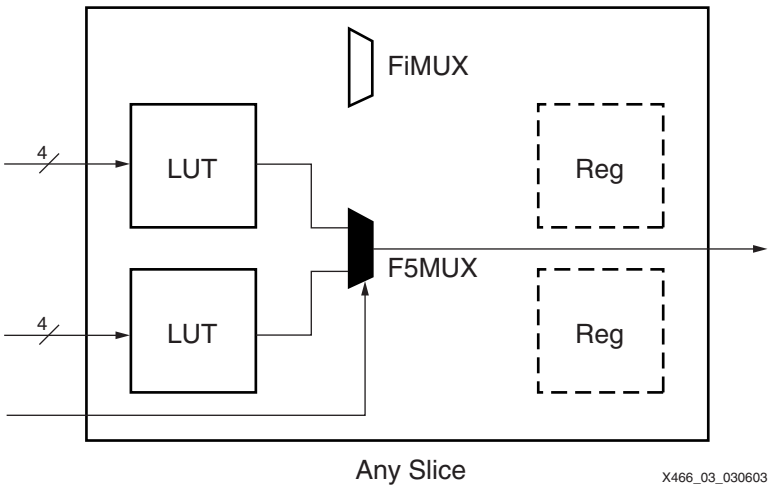



*Figure 3:* **LUTs and F5MUX in a Slice**

## F5MUX

The F5MUX always combines the two LUTs in a slice. If those two LUTs contain 2:1 muxes with the same control input, then the overall result is a 4:1 mux (see Figure 4).

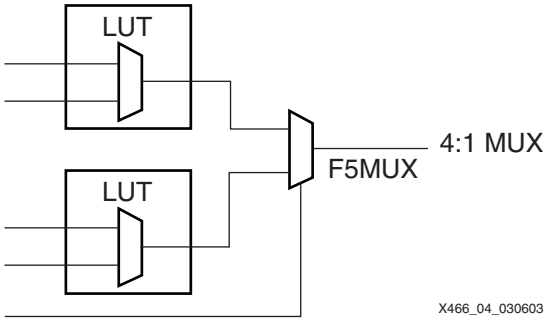



*Figure 4:* **4:1 Mux Implemented Using F5MUX**

The F5MUX is so named because it generates any possible Boolean logic function of five inputs (see Figure 5). If the two LUTs contain independent functions of the same four inputs, the mux select line becomes the fifth input. The F5MUX becomes a function expander that is just as efficient as another 3-input LUT for implementing any 5-input function. This is a significant advantage over other FPGA architectures.

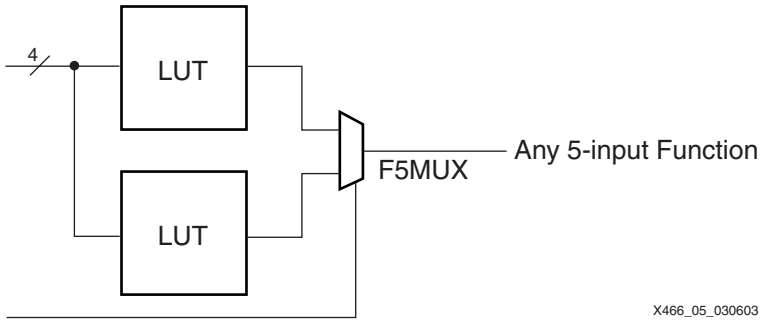



*Figure 5:* **Any 5-input Function Can Be Implemented Using F5MUX**

As shown in Figure 6, the F5MUX also produces some functions of up to nine inputs, if they can be partitioned into two 4-input LUTs and a mux.

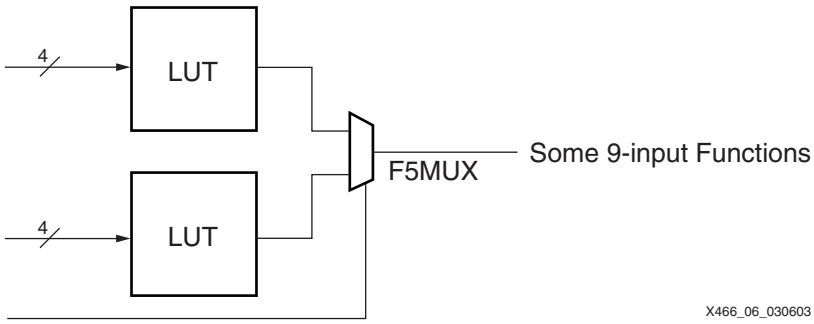



*Figure 6:* **Some 9-input Functions Can Be Implemented Using F5MUX**

Consequently, the F5MUX generates any 5-input function, the 4:1 mux 6-input function, or some 9-input functions.

## FiMUX

The second mux, called the FiMUX, functions as either a F6MUX, F7MUX, or F8MUX, depending on its location and connections to the other muxes.

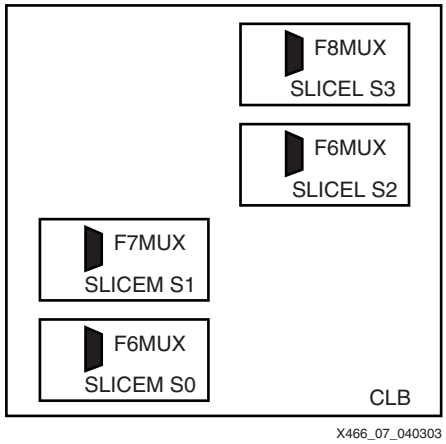



*Figure 7:* **Mux Positions in a CLB**

Each FiMUX receives inputs from muxes of the next lower number; for example, the two F6MUX results drive the F7MUX. Like the F5MUX, the FiMUX has the flexibility to implement other types of functions besides just multiplexers. The F6MUX is so named because it creates any function of six inputs. Similarly, the F7MUX generates any function of seven inputs, and the F8MUX generates any function of eight inputs.

*Table 1:* **Mux Capabilities**

| Mux | Usage | Input Source | Total Number of Inputs per Function | | |
|---|---|---|---|---|---|
| | | | For Any Function | For Mux | For Limited Functions |
| F5MUX | F5MUX | LUTs | 5 | 6 (4:1 mux) | 9 |
| FiMUX | F6MUX | F5MUX | 6 | 11 (8:1 mux) | 19 |
| | F7MUX | F6MUX | 7 | 20 (16:1 mux) | 39 |
| | F8MUX | F7MUX | 8 | 37 (32:1 mux) | 79 |

### Naming Conventions

In this document and in the Spartan-3 data sheet, the mux that serves as either F6MUX, F7MUX, or F8MUX generically is called an FiMUX (i = 6, 7, or 8). This name avoids confusion with the static CLB mux that generates the X output, which the FPGA Editor refers to as the FXMUX. The FiMUX is always referred to as the F6MUX in the FPGA Editor. The timing analyzer also refers to the path through the FiMUX to the CLB pin as TIF6Y, although it may be used as an F7MUX or F8MUX.

The library components are called MUXF5, MUXF6, MUXF7, and MUXF8. MUXF6, MUXF7, and MUXF8 use the FiMUX and restrict the placement to a specific relative location in the CLB.

## Dedicated Local Routing

A significant benefit of the dedicated multiplexers is the dedicated routing that connects between levels. Although each mux is implemented as one pass through the CLB, the outputs connect back to the CLB inputs through local interconnect with zero routing delay. The result is the same as if the muxes were in series within the CLB.

The F5MUX feeds the F5 CLB output pin, which only connects back to an FiMUX input on the same CLB (called FXINA and FXINB). The FiMUX feeds the FX CLB output pin, which also

feeds back to an FiMUX input on the same CLB, or in the case of the F7MUX, also to the CLB below. If the mux result is needed elsewhere, it connects to a general-purpose CLB output (X for the F5MUX, Y for the FiMUX).

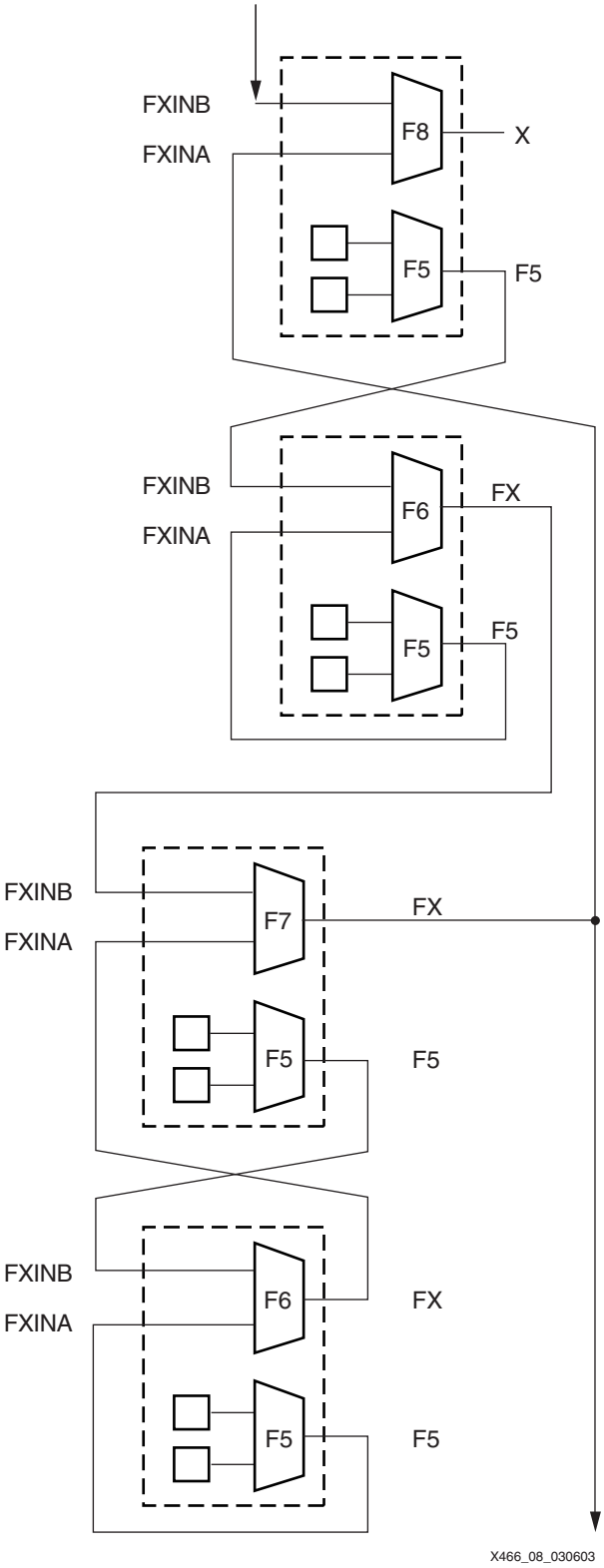



*Figure 8:* **Muxes and Dedicated Feedback in a Spartan-3 CLB**

### Mux Select Inputs

The select inputs for the multiplexers come from general-purpose routing. The select input for the F5MUX is the BX input on the CLB, and the select input for the FiMUX is the BY input on the CLB.

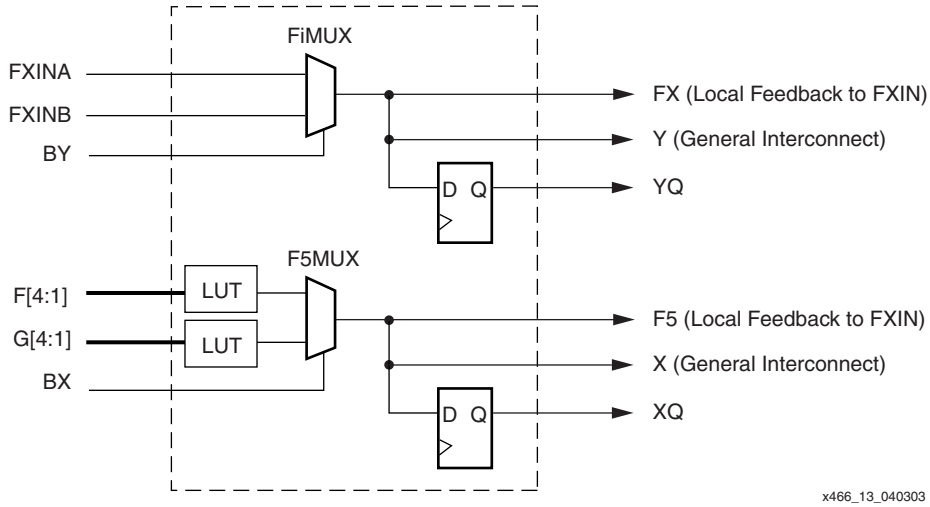



*Figure 9:* **Dedicated Multiplexers in Spartan-3 CLB**

## Implementation Examples

### Wide-Input Multiplexers

Each LUT optionally implements a 2:1 multiplexer. In each slice, the F5MUX and two LUTs can implement a 4:1 multiplexer. As shown in Figure 10, the F6MUX and two slices implement an 8:1 multiplexer. The F7MUX and the four slices of any CLB implement a 16:1 multiplexer, and the F8MUX and two CLBs implement a 32:1 multiplexer.

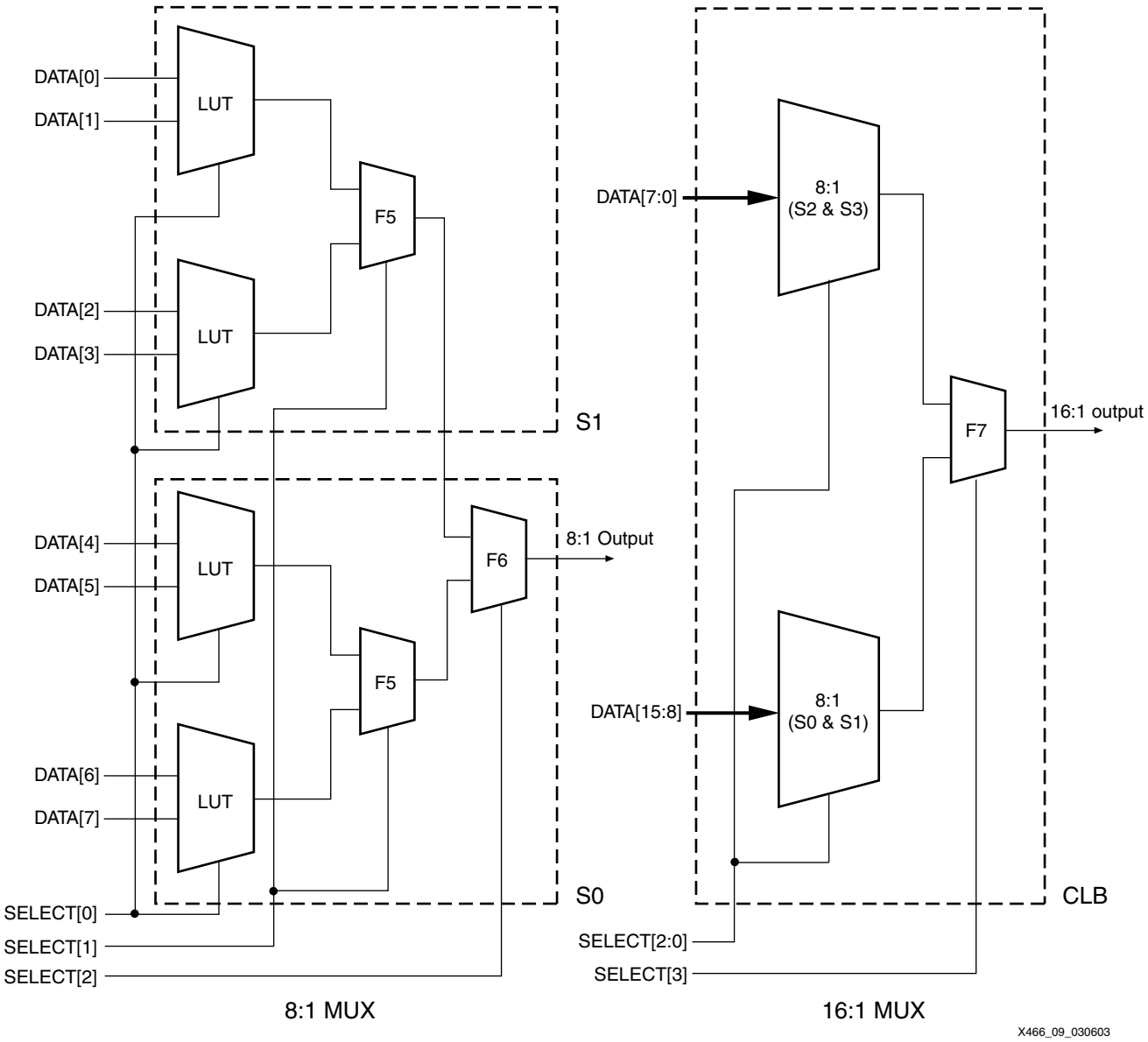



*Figure 10:* **8:1 and 16:1 Multiplexers**

## Wide-Input Functions

Slices S0 and S2 have an F6MUX, designed to combine the outputs of two F5MUX resources. Figure 11 illustrates a combinatorial function up to 19 inputs in the slices S0 and S1, or in the slices S2 and S3.

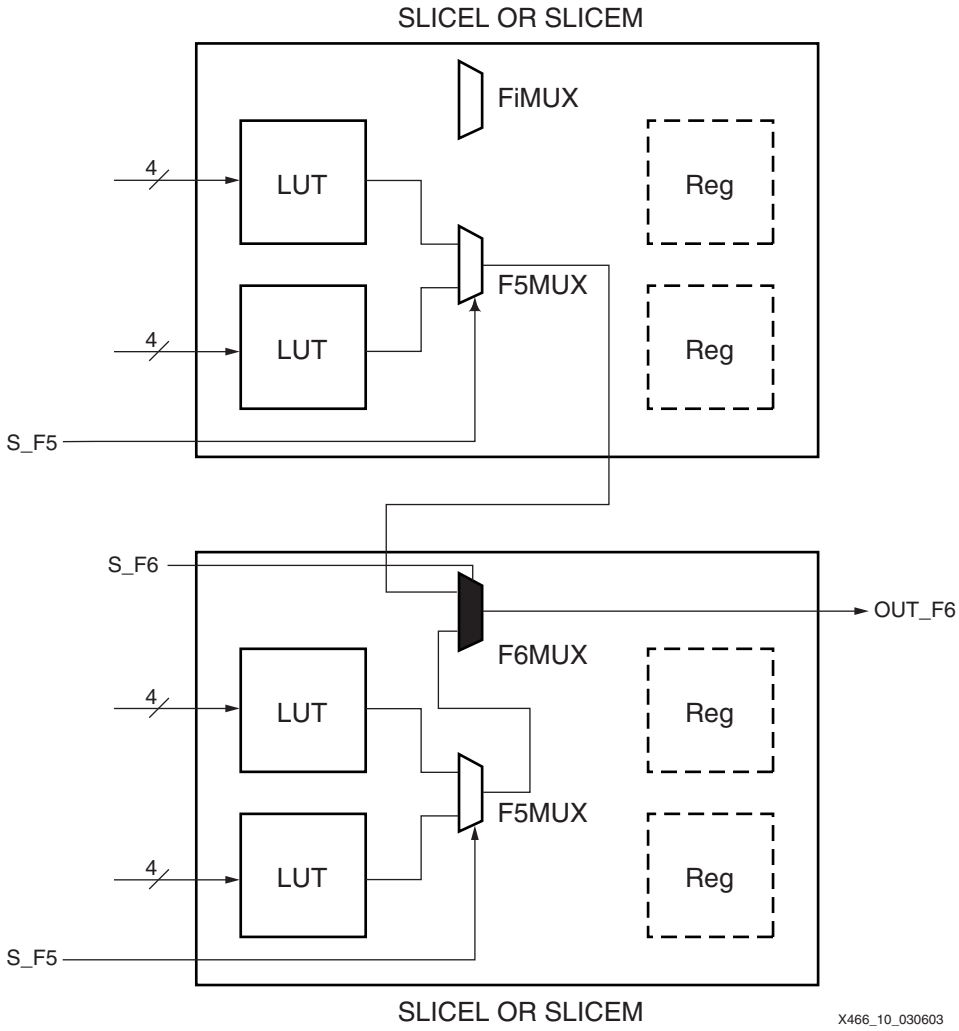



*Figure 11:* **19-input Function Using F6MUX in Two Slices**

The slice S1 has an F7MUX, designed to combine the outputs of two F6MUXs. Figure 12 illustrates a combinatorial function up to 39 inputs in a Spartan-3 CLB.

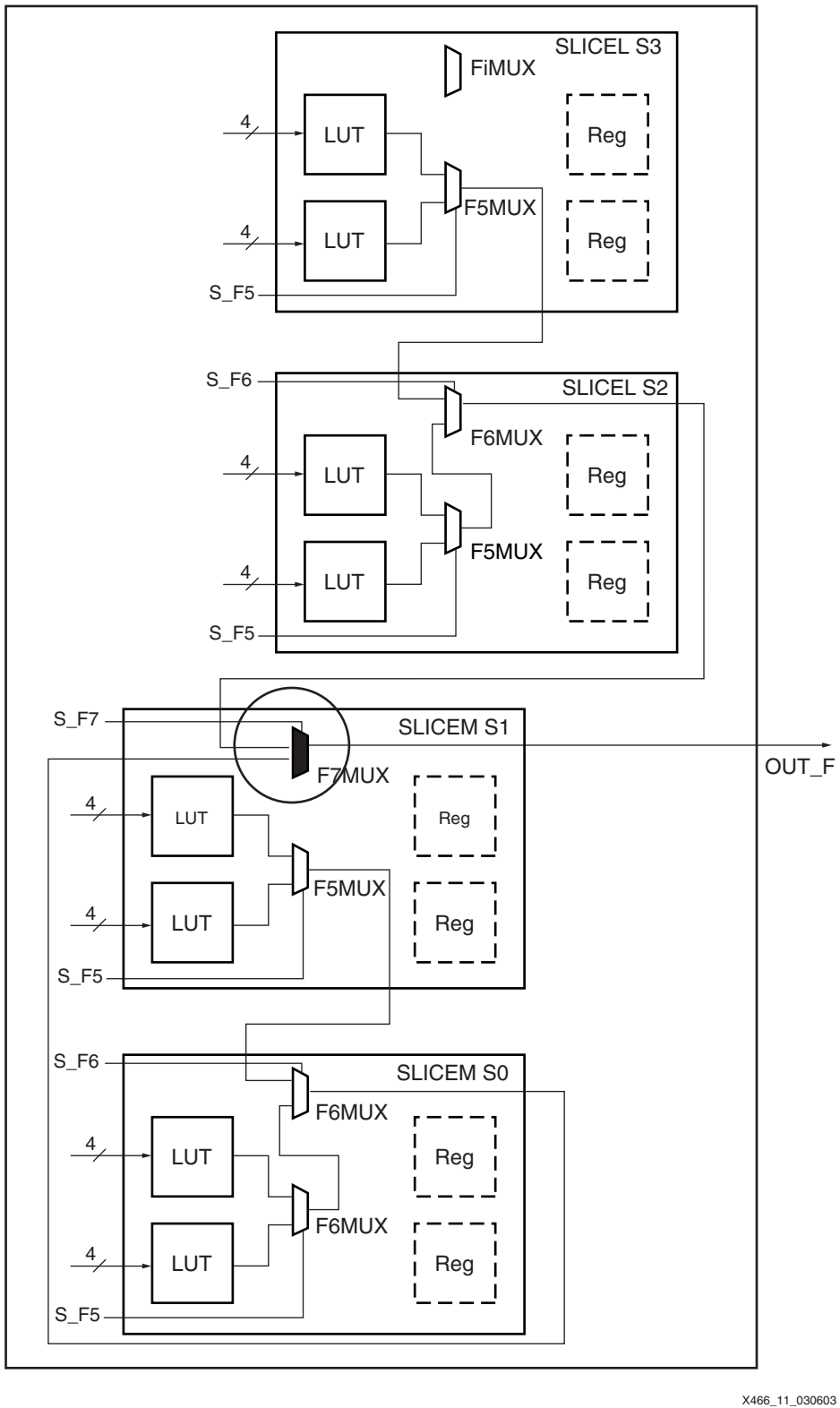



*Figure 12:* **39-input Function Using F7MUX in One CLB**

The slice S3 of each CLB has an F8MUX. Combinatorial functions of up to 79 inputs fit in two CLBs as shown in Figure 13. The outputs of two F7MUXs are combined through dedicated routing resources between two adjacent CLBs in a column.

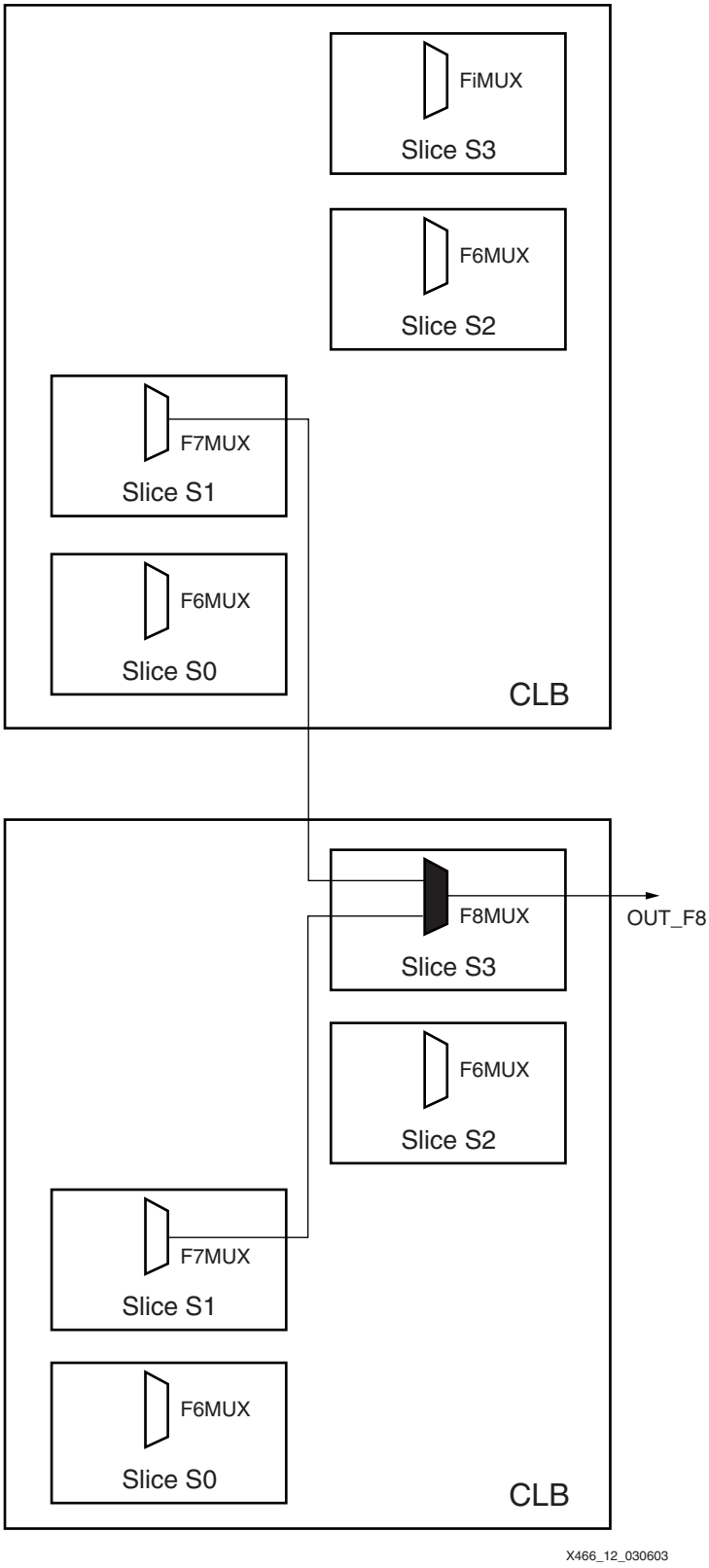



*Figure 13:* **79-input Function Using F8MUX in Two Adjacent CLBs**

## Timing Parameters

There are several possible paths through the CLB multiplexers. The two types of multiplexers are considered separately (F5MUX and FiMUX). Each multiplexer type has two types of inputs: data inputs and select lines. The output of the mux drives the local interconnect through the F5 and FX CLB pins, the general interconnect through the X and Y CLB pins, or the D input on the flip-flop. See Figure 9, page 252 for a block diagram showing dedicated multiplexers in a Spartan-3 CLB. Note that although the mux functionality is identical between the slices with memory and those without, the timing values are independent and may vary slightly.

Although the multiplexers are connected in series inside the CLB, each mux actually feeds a CLB output pin, which feeds back to an input pin through zero-delay local interconnect. Thus each reported block delay element will have only one mux from input to output. The Spartan-3 architecture improves on the Virtex™-II architecture by providing a direct path from the F5MUX or FiMUX to the flip-flop in the CLB.

*Table 2:* **Multiplexer Timing Paths**

| Symbol | CLB Input | Through | CLB Output |
|---|---|---|---|
| $t_{IF5}$ | F/G LUT Inputs | LUT and F5MUX Inputs | F5 |
| $t_{IF5X}$ | F/G LUT Inputs | LUT and F5MUX Inputs | X |
| $t_{IF5CK}$ | F/G LUT Inputs | LUT and F5MUX Inputs | D input on flip-flop |
| $t_{BXF5}$ | BX | F5MUX Select | F5 |
| $t_{BXX}$ | BX | F5MUX Select | X |
| $t_{INAFX}$ | FXINA | FiMUX Inputs | FX |
| $t_{INBFX}$ | FXINB | FiMUX Inputs | FX |
| $t_{IF6Y}$ | FXINA or FXINB | FiMUX Inputs | Y |
| $t_{BYFX}$ | BY | FiMUX Select | FX |
| $t_{BYY}$ | BY | FiMUX Select | Y |

### Programmable Polarity

As with most resources in the Spartan-3 FPGA, inverters are free in large multiplexers. The functions in the LUT can have inverters added to inputs or outputs with no effect on performance or utilization. The control inputs to the F5MUX (BX) and FiMUX (BY) have programmable polarity inside the CLB.

## Related Uses of Multiplexers

### Multiplexers and Three-State Buffers

The LUT and MUX resources multiplex one of several inputs signals onto an internal routing resource, using the routing like an internal bus. This is equivalent to the BUFT-based multiplexers found in other FPGA architectures. In most modern FPGA families, these three-state buffers actually are implemented as dedicated logic gates to avoid possible contention when more than one is enabled at a time. The Spartan-3 family reduces die size and cost by eliminating the overhead of these internal three-state buffer gates. Instead, internal functions defined as a three-state buffer in the Spartan-3 family must be implemented in the LUTs and dedicated muxes.

The CLB multiplexers providing binary encoding of the select lines, requiring fewer signals than the one-hot encoding of the BUFT-based multiplexers. CLB-based multiplexers have no limit on width as BUFT-based multiplexers did, nor nay special placement considerations.

The BUFT component, representing a three-state buffer, is not available in the Spartan-3 library, except for the output function in the IOBs. The CORE Generator™ functions of the BUFT-based Multiplexer (and the equivalent BUFE-based Multiplexer) will be implemented as multiplexers in the CLBs.

### Multiplexers and Memory Functions

The F5MUX and FiMUX are used also to expand the distributed memory and shift register capability in the CLB. Those functions are not included in this document.

### Other CLB Multiplexers

The CLB also contains several other multiplexers for routing signals through the logic resources. The CYMUX for propagating carry signals is the only other dynamic mux. Several other muxes are used for selecting one of multiple paths. One is called the FXMUX in the FPGA Editor, since it routes the F LUT signal to the X CLB output. Do not confuse this static mux with the FXMUX name that is sometimes used for the FiMUX described here.

## Designing with Multiplexers

There are several ways multiplexers can be used in a design. The most common is to simply have them inferred by synthesis tools when appropriate for a design. Library primitives can be used to instantiate specific multiplexers. This document provides HDL submodules that combine the library primitives into larger muxes. The CORE Generator system includes the Bus Multiplexer and Bit Multiplexer functions, and many other CORE solutions take advantage of the dedicated multiplexers.

### Inference

Multiplexers are typically inferred by a conditional statement, most commonly the CASE or IF-THEN-ELSE statement. The IF statement generally produces priority-encoded logic. The CASE statement is more likely to generate an optimized multiplexer.

Synthesis options may determine whether multiplexers are inferred and how they are implemented. For XST, the MUX_EXTRACT constraint specifies whether multiplexers are inferred, and the MUX_STYLE constraint specifies whether they are implemented in the dedicated logic multiplexers or the carry multiplexers (CY_MUX). The default is to infer automatically the best resource.

CASE statements should be full (all branches defined) to avoid creating a latch. They also should be parallel (branch conditions all mutually exclusive) to avoid a priority encoder. Some synthesis tools, such as XST, have options to assume full and parallel CASE statements even if not written that way.

Make sure you do not write the code such that your synthesis tool will infer BUFT-based multiplexers. A BUFT-based multiplexer usually requires a statement with a "Z" value. Some synthesis tools might automatically or optionally convert BUFT logic to multiplexers.

A decoder is a special case of a multiplexer where the inputs are fixed as one-hot values. Decoders of up to 4:16 in size are easily implemented in individual LUTs for each output and do not need to use the dedicated multiplexers, or they can even use the Carry muxes for high performance.

The following subsections provide examples of 2:1 muxes described using the CASE statement in Verilog and VHDL code.

#### Verilog Inference

```
module MUX_2_1 (DATA_I, SELECT_I, DATA_O);

input [1:0]DATA_I;
input SELECT_I;

output DATA_O;
reg DATA_O;

always @ (DATA_I or SELECT_I)

  case (SELECT_I)
```

```
      1'b0 : DATA_O <= DATA_I[0];
      1'b1 : DATA_O <= DATA_I[1];
      default : DATA_O <= 1'bx;
  endcase

endmodule
```

### VHDL Inference

```
entity MUX_2_1 is
    port (
        DATA_I: in std_logic_vector (1 downto 0);
        SELECT_I: in std_logic;
        DATA_O: out std_logic
  );
end MUX_2_1;

architecture MUX_2_1_arch of MUX_2_1 is
--
begin
--
SELECT_PROCESS: process (SELECT_I, DATA_I)
begin
  case SELECT_I is
    when '0' => DATA_O <= DATA_I (0);
    when '1' => DATA_O <= DATA_I (1);
    when others => DATA_O <= 'X';
  end case;
end process SELECT_PROCESS;
--
end MUX_2_1_arch;
```

## Library Primitives

Four library primitives are available that offer access to the dedicated multiplexers in each slice: MUXF5, MUXF6, MUXF7, and MUXF8. Each of the multiplexer primitives looks identical (see Figure 14). The actual selection simply determines where in the CLB the multiplexer can be located, as shown in Table 3.

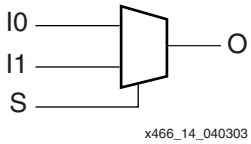



*Figure 14:* **MUXF5 Primitive**

*Table 3:* **Multiplexer 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 |

Note that the generic multiplexer components also can take advantage of the dedicated multiplexers. The M2_1 component is implemented in a look-up table, while the larger multiplexers in the library use the F5MUX and FiMUX components.

### Enable Signals in Multiplexers

An enable signal on a multiplexer can be used to keep the multiplexer output Low when disabled. Note that although the dedicated multiplexers do not have enable signals, the enable can be implemented on the preceding 2:1 mux that will be implemented in a look-up table. The M4_1E and M8_1E library components are built this way, using the F5MUX and F6MUX for the final result, respectively, while the M16_1E library component keeps the enable on the final mux, forcing it into a LUT instead of the F7MUX. Figure 15 shows the M4_1E library component logic.

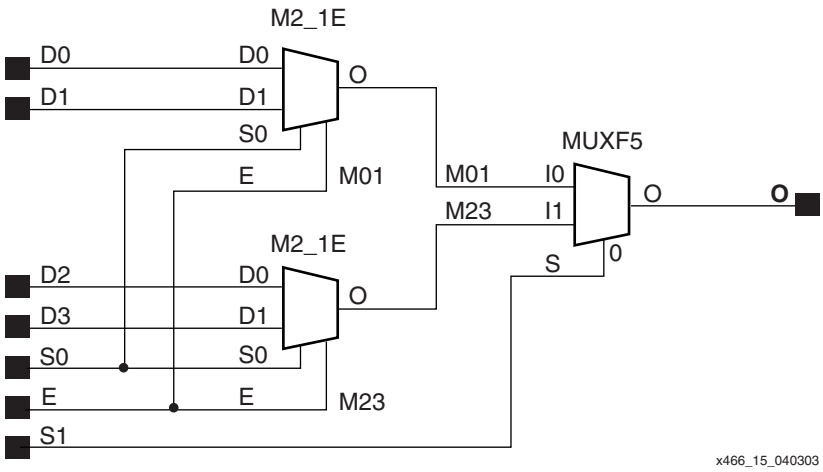



*Figure 15:* **M4_1E Library Component Logic**

### Modeling Local Output Timing

There are also two alternative versions of each library component that are functionally identical but can be used for more accurate timing estimation before implementation. As mentioned previously, the multiplexers can drive one or both CLB outputs. The first output is the special CLB output that feeds directly back through local interconnect to the next multiplexer in series, known as the local output. The second output is the general-purpose CLB output, which can be routed to any other logic. For better pre-implementation timing estimation, the user can substitute special primitives that specify whether to use the local output timing or the general-purpose output timing. The MUXF5_L primitive models the local output, while the MUXF5_D primitive models both output paths (see Figure 16). The functionality is identical to that for the MUXF5 primitive.

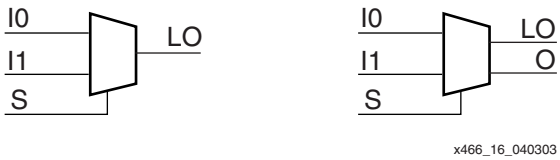



*Figure 16:* **MUXF5_L and MUX5F_D Primitives to Model Local Output Timing**

## Submodules

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 use instantiation of the multiplexers to guarantee an optimized result. Table 4 lists available submodules.

*Table 4:* **Available Submodules**

| Submodule | Multiplexer | Control | Input | Output |
|---|---|---|---|---|
| MUX_2_1_SUBM | 2:1 | SELECT_I | DATA_I[1:0] | DATA_O |
| MUX_4_1_SUBM | 4:1 | SELECT_I[1:0] | DATA_I[3:0] | DATA_O |
| MUX_8_1_SUBM | 8:1 | SELECT_I[2:0] | DATA_I[8:0] | DATA_O |
| MUX_16_1_SUBM | 16:1 | SELECT_I[3:0] | DATA_I[15:0] | DATA_O |
| MUX_32_1_SUBM | 32:1 | SELECT_I[4:0] | DATA_I[31:0] | DATA_O |

## Port Signals

### Data In — DATA_I

The data input provides the data to be selected by the SELECT_I signal(s).

### Control In — SELECT_I

The select input signal or bus determines the DATA_I signal to be connected to the output DATA_O. For example, the MUX_4_1_SUBM multiplexer has a 2-bit SELECT_I bus and a 4-bit DATA_I bus. Table 5 shows the DATA_I selected for each SELECT_I value.

*Table 5:* **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 Spartan-3 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 MUX resources. When synthesis infers the corresponding MUX resource(s), the VHDL or Verilog code is behavioral code (“case” statement). Otherwise, the equivalent “case” statement is provided in comments and the correct MUX are instantiated.

However, most synthesis tools support the inference of all of the MUXs. 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: Spartan-3 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: Spartan-3 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 support 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
*/
//If synthesis tools do not support MUXF7 :
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
```

## CORE Generator System

The CORE Generator system offers the BaseBLOX functions of the Bit Multiplexer and the Bus Multiplexer. The Bit Multiplexer, shown in Figure 17, supports sizes up to 256 inputs; the Bus Multiplexer, shown in Figure 18, supports muxes of up to 32 inputs for buses of up to 256 bits each. These core solutions have a parameter Mux Type to select a BUFT or LUT based multiplexer. Select the appropriate radio button in the CORE Generator system for the construction of the multiplexer. The default setting is LUT Based, which is required for Spartan-3 multiplexers. The CORE Generator system also offers options for registering the output of the multiplexer.

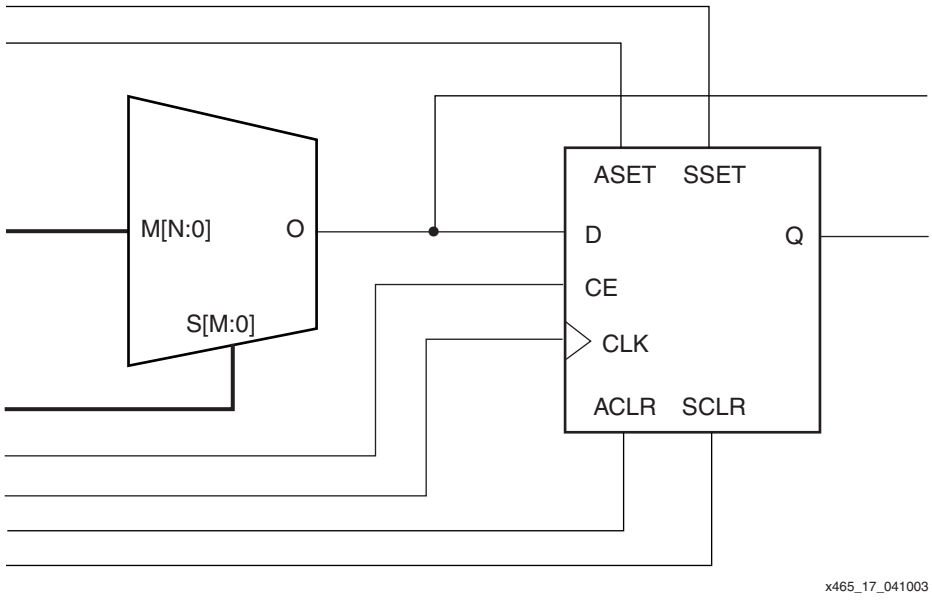



*Figure 17:* **Bit Multiplexer CORE Symbol**

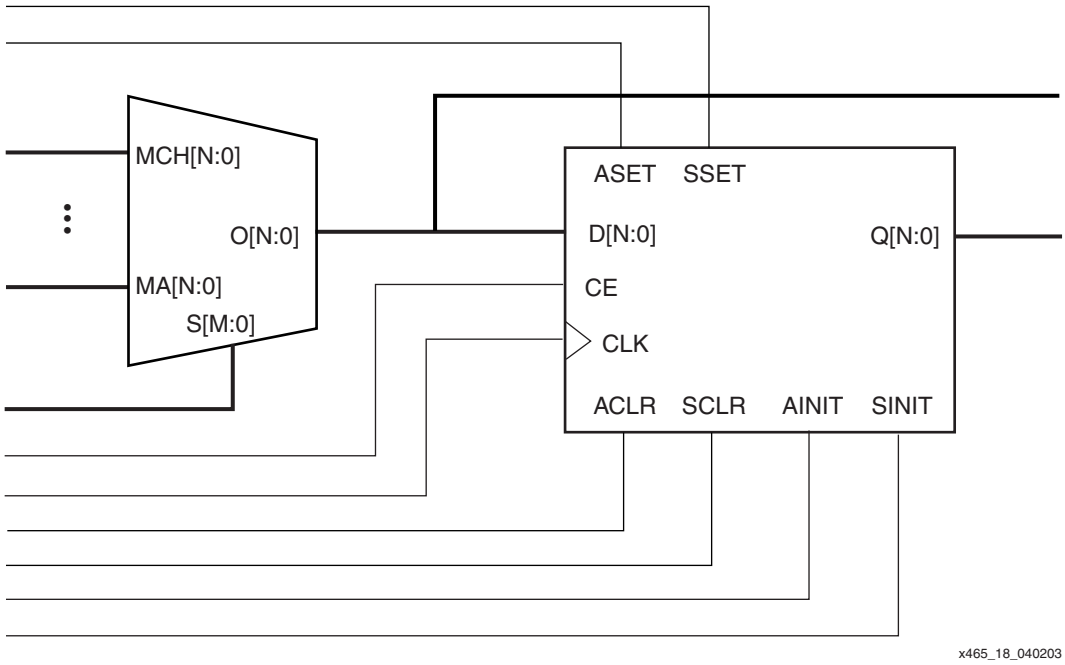



*Figure 18:* **Bus Multiplexer CORE Symbol**

The CORE Generator system also offers the specific functions of the BUFT-based Multiplexer (and the equivalent BUFE-based Multiplexer). As with the generic Bit and Bus Multiplexers, they are implemented in LUTs and/or muxes.

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 04/10/03 | 1.0 | Initial Xilinx release. |
| 07/05/03 | 1.0.1 | Changed title. |

XAPP467 (v1.1) May 13, 2003

# Using Embedded Multipliers in Spartan-3 FPGAs


## Summary

Dedicated 18x18 multipliers speed up DSP logic in the Spartan™-3 family. The multipliers are fast and efficient at implementing signed or unsigned multiplication of up to 18 bits. In addition to basic multiplication functions, the embedded multiplier block can be used as a shifter or to generate magnitude or two's-complement return of a value. The multipliers can be cascaded with each other or CLB logic for larger or more complex functions.


## Introduction

Spartan-3 FPGAs have a number of features to fortify the chip's arithmetic capabilities. Carry logic and dedicated carry routing continues to be provided as in past generations. Dedicated AND gates in the CLBs accelerate array multiplication operations. The newest and most significant addition is the dedicated 18x18 two's-complement multiplier block. With 4 to 104 of these dedicated multipliers in each device, fast arithmetic functions can be implemented with minimal use of the general-purpose resources. In addition to the performance advantage, dedicated multipliers require less power than CLB-based 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 Spartan-3 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.

The 18-bit x 18-bit multipliers can be quickly created using the CORE Generator™ system, or they can be instantiated (or inferred) using VHDL or Verilog.

## 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 two inputs are referred to as the multiplicand and the multiplier, or the factors, while the output is the product. The MULT18X18 primitive is illustrated in Figure 1.

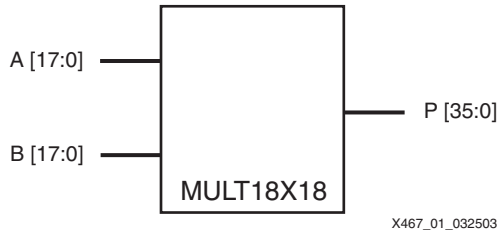



*Figure 1:* **Embedded Multiplier**

In addition, efficient cascading of multipliers up to 35-bit x 35-bit signed can be accomplished by using four embedded multipliers, one 36-bit adder, and one 53-bit adder. See Figure 6.

Binary multiplication is similar to regular multiplication with the multiplicand multiplied by each bit of the multiplier to generate partial products, and then the partial products added together to create the result. The Xilinx multiplier block uses the modified Booth algorithm, in effect using multiplexers to create the partial products.

### Timing Specification

The result is generated faster for the LSBs than the MSBs, since the MSBs require more levels of addition, so timing specifications are different for each of the 36 multiplier outputs. Designs should use only as many output bits as are necessary. For example, if two unsigned numbers will never have a product of $2^{35}$ or higher, the P[35] output is always zero. For any pair of signed numbers of n bits, if you will never have $-2^{n-1} \times -2^{n-1}$, then the MSB is always identical to the next lower-order bit (P[2n-1] = P[2n-2]). Also consider that if some outputs must have longer routing delays, they should be put on the output LSBs to balance with the MSB delays.

For the same reason, the data input setup time for the pipelined multiplier will be shorter for the MSBs than the LSBs, but the timing parameters do not differentiate between pins for setup time. For additional safety margin in a design, slower inputs should be put on the MSBs. The Reset and Clock Enable inputs have much faster setup times than any of the data inputs, and all have zero hold times. The timing parameter name "$t_{MULIDCK}$" (MULtiplier Input Data to ClocK) is used for both the data and control inputs, but will have different values for each type.

## Library Primitives

Two library primitives are available for the embedded multipliers. Table 1 describes these primitives.

*Table 1:* **Multiplier Primitives**

| Primitive | A Width | B Width | P Width | Signed/Unsigned | Output |
|---|---|---|---|---|---|
| MULT18X18 | 18 | 18 | 36 | Signed (Two's Complement) | Combinatorial |
| MULT18X18S | 18 | 18 | 36 | Signed (Two's Complement) | Registered |

The registered version of the multiplier adds a clock input C, an active-High Clock Enable CE, and a synchronous Reset R (see Figure 2). The registers are implemented in the multiplier itself and do not require any other resources. The control inputs C, CE, and R all have built-in programmable polarity. The data inputs, clock enable, and reset all must meet a setup time before the clock edge, and the data on the P outputs changes after the clock-to-output delay.

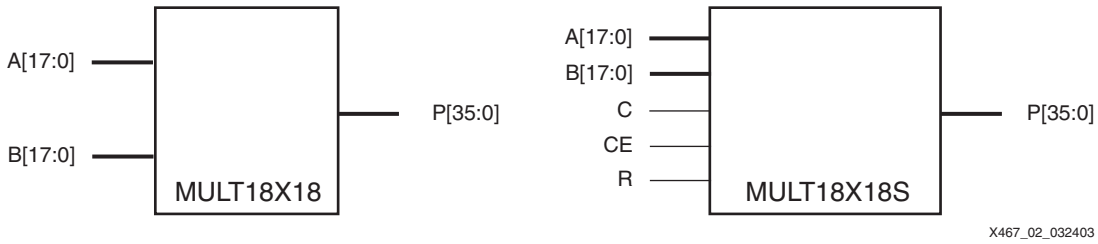



*Figure 2:* **Combinatorial and Registered Multiplier Primitives**

The pin names used in the Xilinx implementation tools, such as the FPGA Editor, are identical to those used in the library primitives.

## VHDL Instantiation Template

```
-- Component Declaration for MULT18X18 should be placed
-- after architecture statement but before begin keyword
component MULT18X18
  port ( P : out STD_LOGIC_VECTOR (35 downto 0);
         A : in STD_LOGIC_VECTOR (17 downto 0);
         B : in STD_LOGIC_VECTOR (17 downto 0));
end component;
-- Component Attribute specification for MULT18X18
-- should be placed after architecture declaration but
-- before the begin keyword
-- Attributes should be placed here
-- Component Instantiation for MULT18X18 should be placed
-- in architecture after the begin keyword
MULT18X18_INSTANCE_NAME : MULT18X18
  port map (P => user_P,
            A => user_A,
            B => user_B);
```

## Verilog Instantiation Template

```
MULT18X18 MULT18X18_instance_name (.P (user_P),
                                   .A (user_A),
                                   .B (user_B));
```

## MULT_STYLE Constraint

The MULT_STYLE constraint controls the implementation of the MULT18X18 primitives. In the Project Navigator (see Figure 3), the default is that the Xilinx Synthesis Tool (XST) will select the best type of implementation. To ensure that the embedded multipliers are used, set MULT_STYLE = Block or select "Block" for the "Multiplier Style" property in the Project Navigator. The MULT_STYLE constraint can also be applied globally at the XST command line or attached to a MULT18X18 primitive. For the MULT18X18S, attach the MULT_STYLE constraint to the component, not the output bus. See the Constraints Guide for more information.

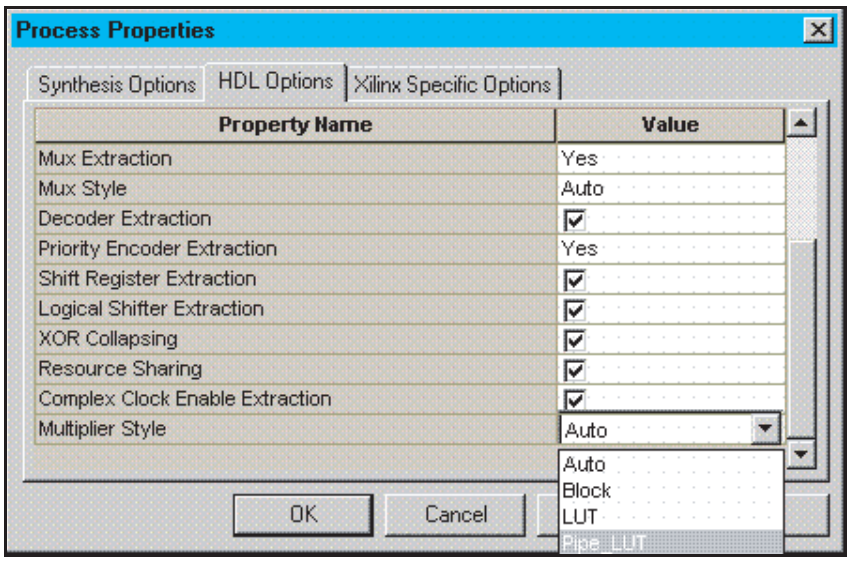
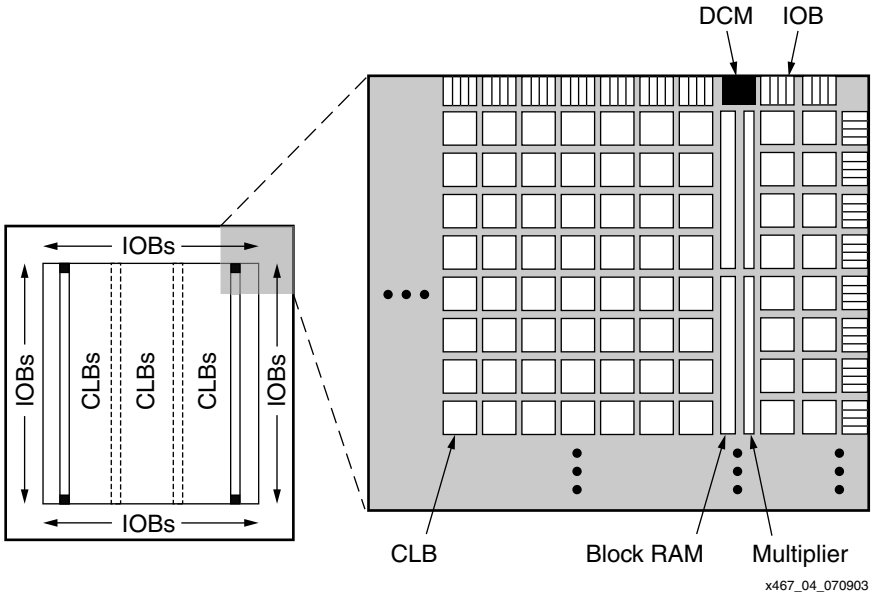



*Figure 3:* **Setting Multiplier Style in Project Navigator Process Properties**

## Multipliers in the Spartan-3 Architecture

The multipliers are located adjacent to the block RAM, making it convenient to store inputs or results in the block memory (see Figure 4). There are two or four columns of multipliers in each device. Where there are two columns, they have two rows of CLBs between them and the edge, allowing the multiplier to be easily driven by CLB or IOB logic. There are four CLBs, or 16 slices and 32 LUTs, on either side of a given multiplier block, allowing 32 input and output signals to be connected immediately adjacent to the multiplier block. One possible high-speed layout is to put A[15:0] on one side, B[15:0] on the other side, and intersperse the P[31:0] outputs on both sides. For a full-size 18x18 multiplier, the extra inputs and outputs can connect to the next CLB column. For best performance, pipeline the inputs with registers in the adjacent CLBs.




**Notes:**

1. The two additional block RAM/multiplier columns of the XC3S4000 and XC3S5000 devices are shown with dashed lines. The XC3S50 device has a single column of block RAM/multipliers along the left edge.

*Figure 4:* **Location of Multipliers in Spartan-3 Architecture**

The 18-bit width of the Spartan-3 multiplier is unusual but matches with the 18-bit width of the block RAM, which includes parity bits. Standard 8-bit or 16-bit multipliers can be created by using part of the multiplier block, or a 32-bit multiplier can be created via cascading. The Xilinx architecture allows any non-standard bit width to be implemented, exactly matching the needs of the application. Unused multiplier inputs are connected automatically to zero via connections to unused LUTs that are set to zero.

*Table 2:* **Number of Multipliers per Spartan-3 Device**

| Device | Multiplier Columns | Multipliers |
|---|---|---|
| XC3S50 | 1 | 4 |
| XC3S200 | 2 | 12 |
| XC3S400 | 2 | 16 |
| XC3S1000 | 2 | 24 |
| XC3S1500 | 2 | 32 |
| XC3S2000 | 2 | 40 |
| XC3S4000 | 4 | 96 |
| XC3S5000 | 4 | 104 |

## Expanding Multipliers

Multiplication using inputs with more than 18 bits is possible by decomposing the multiplication process into smaller subprocesses. The binary representation of either input can be split at any point, provided the proper weighting and sign of the MSBs is taken into account. Splitting off the 18 MSBs of the input makes the best use of the 18-bit signed multipliers.

For example, Figure 5 shows how a 22x16 multiplier could be implemented. The 22-bit value is decomposed into an 18-bit signed value and a 4-bit unsigned value from the LSBs. Two partial products are formed. The first is a 20-bit signed product, which is the result of multiplying the 16-bit signed value by the 4-bit unsigned section. The second is a 34-bit signed product, formed by multiplying the 16-bit signed value by the 18-bit signed section. The addition process restores the weighting of the products (note the least significant bits of the first product bypass the addition) and forms the final 38-bit product. Since the first product is signed, the 20-bit value needs to be sign-extended before addition. The adder itself only needs to be 34 bits, requiring 17 slices.

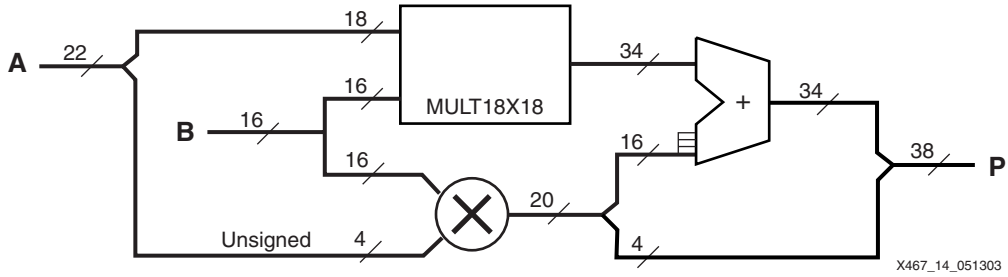



*Figure 5:* **22x16 Multiplier Implementation**

The implementation can vary depending on the performance needs and available resources. The second multiplier can be implemented in the MULT18X18 resource or in CLBs if it is small. Pipelining can be added to improve performance, using the built-in capabilities of the dedicated multipliers. If both inputs are greater than 18 bits, then four partial products are formed, but the purely unsigned result from the LSBs simply can be concatenated with the 36-bit signed product of the MSBs and added to the other two results.

Figure 6 represents the cascaded scheme used to implement a 35-bit by 35-bit signed multiplier utilizing four embedded multipliers and two adders.

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.

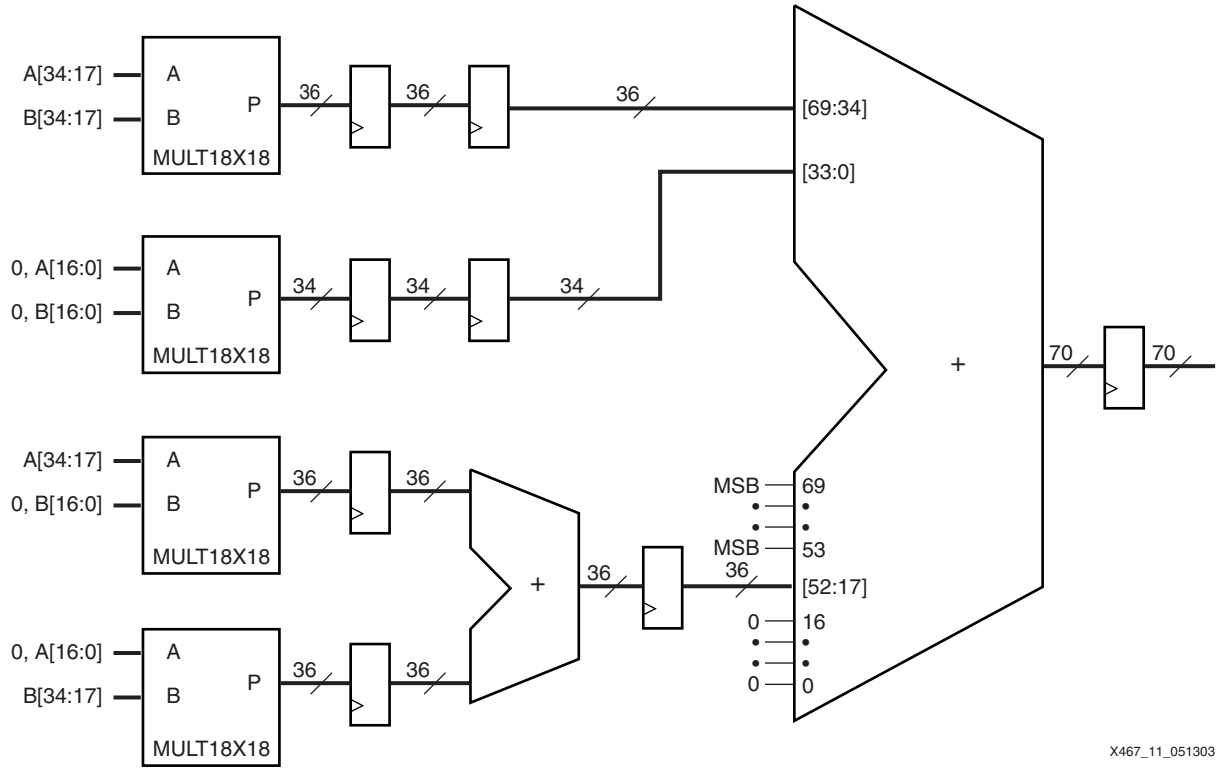



*Figure 6:* **35x35 Signed Multiplier**

## Two Multipliers in a Single Primitive

The dedicated multiplier can be used to multiply two smaller numbers at the same time. By putting one value on the LSBs and one on the MSBs, two independent results can be obtained as long as the results do not overlap with each other on the outputs. Shifting one of the values n positions to the MSBs is the same as multiplying it by $2^n$. If the value shifted to the MSBs is X, then the new value is $X * 2^n$. If the value on the LSBs is Y, then the complete multiplier input is $X * 2^n + Y$.

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:**

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

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

$X^2 * 2^{2n}$ {Y=0} ($X^2$ on the output MSBs)

$Y^2$ {X=0} ($Y^2$ on the output LSBs)

0 {X=0, Y=0}

With both X and Y at non-zero values, care must be taken to avoid overlap between the results on the MSBs and LSBs and the middle term ($XY * 2^{n+1}$). 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:**

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

Table 3 shows values for X and Y where these conditions are met.

*Table 3:* **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 |

Figure 7 represents the MULT18X18 connections for calculating the square of both a 6-bit signed number and a 5-bit unsigned number.

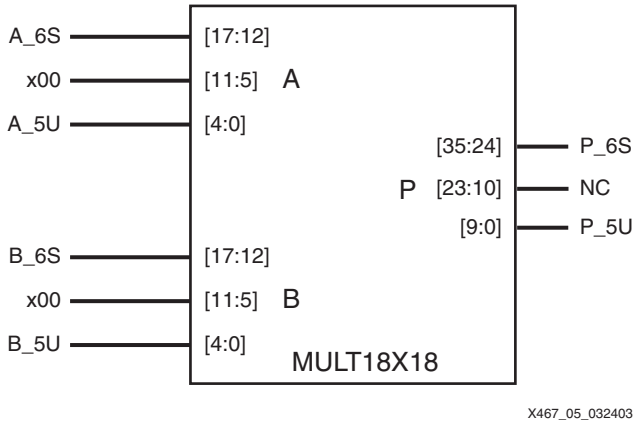



*Figure 7:* **Two Multipliers in One Primitive**

## Design Entry

There are many options for including the Spartan-3 multiplier in a design. The library primitives MULT18X18 and MULT18X18S described earlier can be instantiated in the schematic or HDL code. Synthesis tools can infer a multiplier block from the multiply operator, including Xilinx XST, Synplicity Synplify, and Mentor LeonardoSpectrum. They will infer the MULT18X18S when the operation is controlled by a clock for a synchronous multiplier.

LeonardoSpectrum features a pipeline multiplier that involves putting levels of registers in the logic to introduce parallelism and, as a result, use CLB resources instead of the dedicated multipliers. A certain construct in the input RTL source code description is required to allow the pipelined multiplier feature to take effect. See the Synthesis and Simulation Design Guide for more information.

The following VHDL example will infer the MULT18X18S using XST or Synplify:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity mult18x18s is
  port ( a : in std_logic_vector(7 downto 0);
         b : in std_logic_vector(7 downto 0);
         clk : in std_logic;
         prod : out std_logic_vector(15 downto 0));
end mult18x18s;
architecture arch_mult18x18s of
```

```
  mult18x18s is
begin
process(clk) is begin
  if clk'event and clk = '1' then
    prod <= a*b;
  end if;
end process;
end arch_mult18x18s;
```

The following is a Synchronous Multiplier VHDL Example coded for LeonardoSpectrum:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity mult18x18s is
  port(  clk: in std_logic;
         a: in std_logic_vector(7 downto 0);
         b: in std_logic_vector(7 downto 0);
         prod: out std_logic_vector(15 downto 0));
end mult18x18s;
architecture arch_mult18x18s of
  mult18x18s is
signal reg_prod : std_logic_vector(15 downto 0);
begin
process(clk)
begin
  if(rising_edge(clk))then
    reg_prod <= a * b;
    prod <= reg_prod;
  end if;
end process;
end arch_mult18x18s;
```

The following is a Synchronous Multiplier Verilog Example coded for Synplify and XST:

```
module mult18x18s(a,b,clk,prod);
  input [7:0] a;
  input [7:0] b;
  input clk;
  output [15:0] prod;
  reg [15:0] prod;
  always @(posedge clk) prod <= a*b;
endmodule
```

The following is a Synchronous Multiplier Verilog Example coded for LeonardoSpectrum:

```
module mult18x18s (a,b,clk,prod);
  input [7:0] a;
  input [7:0] b;
  input clk;
  output [15:0] prod;
  reg [15:0] reg_prod, prod;
  always @(posedge clk) begin
  reg_prod <= a*b;
  prod <= reg_prod;
endmodule
```

## Using the CORE Generator System

Multipliers that make use of the embedded Spartan-3 18-bit x 18-bit two's-complement multipliers can be easily generated using v6.0 of the CORE Generator Multiplier module. This core is available with version 5.1i and later of the CORE Generator system. Features of the Multiplier Generator include:

- Generates parallel multipliers using the dedicated multiplier blocks
  - Also can use other resources for parallel multipliers or generate sequential/serial-sequential, and fixed/reloadable constant coefficient multipliers
- Supports two's-complement signed/unsigned modes
- Supports inputs ranging from 1 to 64 bits wide
- Supports outputs ranging from 1 to 129 bits wide
- Generates purely combinatorial and fully pipelined implementations
- Provides optional registered output with optional clock enable and asynchronous and synchronous clears
- Provides optional handshaking signals

Figure 8 shows the logic symbol for the Core Multiplier Generator. The RFD (Ready For Data) output goes High to indicate the multiplier is ready to accept data. The ND (New Data) input can be asserted to indicate new data is available on the multiplier inputs. The RDY (Ready) signal indicates that the output is the current product. LOADB and SWAPB are used in constant coefficient multipliers.

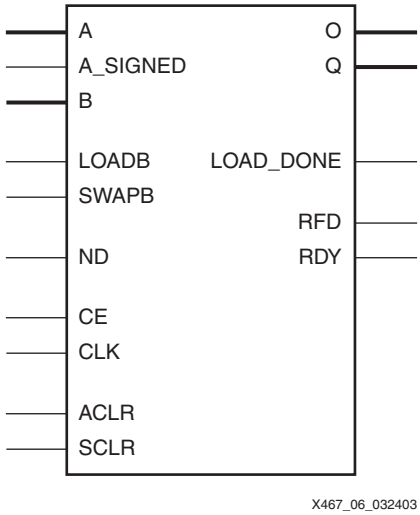



*Figure 8:* **Core Multiplier Generator Symbol**

The CORE Generator system uses the embedded multiplier for the default Parallel multiplier type. The Multiplier Construction option gives the user the choice to implement the function in look-up tables instead.

Figure 9 shows the timing diagram for the Multiplier Generator.

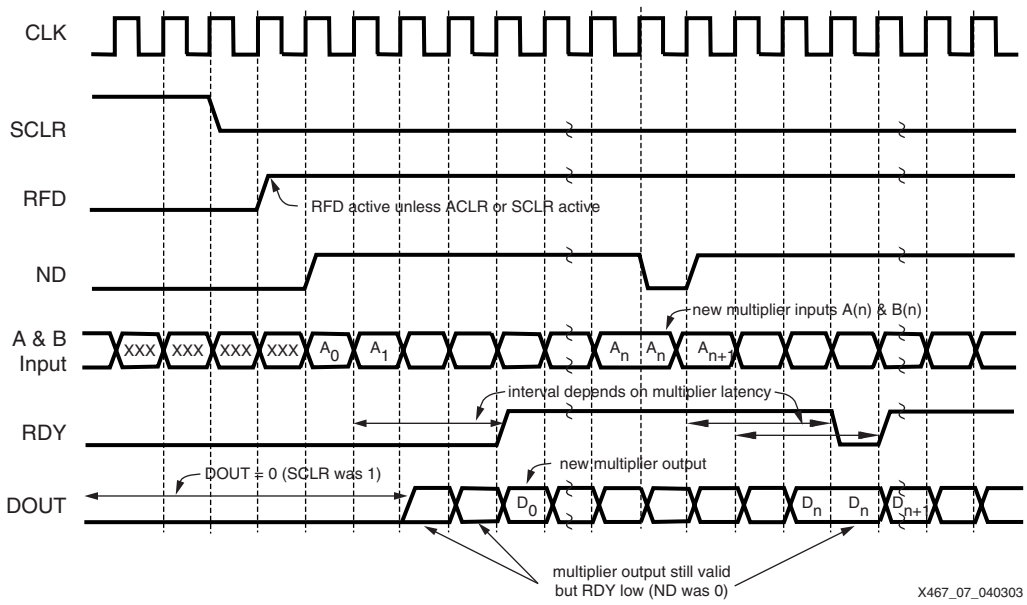



*Figure 9:* **Multiplier Generator Timing Diagram**

## System Generator

The Multiplier Generator is used by the System Generator for DSP when the MULT block is used. System Generator presents a high level and abstract view of the design, but also exposes key features in the underlying silicon, making it possible to build extremely high-performance FPGA implementations. The System Generator also provides blocks for compiling MATLAB® M-code into synthesizable HDL code. The System Generator uses the embedded multiplier when a parallel multiplier is selected and the use of the dedicated multiplier is checked in the System Generator interface.

## MAC Cores

The CORE Generator system and the System Generator can also implement more complex functions using the multiplier as a building block. The Multiply Accumulator (MAC) core supports up to 32-bit inputs and optional user-defined pipelining. The options of an Embedded or LUT Based implementation control whether the dedicated multipliers or CLB resources are used for the function. The MAC implementation uses relatively few CLB resources beyond the dedicated multipliers and provides flexibility that is key to matching a design to the lowest density and lowest cost solution possible.

The MAC and MAC-based FIR filters include an automatic pipeline control which is based on required system clock performance. Levels of pipeline will automatically be inserted based on the design requirement for a perfect speed/area trade-off.

## Multiplier Submodules

This section describes several example submodules that can be used in a Spartan-3 design. Table 4 lists multipliers and two's-complement return functions that utilize one MULT18X18 primitive and are not registered.

*Table 4:* **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 |
| TWOS_CMP18 | 18 | - | 18 | - |
| TWOS_CMP9 | 9 | - | 9 | - |
| MAGNTD_18 | 18 | - | 17 | - |

Figure 10 and Figure 11 represent 4-bit by 4-bit signed multiplier and 4-bit by 4-bit unsigned multiplier implementations, respectively.

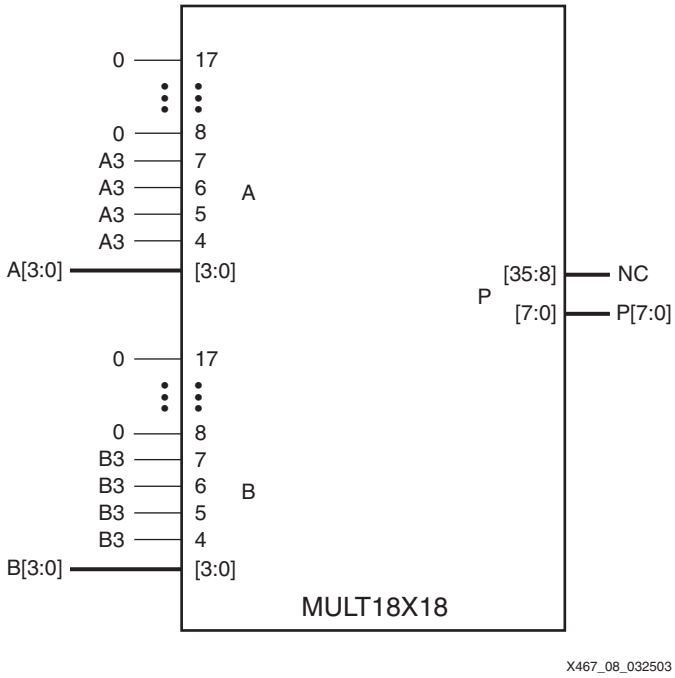



*Figure 10:* **MULT4X4_S Submodule**

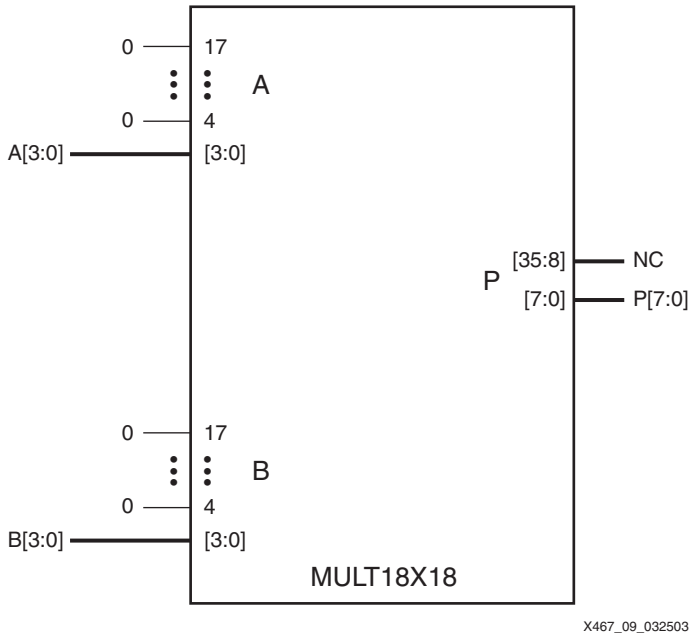



*Figure 11:* **MULT4X4_U Submodule**

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. Additional slice logic can be used with these submodules to efficiently convert sign-magnitude to two's-complement or vice-versa.

Figure 12 shows the connections to a MULT18X18 to create the submodule TWOS_CMP9.

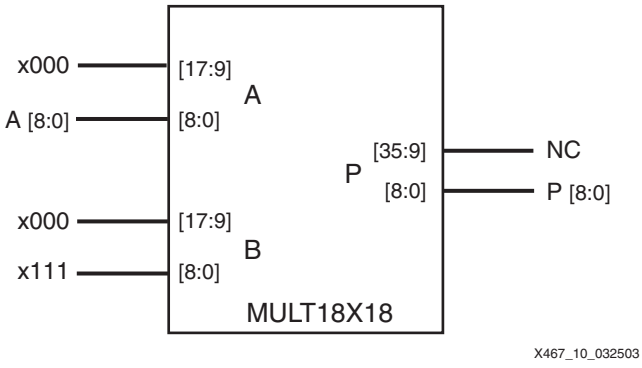



*Figure 12:* **TWOS_CMP9 Submodule**

## VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available as examples of primitives and submodules (see **VHDL and Verilog Templates**, page 279).

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.

## Port Signals

### Data In — A

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

### Data In — B

The data input B 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

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:

- 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)
- 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: Spartan-3 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: Spartan-3 Family
//-----------------------------------------------------------------------
// Instantiation Section
//
MULT18X18 U_MULT18X18
  (
    .A () , // insert input signal #1
    .B () , // insert input signal #2
    .P ()   // insert output signal
  );
```

## Alternative Applications to Multiplication

Since binary multiplication by $2^n$ is the same as shifting the value n places, a multiplier can be used as a shifter or other general-purpose resource. These can be considered in applications that otherwise would not need the large number of available multipliers.

### Shifter

A multiplier can be used as a shifter. One operand is routed to the output, shifted by n positions, if the other operand is a power of two ($2^n$). Since the sign-bit (MSB) cannot be used to control the shift, the 18x18 two's-complement multiplier can shift by 0 to 16 positions.

Of the 36 output lines, those less significant than the shifted data lines are automatically filled with zeros; those more significant than the shifted data are filled with zeros or ones, depending on the state of the MSB input. This is the natural result of the two's-complement multiplication.

The user can either perform a logic shift of 17 input bits by holding the MSB input Low, or perform an arithmetic shift of an 18-bit two's-complement number, effectively sign-extending the MSB.

A conventional CLB-based shifter would use an array of n multiplexers, each with n inputs, and require a large amount of routing resources. Multiplier-based shifters larger than 18 bits, and barrel shifters of any length, require external OR gating of the outputs, but use far fewer CLB resources.

## Magnitude Return

To generate the absolute value of a number by using multiplication, multiply by 1 if it is positive (MSB is zero), and multiply by -1 if it is negative (MSB is one). In two's-complement notation, 1 is all zeros ending in a one as the LSB, and -1 is all ones, including the LSB. Therefore, a magnitude return or absolute value generator can be implemented by multiplying by a value with a one as the LSB and the MSB of the input value in all the other bit positions. Figure 13 shows a magnitude return generator.

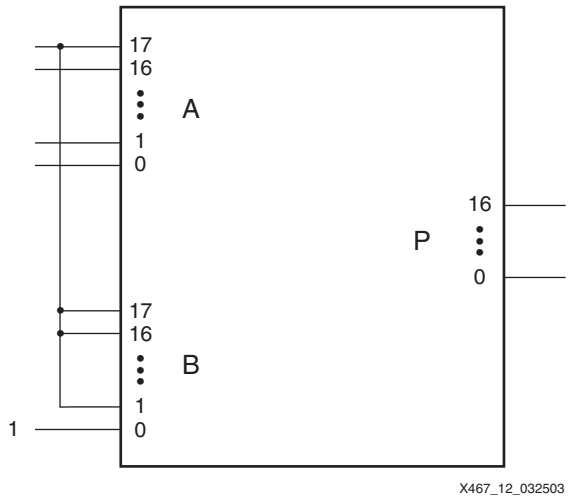



*Figure 13:* **Magnitude Return**

## Two's-Complement Return

Generating the two's complement of a number typically requires only one LUT per bit with the carry logic used for larger numbers. However, if LUTs are heavily used, the multiplier can be used to return the two's complement of the input. Multiplying an input number by an equivalent length number of all ones generates the two's complement of the number over the same length of the output bits. Any extraneous higher-order bits are ignored. Figure 14 shows a two's complement return generator.

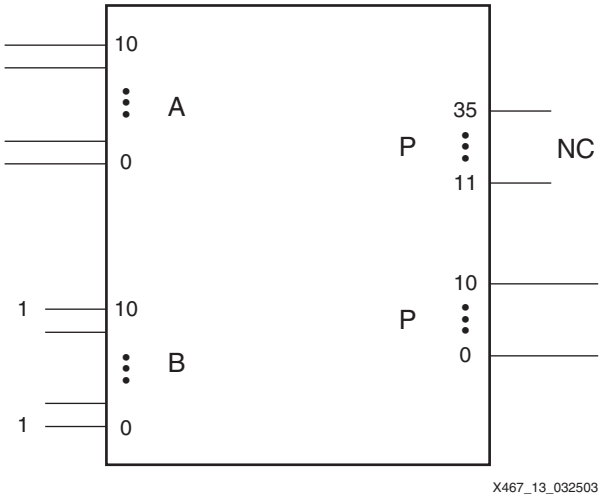



*Figure 14:* **Two's-Complement Return**

## Complex Multiplication

Complex multiplication is multiplication of complex numbers, which contain real and imaginary components with the imaginary unit i equal to the square root of -1. Complex multiplication can be carried out using only three real multiplications: ac, bd, and (a + b)(c + d). The real part of (a + ib)(c + id) is ac - bd, and the imaginary part is (a + b)(c + d) - ac - bd. The large number of multipliers in the Spartan-3 architecture makes it convenient to do even complex multiplication.

## Time Sharing in Matrix Multiplication

Many pipelined functions in the computer graphics and video fields are expressed in matrix mathematics. A 3 x 3 matrix multiplication would require 27 multiplies and 18 adds to generate the 3 x 3 matrix result. Color conversion can be described as a 3 x 3 matrix multiplication by a constant, which requires nine multiplies and six adds to generate the three results.

The high-speed capability of a Spartan-3 device allows the user to "time share" the multipliers. Instead of nine multipliers, the design feeds nine sets of inputs resulting in nine sets of results at nine times the clock rate of the system, reducing the multiplier count to one. The adder logic is implemented in CLB resources, and at every third clock, the adder output is stored in output registers to capture the three results. See XAPP284 for more information.

## Floating-Point Multiplication

Floating-point values add an exponent to the number and sign bit used in binary multiplication. A 32-bit floating-point multiplier can be implemented using four of the dedicated multiplier blocks and CLB resources. Such multipliers are available from Xilinx AllianceCORE™ partners.

## Related Materials and References

- Spartan-3 Family Data Sheet
  Architectural description and timing parameters.
  **DS099-1**, ***Spartan-3 1.2V FPGA Family:*** **Introduction and Ordering Information** (Module 1)
  **DS099-2**, ***Spartan-3 1.2V FPGA Family:*** **Functional Description** (Module 2)
  **DS099-3**, ***Spartan-3 1.2V FPGA Family:*** **DC and Switching Characteristics** (Module 3)
  **DS099-4**, ***Spartan-3 1.2V FPGA Family:*** **Pinout Tables** (Module 4)
- DSP Central (**http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=Xilinx+DSP**)
  Information that will enable you to achieve the maximum benefit from our DSP solutions.
- IP Center (**http://www.xilinx.com/ipcenter**)
  Xilinx and Alliance partner core solutions.
- Xilinx Software Documentation
  (**http://www.xilinx.com/support/sw_manuals/xilinx5/download/**)
  Libraries Guide MULT18X18/S descriptions, Synthesis and Simulation Design Guide instantiation examples for HDL.
- **XAPP284**: *Matrix Math, Graphics, and Video*
  Uses one multiplier running at 9x the clock rate to provide the nine results for a 3x3 matrix multiplication in one system clock cycle.
- **XAPP636**: *Optimal Pipelining of the I/O Ports of Virtex-II Multipliers*
  Describes a high-speed, optimized implementation of the dedicated multiplier resulting from pipelined inputs and outputs and effective placement and routing constraints.
- TechXclusives (**http://www.xilinx.com/support/techxclusives/techX-home.htm**)
  See "Using Leftover Multipliers and Block RAM" by Peter Alfke and "Expanding Virtex-II Multipliers" by Ken Chapman.

## Conclusion

FPGAs have a significant advantage over general-purpose DSP chips because their logic can be customized for the specific application. Some functions can run over 100 times faster and require much less expense in an FPGA. A key feature to take advantage of is the dedicated multiplier block. Take advantage of the automatic optimization of multiplication logic, and the user controls when necessary to get the exact results desired. The CORE Generator system can create simple multipliers or combine them into more complex functions such as MACs.

## Appendix A: Two's-Complement Multiplication

Two's-complement representation allows the use of binary arithmetic operations on signed integers, yielding the correct two's-complement results. Positive two's-complement numbers are represented as simple binary. Negative two's-complement numbers are represented as the binary number that when added to a positive number of the same magnitude equals zero. To calculate the two's complement of an integer, invert the binary equivalent of the number by changing all of the ones to zeros and all of the zeros to ones (also called one's complement), and then add one. The MSB (left-most) bit indicates the sign of the integer; therefore it is sometimes called the sign bit. If the sign bit is zero, the number is positive. If the sign bit is one, the number is negative. To extend a signed integer to a larger width, duplicate the MSB on the left side of the number.

Two's-complement multiplication follows the same rules as binary multiplication, which are the same as the truths of the AND gate:

0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1, and no carry or borrow bits

For example,

1111 1100 = -4

× 0000 0100 = +4

1111 0000 = -16

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 04/06/03 | 1.0 | Initial Xilinx release. |
| 05/13/03 | 1.1 | Updated multiplier information for the XC3S50 device in the **Multipliers in the Spartan-3 Architecture** section.<br>Added new section entitled **Expanding Multipliers**.<br>Added TechXclusives reference to **Related Materials and References** section.<br>Made minor edits for clarification. |

# Spartan-3 Design Software and IP Cores

## Using the ISE Design Tools for Spartan-3 FPGAs

## Using Spartan-3 IP Cores

## Embedded Processing and Control Solutions for Spartan-3 FPGAs


SPARTAN™-3
FPGA
MAKE IT YOUR ASIC

XAPP473 (v1.0) July 11, 2003

# Using the ISE Design Tools for Spartan-3 FPGAs


## Summary

Software is critical to the effective use of programmable logic. The Spartan™-3 family is supported by the complete set of Xilinx Integrated Software Environment (ISE) development tools, with additional support available from a variety of partners. This document provides an overview of those design tools. It is intended primarily for the user who is new to the Xilinx development system. This document can be used to get a better understanding of the specific tools mentioned elsewhere in the Spartan-3 literature. The first half provides an overview of the general design flow, while the second half describes the specific tools used at the different steps in the flow. Use the Xilinx development system documentation for detailed information and introductory tutorials.


## Introduction

Combined with the Spartan-3 FPGA family, ISE's optimized design tools help you finish faster and lower your project costs. The ISE package is a collection of Xilinx software design tools that concentrate on delivering the most productivity available for your Spartan-3 logic performance. With ProActive Timing Closure technology, you get the fastest runtimes in programmable logic ensuring you reach your performance goals quicker. Incremental Design delivers faster re-compile times with guaranteed performance, and the optional Xilinx ChipScope™ Pro verification tools provide real-time debug with advantages that are not possible in ASIC designs. ISE makes sure you get through the logic design process faster, saving both time and project costs, and getting you to market ahead of your competition.

## Design Flow

The standard design flow for Spartan-3 FPGAs consists of the following three major steps. The entire design implementation flow is run simply by selecting the desired result in the Xilinx Graphical User Interface (GUI). The tools automatically determine which programs and files are needed to bring the appropriate output up to date.

1. Design Entry and Synthesis

   In this step of the design flow, you create your design using a Xilinx-supported schematic editor, a Hardware Description Language (HDL) for text-based entry, or both. If you use an HDL for text-based entry, you must synthesize the HDL file into an industry-standard Electronic Data Interchange Format (EDIF) file. If you use the Xilinx Synthesis Technology (XST) tool, a Xilinx-specific NGC netlist file is created, which can be converted to an EDIF file.

2. Design Implementation

   By implementing the specific Xilinx Spartan-3 architecture, you convert the logical design file format, such as EDIF, that you created in the design entry or synthesis stage into a physical file format. The physical information is contained in the Native Circuit Description (NCD) file. Then you create a bitstream file from these files and optionally program a PROM for subsequent programming of your Spartan-3 device.

3. Design Verification

Using a gate-level simulator, you ensure that your design meets your timing requirements and functions properly. In-circuit verification can be performed by downloading your design to the device using Xilinx iMPACT Programming Software. Design verification can begin immediately after design entry and can be repeated after various steps of design implementation.

Figure 1 shows the general overall design flow for Spartan-3 FPGAs.

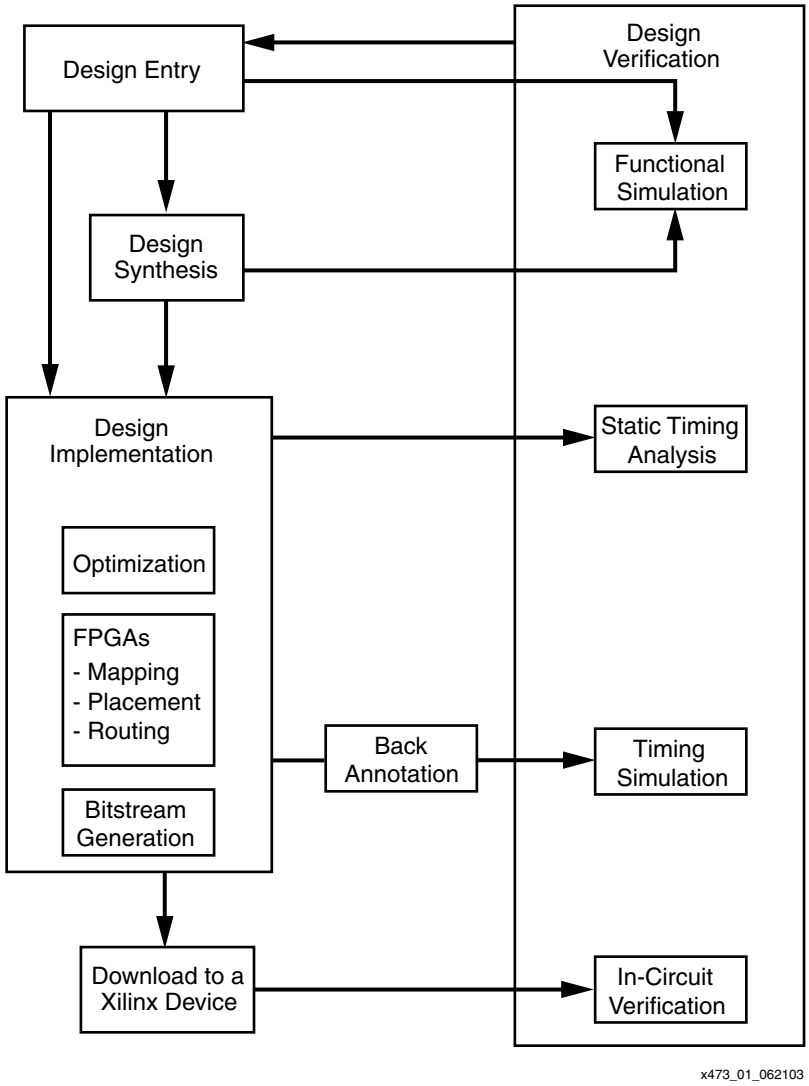



*Figure 1:* **Design Flow**

## Design Entry and Synthesis

You can enter a design with a schematic editor or a text-based tool for HDL code. Design entry begins with a design concept, expressed as a drawing or functional description. From the original design, a generic EDIF netlist is created, then synthesized and translated into a Xilinx netlist file. This file is fed into a program called NGDBuild, which produces a logical Native Generic Database (NGD) file. Xilinx libraries provide access to features specific to the Spartan-3 architecture.

Figure 2 shows the design entry and synthesis flow.

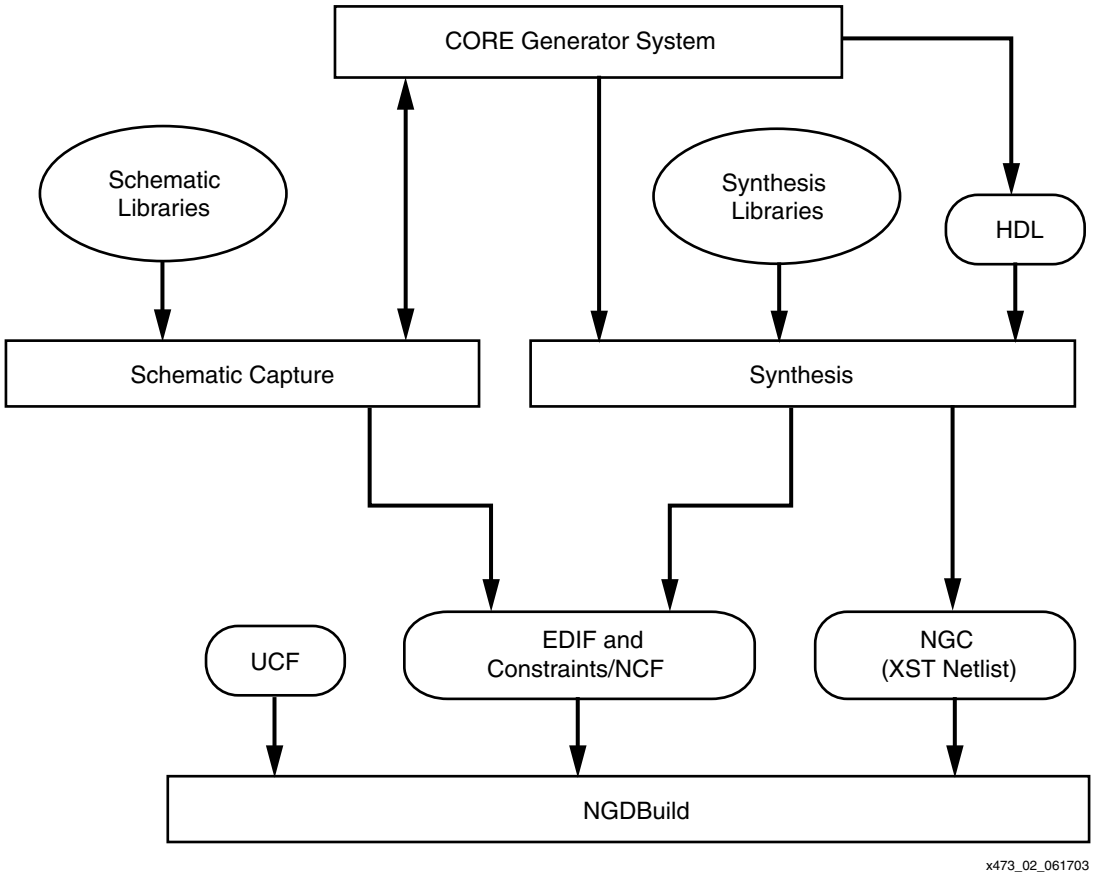



*Figure 2:* **Design Entry and Synthesis Flow**

### Hierarchical Design

Design hierarchy is important in both schematic and HDL entry for the following reasons:

- Helps you conceptualize your design
- Adds structure to your design
- Promotes easier design debugging
- Makes it easier to combine different design entry methods (schematic, HDL, or state editor) for different parts of your design
- Makes it easier to design incrementally, which consists of designing, implementing, and verifying individual parts of a design in stages
- Reduces optimization time
- Facilitates concurrent design, which is the process of dividing a design among a number of people who develop different parts of the design in parallel, such as in Modular Design

Xilinx strongly recommends that you name the components and nets in your design. These names are preserved and used by the Xilinx tools. These names are also used for back-annotation and appear in the debug and analysis tools. If you do not name your components and nets, the tools automatically generate the names, making it difficult to analyze circuits.

### Schematic Entry

Schematic tools provide a graphical interface for design entry. You can use these tools to connect symbols representing the logic components in your design. You can build your design with individual gates, or you can combine gates to create functional blocks.

Primitives and macros are the “building blocks” of a device library. The Xilinx Spartan-3 library provides primitives as well as common high-level macro functions, all optimized for the Spartan-3 architecture. Primitives are basic circuit elements, such as AND and OR gates, and special device resources, such as the DCM and block RAM. Each primitive has a unique library name, symbol, and description.

Macros contain multiple library elements, which can include primitives and other macros. Soft macros have pre-defined functionalities, but have flexible mapping, placement, and routing. Relationally Placed Macros (RPMs) have fixed mapping and relative placement. Macros are not available for synthesis because synthesis tools have their own module generators and do not require RPMs. If you wish to override the module generation, you can instantiate Xilinx-provided CORE Generator™ modules, which include pre-built optimization for the Spartan-3 architecture. For most leading-edge synthesis tools, this is not needed unless it is for a module that cannot be inferred.

### HDL Entry and Synthesis

A typical Hardware Description Language (HDL) supports a mixed-level description in which gate and netlist constructs are used with functional descriptions. This mixed-level capability enables you to describe system architectures at a high level of abstraction, then incrementally refine a design’s detailed gate-level implementation. HDL descriptions offer the following advantages:

- You can verify design functionality early in the design process. A design written as an HDL description can be simulated immediately. Design simulation at this high level — at the gate-level before implementation — allows you to evaluate architectural and design decisions.
- An HDL description is more easily read and understood than a netlist or schematic description. HDL descriptions provide technology-independent documentation of a design and its functionality. Because the initial HDL design description is technology independent, you can use it again to generate the design in a different technology, without having to translate it from the original technology.
- Large designs are easier to handle with HDL tools than schematic tools.

After creating your HDL design, you must synthesize it. During synthesis, behavioral information in the HDL file is translated into a structural netlist, and the design is optimized for the Spartan-3 architecture. Xilinx supports HDL synthesis tools for several third-party synthesis vendor partners. In addition, Xilinx offers its own synthesis tool, Xilinx Synthesis Technology (XST).

Functional simulation tests the logic in your design to determine if it works properly. You can save time during subsequent design steps if you perform functional simulation early in the design flow.

Although HDL entry offers the advantage of technology independence, it is helpful to understand the available resources in the Spartan-3 architecture and design to take advantage of those resources. For example, the abundance of registers at every I/O and following every look-up table encourages pipelining. Most synthesis tools automatically infer Xilinx-specific resources and optimize for the architecture. Simple ways to specify implementation requirements are to instantiate Spartan-3 library components or add constraints.

### Constraints

You might want to constrain your design within certain timing or placement parameters to specify your required pin locations or timing requirements. You can specify logic mapping, block placement, and timing specifications. Constraints can be entered as parameters or attributes on library components. You can enter constraints by hand or use one of several graphical tools for generating constraint files and evaluating the results. Constraints found in the design are written to an NCF file (Netlist Constraints File). Constraints created separately are written to a UCF file (User Constraints File).

## Design Implementation

Design Implementation begins with the translating, then mapping, of a logical design file to a specific Spartan-3 device; it is complete when the physical design is successfully routed and a bitstream is generated. You can alter constraints during implementation in the same way as during the Design Entry step.

Figure 3 shows an overall view of the design implementation flow for Spartan-3 FPGAs.

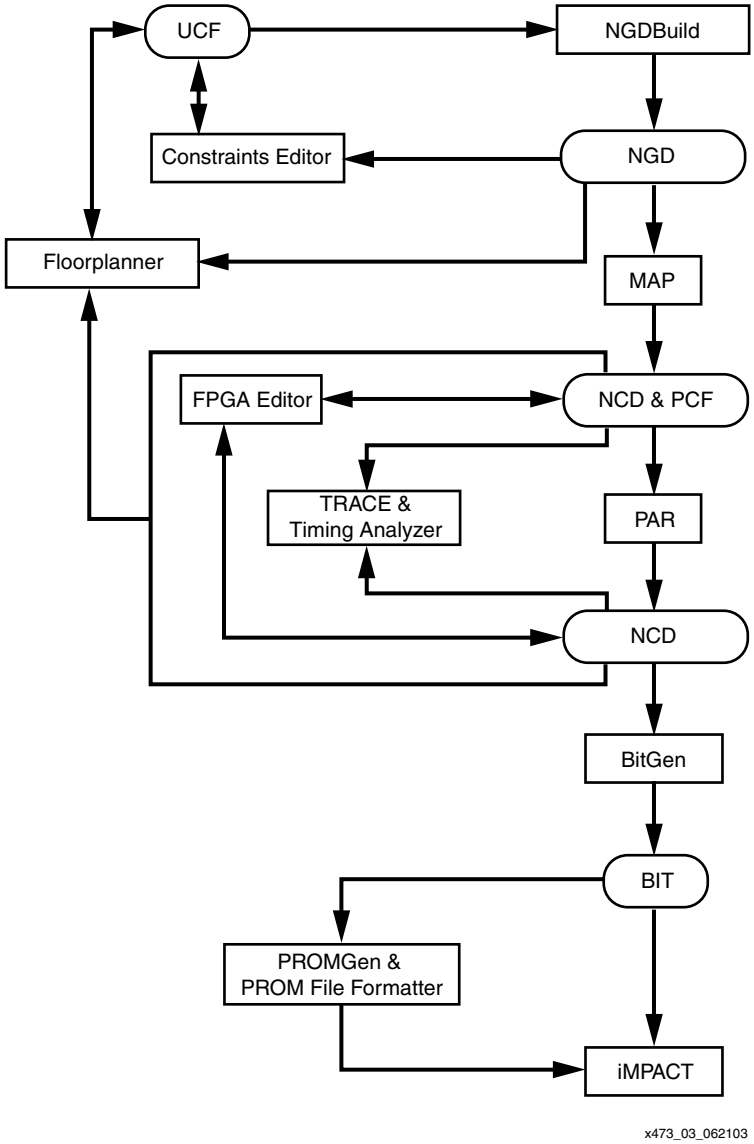



*Figure 3:* **Design Implementation Flow**

### Translating

NGDBuild performs all the steps necessary to read a netlist file in EDIF or NGC format and create an NGD file describing the logical design. A logical design is in terms of logic elements, such as AND gates, OR gates, decoders, flip-flops, and RAMs. The NGD file resulting from an NGDBuild run contains both a logical description of the design reduced to Xilinx primitives and a description in terms of the original hierarchy expressed in the input netlist. The output NGD file then can be mapped to the Spartan-3 device family resources.

NGDBuild performs the following steps to convert a netlist to an NGD file:

1. Reads the source netlist(s). NGDBuild invokes the Netlister Launcher. The Netlist Launcher determines the type of the input netlist and starts the appropriate netlist reader program. The netlist readers incorporate NCF files associated with each netlist. NCF files contain timing and layout constraints for each module.
2. Reduces all components in the design to NGD primitives. NGDBuild merges components that reference other files. NGDBuild also finds the appropriate system library components, physical macros, and behavioral models.
3. Checks the design by running a Logical Design Rule Check (DRC) on the converted design. The Logical DRC is a series of tests on the logical design.
4. Writes an NGD file as output.

### Mapping

The MAP program maps a logical design to a Spartan-3 FPGA. The input to MAP is an NGD file, which contains a logical description of the design in terms of both the hierarchical components used to develop the design and the lower-level Xilinx primitives. Additionally, it contains any number of hard placed-and-routed physical macro files. MAP then maps the logic to the components (logic cells, I/O cells, and other components) in the Spartan-3 architecture. The output design is a Native Circuit Description (NCD) file, which is a physical representation of the design mapped to the components in the Spartan-3 architecture. The NCD file then can be placed and routed.

MAP performs the following steps when mapping a design:

1. Selects the target Xilinx device, package, and speed.
2. Reads the information in the input design file.
3. Performs a Logical DRC (Design Rule Check) on the input design. If any DRC errors are detected, the MAP run is aborted. If any DRC warnings are detected, the warnings are reported, but MAP continues to run.
4. Removes unused logic, where all unused components and nets are removed.
5. Maps pads and their associated logic into IOBs.
6. Maps the logic into Xilinx components (IOBs, CLBs, etc.). If any Xilinx mapping control symbols appear in the design hierarchy of the input file, MAP uses the existing mapping of these components in preference to re-mapping them. The mapping is influenced by various constraints.
7. Updates the information received from the input NGD file and writes this updated information into an NGM file. This NGM file contains both logical information about the design and physical information about how the design was mapped. The NGM file is used only for back-annotation.
8. Creates a physical constraints (PCF) file. This text file contains any constraints specified during design entry. If no constraints were specified during design entry, an empty file is created so that you can enter constraints directly into the file using a text editor.
9. Runs a physical Design Rule Check (DRC) on the mapped design. If DRC errors are found, MAP does not write an NCD file.
10. Creates an NCD file, which represents the physical design. The NCD file describes the design in terms of Xilinx components (CLBs, IOBs, and so forth).
11. Writes a MAP report (MRP) file, which lists any errors or warnings found in the design, details how the design was mapped, and supplies statistics about component usage in the mapped design.

### Placing and Routing

After creating a mapped NCD file, you can place and route the file using the automatic Place And Route (PAR) tool. PAR accepts an NCD file as input, places and routes the design, and outputs an NCD file to be used by the bitstream generator (BitGen). You can use the output NCD file as a guide file for additional runs of PAR after making minor changes to your design.

PAR places and routes a design based on the following considerations:

- Cost-Based: Placement and routing are performed using various cost tables that assign weighted values to relevant factors such as constraints, length of connection, and available routing resources.
- Timing-Driven: The Xilinx timing analysis software enables PAR to place and route a design based upon your timing constraints.

#### *Placing*

The PAR placer executes multiple phases of the placer. PAR writes the NCD after all the phases are completed. During placement, PAR places components into sites based on factors such as constraints specified in the PCF file, the length of connections, and the available routing resources. Timing-driven placement is automatically invoked if PAR finds timing constraints in the physical constraints file.

#### *Routing*

The next stage is routing the placed design. PAR writes the NCD file when the design is fully routed. There still may exist unroutes since power and ground are not considered. At this point the design can be analyzed against timing. A new NCD will be written as the routing improves. The router performs a procedure to converge on a solution that routes the design to completion and meets timing constraints. Timing-driven routing is automatically invoked if PAR finds timing constraints in the physical constraints file.

#### *Floorplanning*

Floorplanning is the process of specifying user placement constraints. The Floorplanner provides a graphical view of placement, while the FPGA Editor provides a graphical view of both placement and routing. Both tools can be used before or after PAR to analyze or constrain the design.

### Bitstream Generation

After the design has been completely routed, it is necessary to configure the device so that it can execute the desired function. This configuration is done using files generated by BitGen, the Xilinx bitstream generation program. BitGen takes a fully routed NCD file as its input and produces a configuration bitstream (binary BIT file).

The BIT file contains all of the configuration information from the NCD file defining the internal logic and interconnections of the Spartan-3 FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT file then can be downloaded into the FPGA memory cells or it can be used to create a PROM file. The PROM file is created by the PROM File Formatter, which is the GUI for the PROMGen tool.

## Design Verification

Design verification is the process of testing the functionality and performance of your design. You can verify Xilinx designs in the following ways:

- Simulation (functional and timing using back-annotation)
- Static timing analysis
- In-circuit verification

Design verification procedures should occur throughout your design process, as shown in Figure 4.

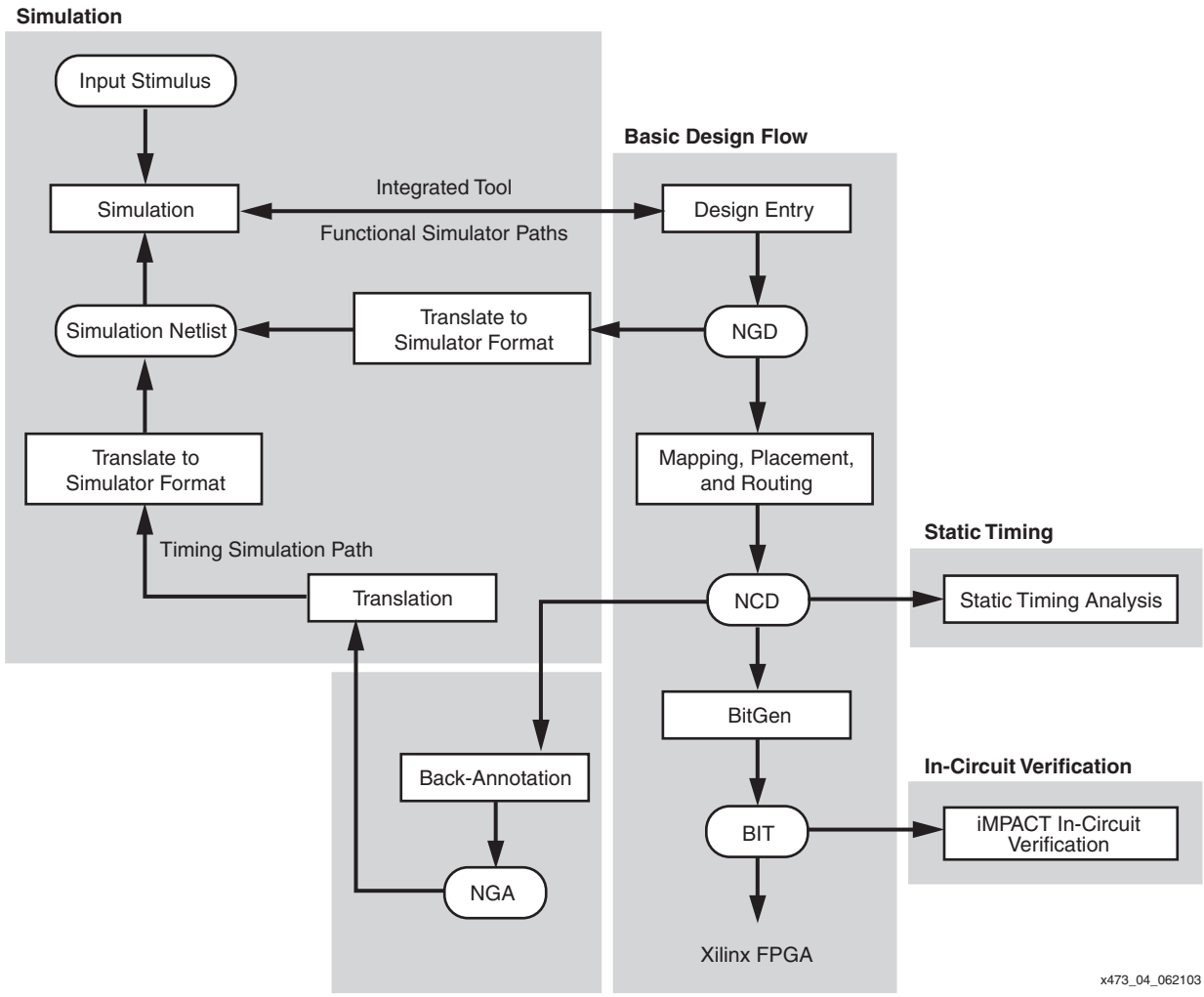



*Figure 4:* **Design Verification Flow**

### Simulation

Design simulation involves testing your design using software models. It is most effective when testing the functionality of your design and its performance under worst-case conditions. You can easily probe internal nodes to check your circuit's behavior, and then use these results to make changes in your design. Simulation is performed using third-party tools that are linked to the Xilinx Development System. The software models provided for your simulation tools are designed to perform detailed characterization of your design. You can perform functional or timing simulation.

#### *Functional Simulation*

Functional simulation determines if the logic in your design is correct before you implement it in a device. Functional simulation can take place at the earliest stages of the design flow. Because timing information for the implemented design is not available at this stage, the simulator tests the logic in the design using unit delays.

### *Timing Simulation*

Timing simulation verifies that your design runs at the desired speed for your device under worst-case conditions. This process is performed after your design is mapped, placed, and routed. At this time, all design delays are known. Timing simulation is valuable because it can verify timing relationships and determine the critical paths for the design under worst-case conditions. It also can determine whether or not the design contains setup or hold violations. Before you can simulate your design, you must go through the back-annotation process, as described below. During this process, the Xilinx netlist writers create suitable formats for various simulators.

Note that naming the nets during your design entry is important for both functional and timing simulation because it allows you to find the nets in the simulations more easily than looking for a software-generated name.

### *Back-Annotation*

Before timing simulation can occur, the physical design information must be translated and distributed back to the logical design. This back-annotation process is done with a program called NGDAnno. These programs create a database for the netlist writers, which translate the back-annotated information into a netlist format that can be used for timing simulation.

NGDAnno is a command line program that distributes information about delays, setup and hold times, clock to out, and pulse widths found in the physical NCD design file back to the logical NGD file. NGDAnno reads an NCD file as input. The NCD file can be a mapped-only design, or a partial or fully placed and routed design. An NGM file, created by MAP, is an optional source of input. NGDAnno merges mapping information from the NGM file with placement, routing, and timing information from the NCD file. NGDAnno outputs a Native Generic Annotated (NGA) file, which is a back-annotated NGD file. This file is input to the appropriate netlist writer, which converts the binary Xilinx database format back to an ASCII netlist.

Netlist Writers (NGD2EDIF, NGD2VER, or NGD2VHDL) take the output of NGDAnno and create a simulation netlist in the specified format. An NGD or NGA file is input to each of the netlist writers. The NGD file is a logical design file containing primitive components, while the NGA file is a back-annotated logical design file.

### Static Timing Analysis

Static timing analysis is best for quick timing checks of a design after it is placed and routed. It also allows you to determine path delays in your design. Following are the two major goals of static timing analysis:

- Timing verification is the process of verifying that the design meets your timing constraints.
- Reporting is the process of enumerating input constraint violations and placing them into an accessible file. You can analyze partially or completely placed and routed designs. The timing information depends on the placement and routing of the input design.

You can run static timing analysis using the Timing Reporter And Circuit Evaluator (TRACE) program, which is accessible through the Timing Analyzer GUI. Use either tool to evaluate how well the place and route tools met the input timing constraints.

### In-Circuit Verification

As a final test, you can verify how your design performs in the target application. In-circuit verification tests the circuit under typical operating conditions. Because you can program your Xilinx devices repeatedly, you can easily load different iterations of your design into your device and test it in-circuit. To verify your design in-circuit, download your design bitstream into a device using the iMPACT programming software with the Parallel Cable IV or MultiPRO cable.

# ISE Development Environment

## Introduction to ISE

Xilinx development systems are available in a number of easy to use configurations, collectively known as the Integrated Software Environment (ISE) Series. Creating Spartan-3 designs is easy with Xilinx ISE development systems, which support advanced design capabilities, including ProActive Timing Closure, integrated logic analysis, and the fastest place and route runtimes in the industry. ISE solutions enable designers to get the performance they need, quickly and easily.

***Note:*** To get the full details on ISE tools for Spartan-3 devices, go to http://www.xilinx.com/ise/ise_promo/ise5_spartan3.htm.

Project Navigator is the user interface that helps you manage the entire design process including design entry, simulation, synthesis, implementation and finally configuration of your device.

The following is an outline of the features offered in ISE:

**Design Entry**

- HDL Editor
- StateCAD® State Machine Editor
- Schematic Editor - Engineering Capture System (ECS)
- CORE Generator system

**Synthesis**

- XST - Xilinx Synthesis Technology
- Integration with LeonardoSpectrum synthesis from Mentor Graphics
- Integration with Synplify/Pro and Amplify synthesis from Synplicity

**Simulation**

- HDL Bencher™ Testbench Generator
- Integration with ModelSim Simulator from Model Technology

**Implementation**

- Translate
- Map
- Place and Route (PAR)
- Floorplanner
- FPGA Editor
- Timing Analyzer
- XPower Power Analysis

**Device Download**

- BitGen Bitstream Generator
- PROMGen PROM File Formatter
- iMPACT Configuration Tool
- ChipScope Pro Logic Analyzer

## ISE Versions

The ISE development systems are available in the following configurations.

- ISE WebPACK™ Tool

  The ISE WebPACK tool is the easiest development system to get. This free tool is downloadable from the Web at: (http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=ISE+WebPack).

  ISE WebPACK software combines support for advanced HDL entry, synthesis, and verification capabilities for all Xilinx CPLDs and lower-density FPGAs.

- ISE BaseX Tool

  The ISE BaseX tool is the industry's most cost-effective, PC-based programmable logic design environment. The ISE BaseX configuration provides all of the capabilities contained within the ISE WebPACK software plus additional tools like the CORE Generator system and FPGA Editor that help you complete your programmable logic design even faster.

- ISE Alliance Tool

  The ISE Alliance tool is designed to fit into your existing design environment. This tool provides full device support and works seamlessly with those tools of our EDA partners. The ISE Alliance tool is for those designers looking to add programmable logic design capabilities into their existing design environment. This tool does not include XST synthesis or ECS schematic capture.

- ISE Foundation™ Tool

  The ISE Foundation tool is a complete, ready-to-use design environment that integrates schematic, synthesis, and verification technologies into an intuitive, yet highly advanced design solution. The tool has full device support as well as the full suite of tools.

To see a table comparison of these versions, see the Development Systems Overview at http://www.xilinx.com/ise/devsys_feature_guide.pdf.

Development system updates are provided on a regular basis. These are available as Service Packs that can be downloaded from the Xilinx website (http://www.xilinx.com/support/software/install_info.htm). Always use the latest development system update for the best results.

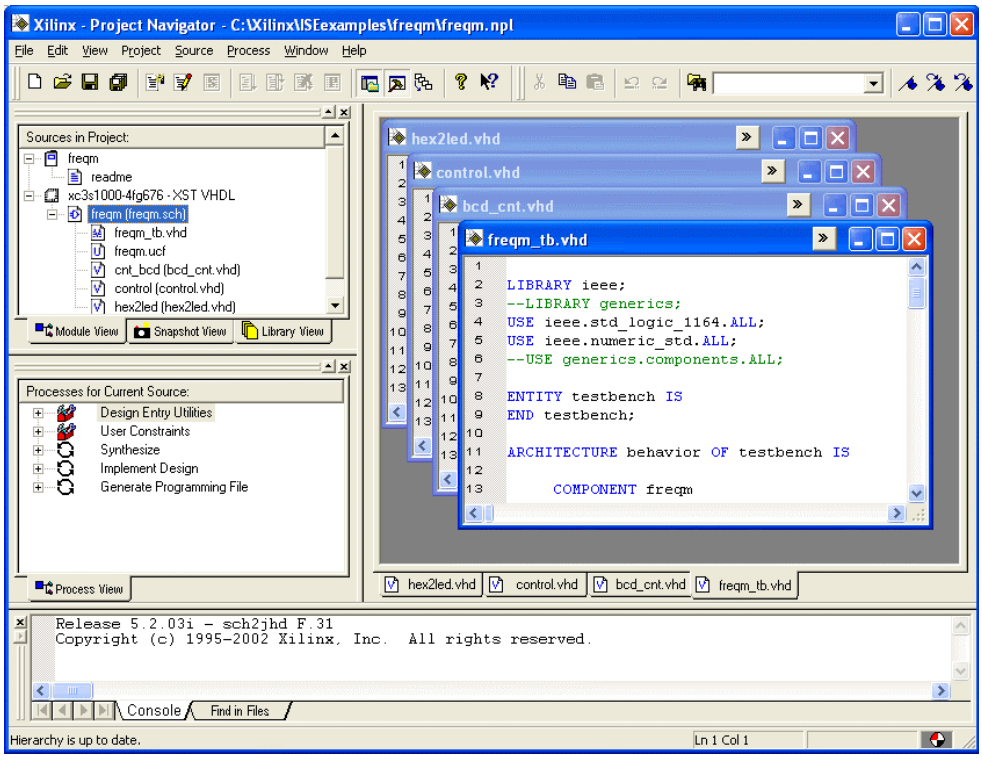
## Project Navigator

Project Navigator is the primary user interface for the Xilinx ISE tools. You can create, define, and compile your Spartan-3 design using a suite of tools accessible from Project Navigator. Each step of the design process, from design entry to downloading the design to the device, is managed from Project Navigator as part of a project. These include:

- Design Entry
- Constraint Entry
- Synthesis
- Simulation
- Implementation
- Device Programming

### Project Navigator Main Window

The Project Navigator workspace is made up of a title bar, a status bar, a menu bar, toolbars and windows.




x473_05_071103

*Figure 5:* **Project Navigator Main Window**

### Project

ISE organizes and tracks your design as a project. A project is a collection of all files necessary to create and download your design to the selected device. The following information is required for each project:

- A unique project name
- A specified target device family (architecture)
- A specified target device
- A specified design flow

Each project has a directory, device family, device, and design flow associated with it as project properties. The project properties enable Project Navigator to display and run only those processes appropriate for the targeted device and design flow.

### Sources

A source is any element that contains information about a design. In Project Navigator, you can create and add sources to your project. Each project can contain many sources, each one representing a different part of the overall design. Sources can include the description of circuits (as represented by schematics and hardware description language files), state diagrams, simulation models, test files, and documentation of the design.

### Source Hierarchy

One source file in a project is the top-level source for the design. The top-level source defines the inputs and outputs to be mapped into the device, and references the logic descriptions contained in lower-level sources in a hierarchical design. A project must contain at least one source as the top-level source. All source files and their accompanying icons are displayed in the Sources in Project window below the project file.

The term instantiation describes when one source references another. Lower-level sources also can instantiate sources to build as many levels of logic hierarchy as necessary to describe your design.

Valid top-level source types include the following:

- Schematics
- HDL files (VHDL or Verilog)
- EDIF

## ISE Tools

ISE includes a number of individual tools and capabilities that can be accessed standalone or within the Project Navigator.

### Engineering Capture System (ECS)

The Engineering Capture System (ECS) allows you to create, view, and edit schematics and symbols. You can use ECS to create a top-level schematic and use any of the following to define the lower levels of the design: ECS, CORE Generator System, or HDL code. Then you can translate the schematics created by ECS to a structural HDL for simulation and synthesis, or use the schematics solely for documentation purposes.

### HDL Editor

The HDL Editor is a text editor designed especially for editing HDL source files. In addition to regular editing features, the editor provides syntax coloring. The syntax-coloring feature supports both VHDL and Verilog. The HDL Editor operates as a standard text editor as well. ISE provides optimized, ready-to-use language and synthesis templates for easy insertion into an HDL source file.

### StateCAD State Machine Editor

StateCAD accelerates design entry by allowing quick entry of finite state machines (FSMs). Each step is automated, reducing development costs and enhancing effectiveness. StateCAD allows users to quickly design FSMs, find and fix design errors, verify behavior, and generate optimized HDL. StateCAD automatically analyzes designs for problems such as stuck-in-states, conflicting state assignments, and indeterminate conditions. This automated error analysis ensures that designs are logically consistent, reducing simulation requirements and improving product reliability.

### Xilinx Synthesis Technology (XST)

Xilinx Synthesis Technology (XST) provides cutting edge design optimization techniques from a Xilinx-developed synthesis tool. XST supports the Verilog and VHDL design languages. XST is included in ISE WebPACK, ISE BaseX, and ISE Foundation packages. RTL Viewer displays the results of XST synthesis in a schematic view.

### HDL Advisor

The HDL Advisor gives advisory messages in the XST synthesis report files. The messages are designed to make suggestions on how code can be changed to reduce design size and meet timing requirements. These HDL advisors allow designers to produce better code earlier, reducing design time, and resulting in better space utilization in the Spartan-3 FPGA.

### Partner Tools

The Xilinx tools provide easy integration with third-party tools including LeonardoSpectrum synthesis from Mentor Graphics, Synplify/Pro and Amplify synthesis from Synplicity, and FPGA Compiler II from Synopsys. These tools can be purchased separately from the vendor.

ModelSim simulators from Model Technology provide the simulation functions for ISE. ModelSim Xilinx Edition II (MXE-II) is available as an option from Xilinx. It offers a complete PC

HDL simulation environment that enables you to verify the HDL source code as well as the functional and timing models of your designs.

### Intellectual Property (IP)

Get to market faster and less expensively using the latest pre-verified, pre-optimized Intellectual Property (IP) Cores, Reference Designs, and Design Services for Xilinx FPGAs. Xilinx-created LogiCORE™ products form the most successful core program in the programmable logic industry, including PCI bus interfaces and MicroBlaze™ soft processors. As a result, Xilinx has gained considerable experience developing and selling cores, and servicing FPGA core customers. Through the AllianceCORE™ program, Xilinx is expanding the availability of quality cores for programmable logic by sharing what has been learned with leading third-party core developers. The AllianceCORE program is a cooperative effort between Xilinx and independent third-party core developers. It is designed to produce a broad selection of industry-standard solutions dedicated for use in Xilinx programmable logic. Xilinx also provides many reference designs and design examples provided “as-is” to help get you started with your own designs.

### CORE Generator System

The Xilinx CORE Generator System provides a catalog of ready-made functions, ranging in complexity from simple arithmetic operators like adders, accumulators, and multipliers, to system-level building blocks such as filters, transforms, and memory resources. Cores are organized by functional type into folders that expand or contract on demand.

The Xilinx CORE Generator System produces an EDIF netlist, schematic symbol, Verilog template file with a Verilog wrapper file, and a VHDL template file with a VHDL wrapper file. The Electronic Data Netlist (EDN) file contains the information for implementing the module. Cores generated in the Xilinx CORE Generator tool can be used in schematic designs. After the core is selected and customized, the CORE Generator tool generates its schematic symbol. The core then can be added to the schematic like any other library component. Finally, the template files contain code that can be used as a model to instantiate a CORE Generator module in a Verilog or VHDL design so that it can be simulated and integrated into a design.

### System Generator for DSP

The System Generator for DSP software enables electronic designs to be created, tested, and translated into hardware for Spartan-3 FPGAs. The tool extends Simulink (from The MathWorks, Inc.) to support bit- and cycle-accurate system-level simulation, and automatic code generation for Xilinx FPGAs. System Generator co-simulation interfaces extend Simulink to incorporate FPGA hardware and HDL simulation into the system-level environment as naturally as other library blocks. System Generator presents a high-level and abstract view of the design, but also exposes key features in the underlying silicon, making it possible to build extremely high-performance FPGA implementations.

### DCM Wizard

To reduce the complexities of new device technologies like Digital Clock Managers (DCM), ISE includes Architecture Wizards, allowing users access through an intuitive easy-to-use dialog. Through the use of the ISE Architecture Wizards, designers can access these leading edge technologies quickly by creating the component through a push-button flow rather than learning all the attributes in HDL. Then the component simply can be instantiated in the user’s design by copying the instantiation template created by ISE. The DCM Wizard supports all the capabilities of the Spartan-3 DCMs.

### Data2BRAM Tool

Data2BRAM is fundamentally a data translation tool. It translates contiguous fragments of data into the proper initialization records for Block RAMs. It automates distribution of that data across multiple physical Block RAMs that constitute a contiguous logical data space. Data2BRAM is also a simplified means for initializing block RAMs.

### Automatic Implementation Tools

The automatic implementation tools (synthesis, translation, mapping, placement, and routing) provide the best results for any design. ProActive Timing Closure technologies deliver the industry's highest performance in programmable logic designs, quickly and efficiently. The technologies include:

- Physical Synthesis
  - Includes place and route information to work on the real critical paths first
  - Achieves better quality of results of 5 to 20%
  - Supported through Synplicity's Amplify, Mentor Graphic's LeonardoSpectrum Time Closer, and Xilinx's own XST synthesis tool
  - Timing optimization prior to physical place and route
- Macro Builder
  - Lets you freeze placement information for a given design
  - You can then re-use that macro in future designs using relative placement
  - Performance preservation
- Advanced Place and Route Algorithms
  - Critical Path Placement first
  - Extra-Effort Mode
  - Directed Routing that lets the designer specify routing with IP
- Timing Improvement Wizard
  - Interactively helps designer improve design
  - Click on a timing problem and receive suggestions that can improve design timing
- Timing Cross-Probing
  - Decreases debug time by cross-probing from the timing report directly to Floorplanner
  - Click on the error, path, or net in the timing report and instantly see it in Floorplanner or Synthesis Source Tool
- HDL Advisors
  - Included in XST synthesis reports, clicking on an error or warning suggests changes to HDL to improve the implementation

### Incremental Design

Incremental Design gets your overall design to market faster by minimizing the impact from late-arriving design changes. The Incremental Design flow facilitates more debug cycles in a day when making small design changes. A designer quickly and easily can floorplan design areas along hierarchy boundaries, and then finish the design as normal. Later, if a design change is required, Incremental Design ensures that only the area of the design change need be re-implemented; the rest of the design stays locked and intact, delivering overall design completion faster.

### Modular Design

Modular Design lets you implement a “divide and conquer” approach to multi-million gate FPGA designs. Partitioning a design into smaller functional modules reduces the complexities of design, implementation, and verification. These design modules then can be brought through the design flow independently, leveraging all of the powerful tools within the Xilinx FPGA design flow. Once completed, a module's implementation is preserved, guaranteeing the timing in the finished device. This technology is a requirement for any organization employing a team design methodology for the design of a multi-million gate FPGA.

### Constraints Editor

Constraints are user instructions placed on elements of a schematic or HDL design, either in the design itself or in a separate file. They can indicate a number of things such as placement, implementation, naming, signal direction, and timing considerations. In the Xilinx development system, logical constraints are placed in a file called the UCF (User Constraints File). The Constraints Editor is a graphical program that you can use to create and modify those constraints.

### PACE

Pinout and Area Constraints Editor (PACE) is an interactive graphical application that you can use to do the following functions:

- View and edit location constraints for I/Os and global logic
- View and create area constraints for hierarchical symbols in your design
- Determine connectivity and resource requirements of your design
- Determine resource layout of your target FPGA
- Determine how your design maps onto the FPGA via location and area constraints

PACE fits into the Xilinx implementation flow at the very beginning. Because PACE supports I/O layout with an NGD file, it can be used early at the design entry stage of the flow. PACE reads an NGD file and reads and writes a UCF file.

### Floorplanner

Use the Floorplanner interactive graphical tool to perform the following functions on your designs:

- Floorplan resource placement at a detailed level
- Use Macro Builder to create a Relationally Placed Macro (RPM) core that can be used in other designs
- View and edit location constraints
- Find logic or nets by name or connectivity
- Cross-probe from the Timing Analyzer to the Floorplanner
- Automatic placement of ports for modular design

The graphical user interface includes pull-down menus and toolbar buttons that contain all of the necessary commands for changing the design hierarchy, floorplanning, and performing design rule checks. Dialog boxes allow you to quickly set parameters and options for command execution.

### FPGA Editor

The FPGA Editor is a graphical application for displaying and configuring FPGAs. The FPGA Editor requires an NCD file. This file contains the logic of your design mapped to components such as CLBs and IOBs. In addition, the FPGA Editor reads from and writes to a Physical Constraints File (PCF).

The following is a list of a few of the functions you can perform on your designs in the FPGA Editor:

- Place and route critical components before running automatic place and route
- Fine-tune placement and routing after running automatic place and route
- Add probes to design to examine the signal states of the targeted device
- Run the Bitstream Generator and download the resulting file to the targeted device
- View and change the nets connected to the capture units of an Integrated Logic Analyzer (ILA) core

- Create an entire design by hand (for advanced users)

### HDL Bencher Software

The HDL Bencher software automates verification of VHDL sources, Verilog sources, and schematics created within ISE. Design sources are imported, a waveform is created, and stimulus is specified by filling in the WaveTable spreadsheet cells. Outputs may be auto-simulated via a command from ISE. A self-checking test bench is exported whenever the waveform is saved. No knowledge of HDL or language scripting is needed to verify the design functions as intended.

Multiple layers of simulation are supported. Waveforms that include the expected timing results are developed for behavioral designs. The waveforms may be simulated behaviorally, after translation, after mapping, or after routing.

The HDL Bencher software constrains the test run to a specific sequence of events, initial conditions, and user-determined results. With the HDL Bencher software, you quickly can validate your design functions as intended.

### Interactive Timing Analyzer

The Interactive Timing Analyzer provides a powerful, flexible, and easy way to perform static timing analysis. With Timing Analyzer, analysis can be performed immediately after mapping, placing, or routing a Spartan-3 FPGA design.

Timing Analyzer verifies that the delay along a given path or paths meets specified timing requirements. It organizes and displays data that allows you to analyze critical paths in a circuit, the cycle time of the circuit, the delay along any specified path(s), and the path with the greatest delay. It also provides a quick analysis of the effect different speed grades have on the same design.

Timing Analyzer creates timing analysis reports based on existing timing constraints or user specified paths within the program. Timing reports have a hierarchical browser to quickly jump to different sections of the reports. Timing paths in reports can be cross-probed to synthesis tools (Exemplar and Synplicity) and the Floorplanner.

### iMPACT Configuration Tool

The iMPACT configuration tool, a command line and GUI based tool, allows you to configure your PLD designs using Boundary Scan, Slave Serial, SelectMap, and Desktop Configuration modes. It also allows you to do the following:

- Download
- Read back and verify design configuration data
- Debug configuration problems
- Create PROM, SVF, STAPL, System ACE™ CF, and System ACE MPM programming files

### ChipScope Pro Analyzer

The ChipScope Pro analyzer delivers in-circuit real-time debugging with shorter verification cycles and lower project costs. By inserting special low-impact IP debugging cores directly into your HDL code or design netlist, you can debug and verify FPGA logic and system bus activity, capturing signals at or near system operating speeds. You easily can change your trace points without having to recompile your design. The ChipScope Pro analyzer embeds Integrated Logic Analyzer (ILA) and Integrated Bus Analyzer (IBA) cores into your design. These cores allow the user to view all the internal signals and nodes within the Spartan-3 FPGA.

### XPower Analysis Tool

XPower is a post-route analysis tool for interactively and automatically analyzing power consumption for Xilinx devices. XPower includes both GUI (XPower) and batch (xpwr) applications.

Earlier in the design flow than ever, you can analyze total device power, power per net, routed, partially routed or unrouted designs, all driven from a comprehensive graphical interface or command-line driven batch mode. XPower also reads VCD simulation data from the ModelSim family of HDL simulators to set estimation stimulus, reducing setup time, as well as from additional simulators.

XPower uses device knowledge and design data to estimate device power and by-net power utilization. Information is presented in both HTML and ASCII (text) report formats. The accuracy of XPower is higher than the power estimator worksheets available for pre-design analysis.

## Conclusion

The ISE design environment brings you the fastest, most complete family of design tools available. The ISE tools are available in multiple configurations with various optional tools and interfaces to third-party tools, allowing you to customize the set of tools for your own needs. ISE combines advanced technologies such as ProActive Timing Closure with a flexible, easy-to-use graphical interface to help you achieve the best possible designs with the least time and effort, regardless of your experience level.

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 07/11/03 | 1.0 | Initial Xilinx release. |

# Using Spartan-3 IP Cores

XAPP474 (v1.0) July 11, 2003


## Summary

This document provides an overview of the Xilinx CORE Generator™ System and the Xilinx Intellectual Property (IP) offerings that facilitate the Spartan™-3 design process. For more detailed and complete information, consult the CORE Generator Guide available at http://www.xilinx.com/support/software_manuals.htm, and the Xilinx IP Center available at http://www.xilinx.com/ipcenter/index.htm.


## The CORE Generator System

The Xilinx CORE Generator System is the cataloging, customization, and delivery vehicle for IP cores targeted to Xilinx FPGAs. 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 makes it very easy to access the latest Spartan-3 IP releases and to get helpful, up-to-date information. Links to partner IP providers also are built in for the various partner-supplied AllianceCORE products. The use of CORE Generator IP cores in Spartan-3 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 Spartan-3 architecture.

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 Spartan-3 IP for the CORE Generator can be downloaded from the IP Center and added to the CORE Generator catalog.

## Xilinx IP Solutions and the IP Center

The CORE Generator works in conjunction with the Xilinx IP Center (www.xilinx.com/ipcenter). To make the most of this resource, Xilinx highly recommends that whenever starting a design, one first does a quick search of the IP Center 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

## LogiCORE Products

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. The CORE Generator commonly is used to quickly generate Spartan-3 block and distributed memories. A more detailed listing of available Spartan-3 LogiCORE products is available in Table 1, page 307 and on the Xilinx IP Center website (http://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 Interfaces

## AllianceCORE Products

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. A large percentage of Xilinx AllianceCORE partners focus on data and telecommunication applications, as well as processor and processor peripheral designs.

AllianceCORE products include customizable cores that 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.

## Reference Designs

Xilinx offers two types of design files: XAPP application notes 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. Application notes developed by Xilinx usually include supporting design files. They are supplied free of charge, without technical support or warranty.

Reference designs often can be used as starting points for implementing a broad spectrum of functions in Xilinx programmable logic. 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 partnerships with other semiconductor manufacturers and design houses so as to assist in the development of high-quality, multi-component 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.

## XPERTS Partner Consultants

Xilinx established the XPERTS Program to provide customers with access to a worldwide network of certified design consultants who are proficient with Xilinx FPGAs, software, and IP core integration. All XPERTS 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 the Xilinx strategy to provide customers with cost-efficient design solutions, while accelerating time to market.

## 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.

# Spartan-3 IP Cores

Table 1 provides a partial listing of cores available for Spartan-3 designs. For a complete catalog of Spartan-3 IP solutions, visit the Xilinx IP Center website at http://www.xilinx.com/ipcenter and search for the latest Spartan-3 core solutions.

*Table 1:* **Spartan-3 IP Cores Support**

| Function | Vendor Name | IP Type | Key Features | Application Examples |
|---|---|---|---|---|
| **Basic Elements** | | | | |
| Binary Counter | Xilinx | LogiCORE | 2-256 bit output width | |
| Binary Decoder | Xilinx | LogiCORE | 2-256 bit 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 | |
| BUFE-based Multiplexer Slice | Xilinx | LogiCORE | 1-256 bits wide | |
| BUFT-based Multiplexer Slice | Xilinx | LogiCORE | 1-256 bits wide | |
| Bus Gate | Xilinx | LogiCORE | 1-256 bits wide | |
| Bus Multiplexer | Xilinx | LogiCORE | I/O 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** | | | | |
| 8b/10b Decoder | Xilinx | LogiCORE | Industry standard 8b/10b encode/decode for serial data transmission | Physical layer of Fibre Channel |

*Table 1:* **Spartan-3 IP Cores Support**

| Function | Vendor Name | IP Type | Key Features | Application Examples |
|---|---|---|---|---|
| 8b/10b Encoder | Xilinx | LogiCORE | Industry standard 8b/10b encode/decode for serial data transmission | Physical layer of Fibre Channel |
| AES Standard Encryptor/Decryptor | Helion Technology Limited | AllianceCORE | Implements AES (Rijndael) to latest NIST FIPS PUB 197; Full dynamic support for all AES key sizes (128, 192 and 256 bits); Medium speed/low gate count version; Separate building blocks available for encryption and decryption | Security in wireless applications; 802.11 WLAN, 802.15 PAN, 802.16 MAN. Satellite communications, Networked environments; Virtual Private Networks (VPN), Storage Area Networks (SAN), Voice over IP (VoIP), Securing program content, Securing financial data |
| AES Tiny Encryptor/Decryptor | Helion Technology Limited | AllianceCORE | Implements AES (Rijndael) to latest NIST FIPS PUB 197; Full dynamic support for all AES key sizes (128, 192 and 256 bits); Low speed/ultra-low gate count version; The smallest full hardware AES solution available anywhere, fully integrated encryptor and decryptor | Security in wireless applications; 802.11 WLAN, 802.15 PAN, 802.16 MAN. Satellite communications, Networked environments; Virtual Private Networks (VPN), Storage Area Networks (SAN), Voice over IP (VoIP), Securing program content, Securing financial data |
| Convolutional Encoder | Xilinx | LogiCORE | k from 3 to 9, puncturing from 2/3 to 12/13 | 3G base stations, broadcast, wireless LAN, cable modem, xDSL, satellite, microwave |
| DES3 Encryption | CAST, Inc. | AllianceCORE | | |
| Ethernet MAC, 10/100 | Zuken, Inc. | AllianceCORE | IEEE802.3 1998 Edition Compliant; 10BASE, 100BASE MAC function; Half/Full-Duplex Operation; MII interface | ISDN network controller, NIC, switch fabric interface |
| HDLC, Single Channel | Memec Core | AllianceCORE | 16/32-bit frame seq, 8/16-bit address, insert/delete, flag/zerop, insert/detection | X.25, Frame Relay, B/D-Channel |
| Interleaver/De-interleaver | Xilinx | LogiCORE | Block & convolutional, width up to 256 bits, 256 branches | Broadcast, wireless LAN, cable modem, xDSL, satellite, microwave nets, digital TV, CDMA2000 |
| MD5 Message Digest Algorithm | CAST, Inc. | AllianceCORE | RFC 1321 compliant, suitable for data authentication applications, fully synchronous design | Electronic funds transfer, authenticated electronic data transfers, encrypted data storage |
| Reed Solomon Decoder | Xilinx | LogiCORE | Standard or custom coding, 3-12 bit symbol width, up to 4095 symbols, error & erasure decoding | Broadcast, wireless LAN, cable modem, xDSL, satellite, microwave nets, digital TV |
| Reed Solomon Encoder | Xilinx | LogiCORE | Standard or custom coding, 3-12 bit width, up to 4095 symbols with 256 check symbols | Broadcast, wireless LAN, cable modem, xDSL, satellite, microwave nets, digital TV |
| SPI-3 (POS-PHY L3) Link Layer Interface, 1-256 Channels | Xilinx | LogiCORE | OIF SPI-3 (POS-PHY L3) compliant. Fully HW interoperable with PMC-Sierra OC-48 framers. | Line cards, iSCSI cards, gigabit routers, and switches |
| SPI-4.2 Lite (POS-PHY L4) | Xilinx | LogiCORE | Functionally compliant with SPI-4.2 spec, but able to run at 1/4 data rate (2.5G vs. 10G). Optimized for low cost - it requires less logic to implement so it can run on a smaller, slower speed grade device. | SPA daughter cards sitting on top of optical line cards |

*Table 1:* **Spartan-3 IP Cores Support**

| Function | Vendor Name | IP Type | Key Features | Application Examples |
|---|---|---|---|---|
| Viterbi Decoder, General Purpose | Xilinx | LogiCORE | Puncturing, serial & parallel architecture, dynamic rate change, parameterized constraint length, soft/hard decision with programmable number of soft bits, dual rate decoder, erasure pins for external puncturing, compatible with standards such as DVB ETS, 3GPP2, IEEE802.16, HiperLAN, Intelsat IESS-308/309 | 3G base stations, broadcast, wireless LAN, cable modem, xDSL, satellite, microwave, CDMA2000 |
| **Digital Signal Processing** | | | | |
| CORDIC | Xilinx | LogiCORE | Polar to rectangular, rectangular to polar, sin & cos, sinh & cosh, atan & atanh, square root | Digital receivers |
| Direct Digital Synthesizer (DDS) | Xilinx | LogiCORE | 8-65K samples, 32-bit output precision, phase dithering/offset | |
| Fast Fourier Transform | Xilinx | LogiCORE | New core that supersedes the 64-256-1024-point Complex FFT. Transform sizes ranging from 16 to 16384 pts, selectable data precision: 8, 12, 16, 20, 24 bits, selectable phase factor precision: 8, 12, 16, 20, 24 bits, supports unscaled fixed point, scaled fixed point, block floating point, Block RAM or Distributed RAM for data or phase factor storage. | |
| FIR Filter, Distributed Arithmetic (DA) | Xilinx | LogiCORE | 32-bit input/coeff width, 1024 taps, 1-8 channels, polyphase, online coeff reload | |
| FIR Filter, MAC | Xilinx | LogiCORE | Single rate, Polyphase Decimator, Polyphase Interpolator | 3G base stations, wireless communications, image filtering |
| LFSR, Linear Feedback Shift Register | Xilinx | LogiCORE | 168 input widths, SRL16/register implementation | |
| **Math Functions** | | | | |
| Accumulator | Xilinx | LogiCORE | 1-256 bit wide | |
| Adder Subtracter | Xilinx | LogiCORE | 1-256 bit wide | |
| Multiply Accumulator (MAC) | Xilinx | LogiCORE | Input width up to 32 bits, 65-bit accumulator, truncation rounding | |
| Multiply Generator | Xilinx | LogiCORE | 64-bit input data width, constant, reloadable or variable inputs, parallel/sequential implementation | |
| Sine Cosine Look-Up Table | Xilinx | LogiCORE | 3-10 bits in, 4-32 bits out, distributed/block ROM | |
| Twos Complementer | Xilinx | LogiCORE | Input width up to 256 bits | |
| **Memories & Storage Elements** | | | | |
| Block Memory, Dual-Port | Xilinx | LogiCORE | 1-256 bits, 2-13K words | |
| Block Memory, Single-Port | Xilinx | LogiCORE | 1-256 bits, 2-128K words | |
| Content Addressable Memory (CAM) | Xilinx | LogiCORE | 1-512 bits, 2-10K words, SRL16 | |

*Table 1:* **Spartan-3 IP Cores Support**

| Function | Vendor Name | IP Type | Key Features | Application Examples |
|---|---|---|---|---|
| Distributed Memory | Xilinx | LogiCORE | 1-1024 bits, 16-65536 words, RAM/ROM/SRL16, optional output regs and pipelining | |
| FIFO, Asynchronous | Xilinx | LogiCORE | 1-256 bits, 15-65535 words, DRAM or BRAM, independent I/O clock domains | |
| FIFO, Synchronous | Xilinx | LogiCORE | 1-256 bits, 16-256 words, distributed/block RAM | |
| SDRAM Controller, Pipelined | Eureka Technology | AllianceCORE | | Networking, communication, video system, image processing equipment, medical, avionics and PC peripheral equipment |
| **Microprocessors, Controllers, & Peripherals** | | | | |
| 16450 UART with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Evaluation Core (Available with EDK) or High Value Core to be bought separately | Processor applications |
| 16550 UART with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Evaluation Core (Available with EDK) or High Value Core to be bought separately | Processor applications |
| 16-bit proprietary RISC Processor | Loarant Corporation | AllianceCORE | 44 opcodes, 64K word data, program, Harvard architecture | Control functions, State machines, Coprocessor |
| Arbiter and Bus Structure with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Infrastructure Core (includes device drivers). Available with EDK | Processor applications |
| BRAM Controller with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Memory Controller Core. Available with EDK | Processor applications |
| External Memory Controller (EMC) with OPB interface (Includes support for Flash, SRAM, ZBT, System ACE) | Xilinx | LogiCORE | CoreConnect Bus (OPB), Memory Controller Core. Available with EDK | Processor applications |
| Generic compact UART | Memec Core | AllianceCORE | UART and baud rate generator | Serial data communication |
| GPIO with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB). Available with EDK | Processor applications |
| $I^2C$ with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Evaluation Core (Available with EDK) or High Value Core to be bought separately | Networking, communications, processor applications |
| Interrupt Controller (IntC) with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Peripheral Core (includes device drivers, RTOS adaptation layers). Available with EDK | Processor applications |
| JTAG UART with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Peripheral Core (includes device drivers, RTOS adaptation layers). Available with EDK | Processor applications |
| MicroBlaze Soft RISC Processor | Xilinx | LogiCORE | 32-bit Soft Processor Core. Available with EDK | Networking, communications |
| OPB2OPB Bridge (Lite) | Xilinx | LogiCORE | CoreConnect Bus (OPB), Infrastructure Core (includes device drivers). Available with EDK | Processor applications |
| OPB2PCI Full Bridge (32/33) | Xilinx | LogiCORE | CoreConnect Bus (OPB), Infrastructure Core (includes device drivers). Available with EDK | Processor applications |

*Table 1:* **Spartan-3 IP Cores Support**

| Function | Vendor Name | IP Type | Key Features | Application Examples |
|---|---|---|---|---|
| PowerPC Bus Master | Eureka Technology | AllianceCORE | | |
| PowerPC Bus Slave | Eureka Technology | AllianceCORE | | |
| SDRAM Controller with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Memory Controller Core. Available with EDK | Processor applications |
| SPI Master and Slave with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Peripheral Core (includes device drivers, RTOS adaptation layers). Available with EDK | Networking, communications, processor applications |
| Timebase/Watch Dog Timer (WDT) with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Peripheral Core (includes device drivers, RTOS adaptation layers). Available with EDK | Processor applications |
| Timer/Counter with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Peripheral Core (includes device drivers, RTOS adaptation layers). Available with EDK | Processor applications |
| UART Lite with OPB interface | Xilinx | LogiCORE | CoreConnect Bus (OPB), Peripheral Core (includes device drivers, RTOS adaptation layers). Available with EDK | Processor applications |
| **Standard Bus Interfaces** | | | | |
| CAN 2.0 B Compatible Network Controller | Xylon d.o.o. | AllianceCORE | In compliance with CAN 2.0A and CAN 2.0B protocol specifications; Bit timing requirements, hard synchronization and resynchronization supported; Support CAN bus arbitration, automatic retransmission in error case and arbitration lost, transmission abort | Standard CAN 2.0. A/B, simple multiplex wiring systems, highly integrated automotive or building management, communication protocol bridges/gateways |
| CAN Bus Controller 2.0B | CAST, Inc. | AllianceCORE | Implementation of the Basic CAN specification; No generated Overload Frames; Receiving and transmitting of both identifiers (CAN specification 2.0B); Programmable data rate up to 1 Mbps; Programmable baud rate prescaler (up to 1/30) | Railway, Automotive, Industrial |
| CAN with 32 mail boxes | Robert Bosch GmbH | AllianceCORE | Supports CAN protocol version 2.0 part A, B; Bit rates up to 1 MBit/s; Disable Automatic Retransmission mode for Time; Triggered CAN applications; 32 Message Objects; Each Message Object has its own Identifier Mask; Programmable FIFO mode; Maskable interrupt | Automotive, Industrial Control, Telematics, Medical Engineering |
| Controller Area Network | Memec Core | AllianceCORE | | |
| PCI32 Interface Design Kit (DO-DI-PCI32-DKT) | Xilinx | LogiCORE | Includes PCI32 board, driver development kit, and customer education 3-day training class for US & Canada locations | PC boards, CPCI, Embedded, high-performance video, Gb Ethernet |
| PCI32 Interface, IP Only (DO-DI-PCI32-IP) | Xilinx | LogiCORE | v2.3 compliant, assured PCI timing, 3.3V, 0 wait state, CPCI hot swap friendly | PC add-in boards, CPCI, Embedded |
| PCI32 Single-Use License for Spartan (DO-DI-PCI32-SP) | Xilinx | LogiCORE | v2.3 compliant, assured PCI timing, 3.3V, 0 wait state, CPCI hot swap friendly | PC add-in boards, CPCI, Embedded |

*Table 1:* **Spartan-3 IP Cores Support**

| Function | Vendor Name | IP Type | Key Features | Application Examples |
|---|---|---|---|---|
| PCI64 & PCI32, IP Only (DO-DI-PCI-AL) | Xilinx | LogiCORE | v2.3 compliant, assured PCI timing, 3.3V, 0 wait state, CPCI hot swap friendly, 32 bit | PC boards, CPCI, Embedded, high performance video, Gb Ethernet |
| XAPP653: Virtex-II Pro/Spartan-3 3.3V PCI Reference Design | Xilinx | Reference Design | Application note describes how to create compliant PCI 3.3V designs | |
| **Video & Image Processing** | | | | |
| BURST_PLL | Pinpoint Solutions, Inc. | AllianceCORE | Locks to subcarrier with 1° of accuracy within 1 frame in video applications; Even higher accuracy can be guaranteed in specific configurations; BURST_PLL is fully synchronous | Video color subcarrier recovery for color regeneration Any burst locked sinusoidal wave regeneration system |
| JPEG Fast Codec | CAST, Inc. | AllianceCORE | Baseline ISO/IEC 10918-1 JPEG compliance; Fully programmable through standard JPEG stream marker segments; 4 stream defined Huffman tables; 4 stream defined Quantization tables | Printers, Digital Cameras/Camcorders, projection systems, Video conference & surveillance |
| NTSC-COSEP | Pinpoint Solutions, Inc. | AllianceCORE | 10-bit NTSC input; 8-bit BT.656 output; Proprietary 2D adaptive comb filter; Built-in colorbar test output mode; Fixed 27MHz design; AGC feedback interface for analog front end; External timing reference; Fully synchronous design | LCD Panel Controllers, Set-top Box, Digital TV / Converters, PC Desktop Video Systems, Video system design requiring NTSC input, video editing and production, Video test / verification equipment, Video storage, Video teleconferencing |

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 07/11/03 | 1.0 | Initial Xilinx release. |

XAPP477 (v1.0.1) August 11, 2003

# Embedded Processing and Control Solutions for Spartan-3 FPGAs

## Introduction

In a variety of applications, an embedded processor or controller is key to system flexibility, maintainability, and low cost. Spartan-3™ FPGAs support two powerful yet flexible Field Programmable Controller (FPC) solutions, shown in Table 1. The PicoBlaze™ FPC is a simple, highly efficient 8-bit RISC controller optimized for the Spartan-3 FPGA architecture. The MicroBlaze™ FPC is a powerful, full-featured, high-performance 32-bit RISC processor offering high-level language and real-time operating system (RTOS) support.

*Table 1:* **Embedded Processing/Control Solutions for Spartan-3 FPGAs**

| Function/Feature | PicoBlaze FPC | MicroBlaze FPC |
|---|---|---|
| Processor Architecture | 8-bit RISC controller | 32-bit RISC CPU |
| Typical Applications | Embedded control, state machines, I/O processing | Embedded computation and control |
| Memory Architecture | Harvard (separate data/code data paths) | Harvard (separate data/code data paths) |
| ALU/register width | 8 bits (byte) | 32 bits (word) |
| Registers | 16 byte-wide | 32 word-wide |
| Pipeline Stages | 0 | 3 |
| Code Address Space | 512 or 1K instructions | 512 to 4G bytes |
| Code Storage | Block RAM (internal) | Block RAM (internal)<br>External memory |
| Data Address Space | 64 bytes (internal) | 0 to 4G bytes |
| Data Storage | Distributed RAM (internal) | Block RAM (internal)<br>External memory |
| I/O Address Space | 256 locations | N/A |
| Processor Instructions | 57 | 106 |
| Operands per Instruction | 2 | 3 |
| Clocks per Instruction | 2 | 1 to 3, 34 for integer divide |
| Call/Return/Interrupt Stack | 31 locations (internal) | Variable size, in data memory |
| Interrupts | 1, Expandable | 1, Expandable |
| Maximum Interrupt Latency | 4 clock cycles (46 ns at maximum clock rate) | 7 to 40 clock cycles (application dependent) |
| Instruction Cache | N/A | 0, 2K, 4K, 8K, 16K, 32, or 64K |

*Table 1:* **Embedded Processing/Control Solutions for Spartan-3 FPGAs *(Continued)***

| Function/Feature | PicoBlaze FPC | MicroBlaze FPC |
|---|---|---|
| Data Cache | N/A | 0, 2K, 4K, 8K, 16K, 32, or 64K |
| Hardware Multiplier | N/A | 32x32 = 32 in 3 cycles |
| Hardware Divider | N/A | Optional, up to 20% performance improvement |
| Hardware Barrel Shifter | N/A | Optional, up to 15X performance improvement |
| Hardware Debugger Support | N/A | ✓ |
| LocalLink Direct Processor Interface | N/A | 200 MB/sec communication |

The PicoBlaze FPC is always fully embedded within a Spartan-3 FPGA using on-chip block RAM and distributed RAM for code and data storage. The MicroBlaze FPC optionally uses internal FPGA memory resources or interfaces to external memory to support larger code or data storage requirements. The Embedded Development Kit (EDK) for the MicroBlaze FPC includes hardware IP cores to support external Flash, SRAM, SDRAM, DDR DRAM, and ZBT SRAM memory. Similarly, the MicroBlaze FPC supports both instruction and data caches, each up to 64K bytes, to increase performance when connected to external memory.

*Table 2:* **PicoBlaze and MicroBlaze Resource Requirements and Performance**

| Function/Feature | PicoBlaze FPC | MicroBlaze FPC |
|---|---|---|
| **Resource Requirements** | | |
| Slices (4 slices = 1 CLB) | 96 | 525 |
| Block RAMs | 0.5 or 1 | 2+ |
| Effective cost in high-volume applications (250Ku, 2004) | From US$0.40 | From US$1.40 |
| Percent of XC3S50 | 13% – 25% | 68%+ |
| Percent of XC3S200 | 4% – 8% | 27%+ |
| Percent of XC3S400 | 3% – 6% | 15%+ |
| Percent of XC3S1000 | 2% – 4% | 7%+ |
| Percent of XC3S1500 | 2% – 3% | 4%+ |
| Percent of XC3S2000 | 1.3% – 3% | 3%+ |
| Percent of XC3S4000 | 0.5% – 1% | 2%+ |
| Percent of XC3S5000 | 0.5% – 1% | 1.6%+ |
| **Performance (Spartan-3 –4 speed grade)** | | |
| Maximum clock frequency | 87 MHz | 85 MHz |
| Instructions per second | 43.5M | 85M |
| Dhrystone MIPS (D-MIPS) | N/A | 68 |

Using Spartan-3 FPGAs, both MicroBlaze and PicoBlaze FPCs consume minimal FPGA resources and are highly cost effective, as shown in Table 2. Complete PicoBlaze solutions

cost as little as $0.40 in high-volume applications. MicroBlaze solutions start from $1.40 in volume.

Both the MicroBlaze and PicoBlaze FPCs provide significant numbers of flexible I/O at much lower cost than off-the-shelf controllers. Similarly, the peripheral set for both FPCs can be customized to meet the specific feature, function, and cost requirements of the target application. Because both FPCs are delivered in synthesizable HDL, both cores are future-proof, safe from any possible product obsolescence. Being integrated into the FPGA, both FPCs reduce board space, design cost, and inventory.

## PicoBlaze Application Development Support

The PicoBlaze FPC solution is a simple 8-bit RISC controller with an easy-to-use assembler. The PicoBlaze core has no direct support for in-system debugging although it can be debugged using the standard Xilinx JTAG-based interface. A simple instruction-set simulator is available.

The PicoBlaze reference design also includes UART transmitter and receiver macros with integrated 16-byte FIFOs. The UART supports 8-bit data, no parity, with one stop bit.

## MicroBlaze Application Development Support

The MicroBlaze FPC offers complete application development support, including a full suite of software development tools, an IP library of processor hardware peripheral functions, plus in-circuit hardware debugger/emulation support.

### Embedded Development Kit (EDK)

The Embedded Development Kit (EDK) is an all-encompassing solution for creating embedded programmable systems design. The EDK includes and supports the MicroBlaze soft processor core. The EDK also includes support for the PowerPC™ hard processor core, which is only available within the Xilinx Virtex-II Pro and Virtex-II Pro X FPGA families.

**Xilinx Platform Studio (XPS)**

- Tools for editing software; creating hardware and software platforms
- Runs library generation, and compiler tool chains; generates implementation and simulation netlists for use with ISE Logic Design Tools

**GNU Software Development Tools**

- C/C++ compiler for MicroBlaze and PowerPC cores (GNU gcc)
- Debugger for MicroBlaze and PowerPC cores (GNU gdb)
- Other GNU utilities

**Hardware/Software Development Tools**

- XMD - Xilinx Microprocessor Debug engine for MicroBlaze and PowerPC cores
- SystemACE tools
- Data2BRAM – Updates internal block RAM contents without recompiling the FPGA design

**Board Support Packages (BSPs)**

- Stand Alone BSP - For non-RTOS systems (MicroBlaze and PowerPC cores)

### Operating Systems

Many embedded processing applications require operating system capabilities. The following operating systems and real-time operating systems (RTOS) have ports to the MicroBlaze FPC.

- Micriμm μC/OS-II Real-Time Operating System
  http://ucos-ii.com/

- µClinux Operating System
  http://www.uclinux.org
  http://www.uclinux.org/pub/uClinux/ports/microblaze/
- ATI Nucleus Real-Time Operating System
  http://www.mentor.com/nucleus
  http://www.mentor.com/nucleus/nucleus_cpu_support.html#xilinx
- Xilinx Microkernel (XMK) Libraries
  - Highly modular scheduler, network stack, and file system
  - Minimal resource requirements and footprint size
  - Royalty-free license included with EDK purchase
  - Fully supported by Xilinx

## Processor Peripheral IP Functions

The EDK includes the following processor IP cores that support the MicroBlaze FPC. The IP cores also include device drivers and RTOS adaptation layers. Add one or more IP cores to create a custom processor to meet specific application requirements.

**Processor Peripherals**

- Timer/Counter
- Timebase/Watchdog Timer
- UART-Lite
- Interrupt Controller
- General-Purpose I/O port (GPIO)

**Serial I/O**

- SPI Master and Slave
- JTAG UART
- 16450 UART*
- 16550 UART*
- $I^2C$ two-wire serial Master and Slave*

**Memory Interfaces**

- SDRAM controller and interface
- DDR SDRAM controller and interface
- Flash memory interface
- SRAM memory interface
- Block RAM interface

**Networking Interfaces**

- Single-channel HDLC controller*
- ATM Utopia L2 master and slave controller*
- 10/100 Ethernet Media Access Controller (MAC)* (Full and Lite versions)

* IP core available as a separate product. Plugs into EDK. Evaluation versions available.

### In-Circuit Hardware Debugger Support

- EDK Software Debugger
  - Requires MicroBlaze Hardware Debug Module
  - Connects via FPGA JTAG port using Xilinx Parallel Cable IV
- Nohau In-Circuit Hardware Debugger for MicroBlaze FPC
  http://www.nohau.com/emul-microblaze-pc.html

## Related Materials and References

- MicroBlaze 32-bit RISC Processor
  http://www.xilinx.com/ipcenter/processor_central/microblaze
- PicoBlaze 8-bit RISC Controller
  http://www.xilinx.com/ipcenter/processor_central/picoblaze
- Embedded Development Kit (EDK)
  http://www.xilinx.com/ise/embedded/edk.htm
- Embedded Systems Development Training Course
  http://www.xilinx.com/support/training/abstracts/embedded-systems.htm
- MicroBlaze Recorded Lectures
  http://www.xilinx.com/support/training/mb.htm

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 07/11/03 | 1.0 | Initial Xilinx release. |

# Configuration Solutions and Considerations

## Platform Flash In-System Programmable Configuration PROMs

## Bitstream Generator (BitGen) Switches and Options


Platform Flash

# Platform Flash In-System Programmable Configuration PROMs

DS123 (v1.1) June 3, 2003

**Preliminary Product Specification**

## Features

- In-system programmable 3.3V PROMs for configuration of Xilinx FPGAs
- Low-power advanced CMOS FLASH process
- Endurance of 20,000 program/erase cycles
- Program/erase over full industrial temperature range (–40°C to +85°C)
- IEEE Standard 1149.1 boundary-scan (JTAG) support
- IEEE Standard 1532 in-system programming compatible
- Serial Slow/Fast FPGA configuration (up to 33 MHz)
- Cascadable for storing longer or multiple bitstreams
- 5V tolerant I/O pins accept 5V, 3.3V, and 2.5V signals when VCCO is at 3.3V or 2.5V
- 3.3V tolerant I/O pins accept 3.3V, 2.5V, or 1.8V signals when VCCO is at 1.8V
- 3.3V, 2.5V, or 1.8V output capability
- Available in the VO20 package
- Design support using the Xilinx Alliance and Foundation ISE series software packages.
- JTAG command initiation of standard FPGA configuration

## Description

Xilinx introduces the Platform Flash series of in-system programmable configuration PROMs (Figure 1). This 3.3V family includes a 4-megabit, a 2-megabit, and a 1-megabit PROM that provide an easy-to-use, cost-effective method for reprogramming and storing large Xilinx FPGA configuration bitstreams. The Platform Flash PROM family supports both Master Serial and Slave Serial FPGA configuration modes.

When the FPGA is in Master Serial mode, it generates a configuration clock that drives the PROM. A short access time after $\overline{CE}$ and OE are enabled, data is available on the PROM DATA (D0) pin that is connected to the FPGA $D_{IN}$ pin. New data is available a short access time after each rising clock edge. The FPGA generates the appropriate number of clock pulses to complete the configuration. When the FPGA is in Slave Serial mode, the PROM and the FPGA are clocked by an external clock.

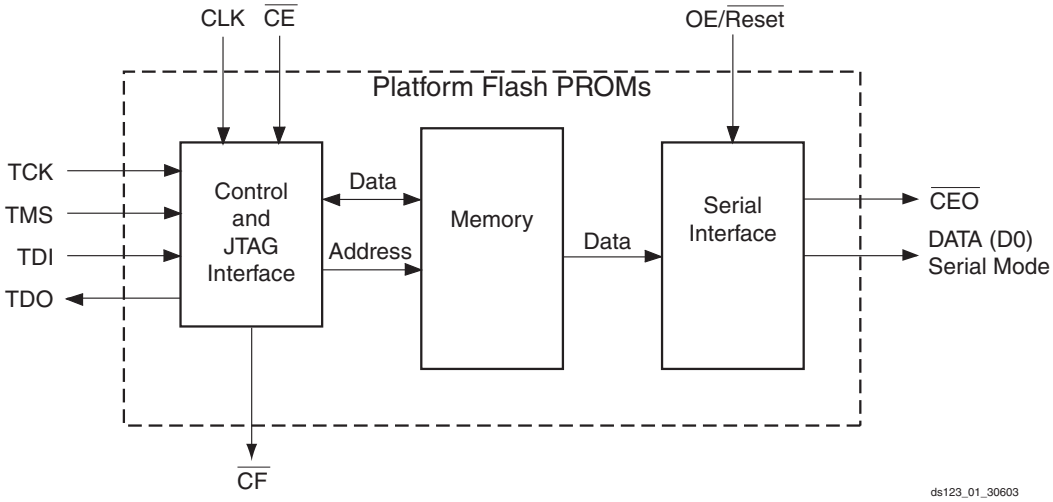



*Figure 1:* **Platform Flash PROMs Block Diagram**

Multiple devices can be concatenated by using the $\overline{CEO}$ output to drive the $\overline{CE}$ input of the following device. The clock inputs and the DATA outputs of all PROMs in this chain are interconnected. All devices are compatible and can be cascaded with other members of the family.

## Pinout and Pin Descriptions

Table 1 provides a list of the pin names and descriptions for the 20-pin VO20 package.

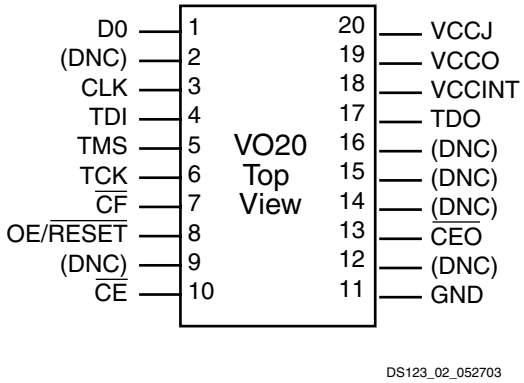
*Table 1:* **Pin Names and Descriptions**

| Pin Name | Boundary Scan Order | Function | Pin Description | 20-pin TSSOP (VO20) |
|---|---|---|---|---|
| D0 | 4 | Data Out | Serial Data Output. D0 is the DATA output pin to provide data for configuring an FPGA in serial mode. | 1 |
| | 3 | Output Enable | | |
| CLK | 0 | Data In | Configuration Clock Input. Each rising edge on the CLK input increments the internal address counter if both $\overline{CE}$ is Low and OE/$\overline{RESET}$ is High. | 3 |
| OE/$\overline{RESET}$ | 20 | Data In | Output Enable/Reset (Open-Drain I/O). When Low, this input holds the address counter reset and the DATA output is in a high-impedance state. This is a bidirectional open-drain pin that is held Low while the PROM is reset. Polarity is NOT programmable. | 8 |
| | 19 | Data Out | | |
| | 18 | Output Enable | | |
| $\overline{CE}$ | 15 | Data In | Chip Enable Input. When $\overline{CE}$ is High, the device is put into low-power standby mode, the address counter is reset, and the DATA pins are put in a high-impedance state. | 10 |
| $\overline{CF}$ | 22 | Data Out | Configuration Pulse (Open-Drain Output). Allows JTAG CONFIG instruction to initiate FPGA configuration without powering down FPGA. This is an open-drain output that is pulsed Low by the JTAG CONFIG command. | 7 |
| | 21 | Output Enable | | |
| $\overline{CEO}$ | 12 | Data Out | Chip Enable Output. Chip Enable Output ($\overline{CEO}$) is connected to the $\overline{CE}$ input of the next PROM in the chain. This output is Low when $\overline{CE}$ is Low and OE/$\overline{RESET}$ input is High, AND the internal address counter has been incremented beyond its Terminal Count (TC) value. $\overline{CEO}$ returns to High when OE/$\overline{RESET}$ goes Low or $\overline{CE}$ goes High. | 13 |
| | 11 | Output Enable | | |
| GND | | | Ground. | 11 |
| TMS | | Mode Select | JTAG Mode Select Input. The state of TMS on the rising edge of TCK determines the state transitions at the Test Access Port (TAP) controller. TMS has an internal 50K ohm resistive pull-up on it to provide a logic “1” to the device if the pin is not driven. | 5 |
| TCK | | Clock | JTAG Clock Input. This pin is the JTAG test clock. It sequences the TAP controller and all the JTAG test and programming electronics. | 6 |
| TDI | | Data In | JTAG Serial Data Input. This pin is the serial input to all JTAG instruction and data registers. TDI has an internal 50K ohm resistive pull-up on it to provide a logic “1” to the system if the pin is not driven. | 4 |
| TDO | | Data Out | JTAG Serial Data Output. This pin is the serial output for all JTAG instruction and data registers. TDO has an internal 50K ohm resistive pull-up on it to provide a logic “1” to the system if the pin is not driven. | 17 |
| VCCINT | | | +3.3V Supply. This supply voltage for internal logic. | 18 |

*Table 1:* **Pin Names and Descriptions *(Continued)***

| Pin Name | Boundary Scan Order | Function | Pin Description | 20-pin TSSOP (VO20) |
|---|---|---|---|---|
| VCCO | | | +3.3V, 2.5V, or 1.8V I/O Supply. This supply voltage connected to the output voltage drivers and input buffers. | 19 |
| VCCJ | | | +3.3V or 2.5V JTAG Signals I/O Supply | 20 |
| DNC | | | Do not connect. (These pins must be left unconnected.) | 2, 9, 12, 14, 15, 16 |

## Pinout Diagram




DS123_02_052703

## Xilinx FPGAs and Compatible PROMs

Table 2 provides a list of Xilinx FPGAs and compatible PROMs.

*Table 2:* **Xilinx FPGAs and Compatible PROMs**

| Device | Configuration Bitstream | Platform Flash PROM |
|---|---|---|
| **Virtex-II Pro™ FPGAs** | | |
| XC2VP2 | 1,305,440 | XCF02S |
| XC2VP4 | 3,006,560 | XCF04S |
| XC2VP7 | 4,485,472 | XCF04S + XCF01S(1) |
| XC2VP20 | 8,214,624 | 2 of XCF04S(1) |
| XC2VP30 | 11,589,984 | 3 of XCF04S(1) |
| XC2VP40 | 15,868,256 | 4 of XCF04S(1) |
| XC2VP50 | 19,021,408 | 5 of XCF04S(1) |
| XC2VP70 | 26,099,040 | 6 of XCF04S + XCF01S(1) |
| XC2VP100 | 34,292,832 | 8 of XCF04S + XCF01S(1) |
| XC2VP125 | 43,602,784 | 10 of XCF04S + XCF02S(1) |

*Table 2:* **Xilinx FPGAs and Compatible PROMs**

| Device | Configuration Bitstream | Platform Flash PROM |
|---|---|---|
| **Virtex™-II FPGAs** | | |
| XC2V40 | 360,096 | XCF01S |
| XC2V80 | 635,296 | XCF01S |
| XC2V250 | 1,697,184 | XCF02S |
| XC2V500 | 2,761,888 | XCF04S |
| XC2V1000 | 4,082,592 | XCF04S |
| XC2V1500 | 5,659,296 | XCF04S + XCF02S(1) |
| XC2V2000 | 7,492,000 | 2 of XCF04S(1) |
| XC2V3000 | 10,494,368 | 3 of XCF04S(1) |
| XC2V4000 | 15,659,936 | 4 of XCF04S(1) |
| XC2V6000 | 21,849,504 | 5 of XCF04S + XCF02S(1) |
| XC2V8000 | 29,063,072 | 7 of XCF04S(1) |
| **Spartan-IIE FPGAs** | | |
| XC2S50E | 630,048 | XCF01S |
| XC2S100E | 863,840 | XCF01S |
| XC2S150E | 1,134,496 | XCF02S |
| XC2S200E | 1,442,016 | XCF02S |
| XC2S300E | 1,875,648 | XCF02S |
| XC2S400E | 2,693,440 | XCF04S |
| XC2S600E | 3,961,632 | XCF04S |
| **Spartan-3 FPGAs** | | |
| XC3S50 | 439,264 | XCF01S |
| XC3S200 | 1,047,616 | XCF01S |
| XC3S400 | 1,699,136 | XCF02S |
| XC3S1000 | 3,223,488 | XCF04S |
| XC3S1500 | 5,214,784 | XCF04S + XCF01S(1) |

*Table 2:* **Xilinx FPGAs and Compatible PROMs**

| Device | Configuration Bitstream | Platform Flash PROM |
|---|---|---|
| XC3S2000 | 7,673,024 | 2 of XCF04S(1) |
| XC3S4000 | 11,316,864 | 3 of XCF04S(1) |
| XC3S5000 | 13,271,936 | 3 of XCF04S + XCF01S(1) |

**Notes:**
1. Contact your Xilinx sales representative for future availability of the XCF08P, XCF16P, and XCF32P PROMs.

## Capacity

| Devices | Configuration Bits |
|---|---|
| XCF04S | 4,194,304 |
| XCF02S | 2,097,152 |
| XCF01S | 1,048,576 |

## In-System Programming

In-System Programmable PROMs can be programmed individually, or two or more can be daisy-chained together and programmed in-system via the standard 4-pin JTAG protocol as shown in Figure 2. The Platform Flash PROMs are IEEE Standard 1532 in-system programming compatible. In-system programming offers quick and efficient design iterations and eliminates unnecessary package handling or socketing of devices. The Xilinx development system provides the programming data sequence using either Xilinx iMPACT software and a
download cable, a third-party JTAG development system, a JTAG-compatible board tester, or a simple microprocessor interface that emulates the JTAG instruction sequence. The iMPACT software also outputs serial vector format (SVF) files for use with any tools that accept SVF format and with automatic test equipment.

All outputs are held in a high-impedance state or held at clamp levels during in-system programming.

## OE/$\overline{\text{RESET}}$

The ISP programming algorithm requires issuance of a reset that causes OE to pulse Low.

## External Programming

Xilinx reprogrammable PROMs can also be programmed by the Xilinx MultiPRO Desktop Tool or a third-party device programmer. This provides the added flexibility of using pre-programmed devices with an in-system programmable option for future enhancements and design changes.

## Reliability and Endurance

Xilinx in-system programmable products provide a guaranteed endurance level of 20,000 in-system program/erase cycles and a minimum data retention of 20 years. Each device meets all functional, performance, and data retention specifications within this endurance limit.

## Design Security

The Xilinx in-system programmable PROM devices incorporate advanced data security features to fully protect the programming data against unauthorized reading via JTAG. Table 3 shows the security setting available.

The read security bit can be set by the user to prevent the internal programming pattern from being read or copied via JTAG. When set, it allows device erase. Erasing the entire device is the only way to reset the read security bit.

*Table 3:* **Data Security Options**

| Default = Reset | Set |
|---|---|
| Read Allowed<br>Program/Erase Allowed<br>Verify Allowed | Read Inhibited via JTAG<br>Program/Erase Allowed<br>Verify Inhibited |

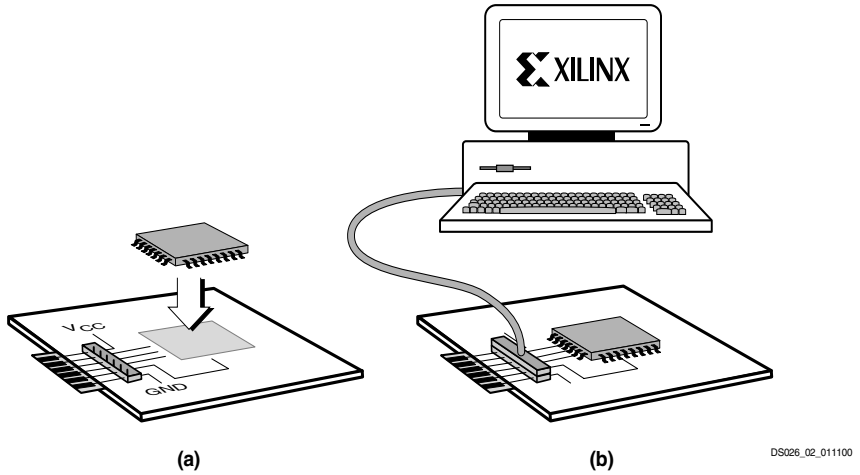



*Figure 2:* **In-System Programming Operation (a) Solder Device to PCB and (b) Program Using Download Cable**

# IEEE 1149.1 Boundary-Scan (JTAG)

The Platform Flash PROM family is fully compliant with the IEEE Std. 1149.1 Boundary-Scan, also known as JTAG. A Test Access Port (TAP) and registers are provided to support all required boundary scan instructions, as well as many of the optional instructions specified by IEEE Std. 1149.1. In addition, the JTAG interface is used to implement in-system programming (ISP) to facilitate configuration, erasure, and verification operations on the Platform Flash PROM device.

Table 4 lists the required and optional boundary-scan instructions supported in the Platform Flash PROMs. Refer to the IEEE Std. 1149.1 specification for a complete description of boundary-scan architecture and the required and optional instructions.

# Instruction Register

The Instruction Register (IR) for the Platform Flash PROM is eight bits wide and is connected between TDI and TDO during an instruction scan sequence. In preparation for an instruction scan sequence, the instruction register is parallel loaded with a fixed instruction capture pattern. This pattern is shifted out onto TDO (LSB first), while an instruction is shifted into the instruction register from TDI. The detailed composition of the instruction capture pattern is illustrated in Figure 3.

The ISP Status field, IR(4), contains logic “1” if the device is currently in ISP mode; otherwise, it contains logic “0”. The Security field, IR(3), contains logic “1” if the device has been programmed with the security option turned on; otherwise, it contains logic “0”.

*Table 4:* **Boundary Scan Instructions**

| Boundary-Scan Command | Binary Code [7:0] | Description |
|---|---|---|
| **Required Instructions** | | |
| BYPASS | 11111111 | Enables BYPASS |
| SAMPLE/ PRELOAD | 00000001 | Enables boundary-scan SAMPLE/PRELOAD operation |
| EXTEST | 00000000 | Enables boundary-scan EXTEST operation |
| **Optional Instructions** | | |
| CLAMP | 11111010 | Enables boundary-scan CLAMP operation |
| HIGHZ | 11111100 | all outputs in high-impedance state simultaneously |
| IDCODE | 11111110 | Enables shifting out 32-bit IDCODE |
| USERCODE | 11111101 | Enables shifting out 32-bit USERCODE |
| **Platform Flash PROM Specific Instructions** | | |
| CONFIG | 11101110 | Initiates FPGA configuration by pulsing $\overline{CF}$ pin Low once |

| | IR[7:5] | IR[4] | IR[3] | IR[2] | IR[1:0] | |
|---|---|---|---|---|---|---|
| TDI-> | Reserved | ISP Status | Security | 0 | 0 1 | ->TDO |

**Notes:**
1. IR(1:0) = 01 is specified by IEEE Std. 1149.1

*Figure 3:* **Instruction Register Values Loaded into IR as Part of an Instruction Scan Sequence**

## Boundary Scan Register

The boundary-scan register is used to control and observe the state of the device pins during the EXTEST, SAMPLE/PRELOAD, and CLAMP instructions. Each output pin on the Platform Flash PROM has two register stages that contribute to the boundary-scan register, while each input pin only has one register stage.

For each output pin, the register stage nearest to TDI controls and observes the output state, and the second stage closest to TDO controls and observes the High-Z enable state of the pin.

For each input pin, the register stage controls and observes the input state of the pin.

## Identification Registers

The IDCODE is a fixed, vendor-assigned value that is used to electrically identify the manufacturer and type of the device being addressed. The IDCODE register is 32 bits wide. The IDCODE register can be shifted out for examination by using the IDCODE instruction. The IDCODE is available to any other system component via JTAG.

The IDCODE register has the following binary format:

```
vvvv:ffff:ffff:aaaa:aaaa:cccc:cccc:ccc1
```

where

v = the die version number

f = the family code (50h for Platform Flash PROM family)

a = the ISP PROM product ID (46h for the XCF04S)

c = the company code (49h for Xilinx)

**Notes:**
1. The LSB of the IDCODE register is always read as logic “1” as defined by IEEE Std. 1149.1.

Table 5 lists the IDCODE register values for the Platform Flash PROMs.

*Table 5:* **IDCODES Assigned to Platform Flash PROMs**

| ISP-PROM | IDCODE |
|---|---|
| XCF01S | 05044093h |
| XCF02S | 05045093h |
| XCF04S | 05046093h |

The USERCODE instruction gives access to a 32-bit user programmable scratch pad typically used to supply information about the device's programmed contents. By using the USERCODE instruction, a user-programmable identification code can be shifted out for examination. This code is loaded into the USERCODE register during programming of the Platform Flash PROM. If the device is blank or was not loaded during programming, the USERCODE register contains FFFFFFFFh.

## Platform Flash PROM TAP Characteristics

The Platform Flash PROM family performs both in-system programming and IEEE 1149.1 boundary-scan (JTAG) testing via a single 4-wire Test Access Port (TAP). This simplifies system designs and allows standard Automatic Test Equipment to perform both functions. The AC characteristics of the Platform Flash PROM TAP are described as follows.

## TAP Timing

Figure 4 shows the timing relationships of the TAP signals. These TAP timing characteristics are identical for both boundary-scan and ISP operations.

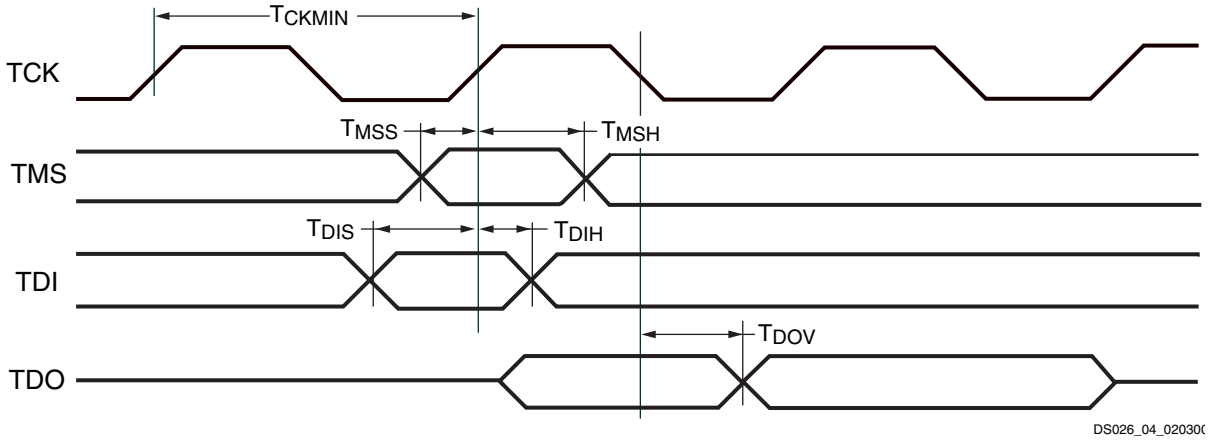



*Figure 4:* **Test Access Port Timing**

## TAP AC Parameters

Table 6 shows the timing parameters for the TAP waveforms shown in Figure 4.

*Table 6:* **Test Access Port Timing Parameters**

| Symbol | Parameter | Min | Max | Units |
|---|---|---|---|---|
| $T_{CKMIN1}$ | TCK minimum clock period when $V_{CCJ}$ = 2.5V or 3.3V | 100 | - | ns |
| $T_{CKMIN2}$ | TCK minimum clock period, Bypass Mode, when $V_{CCJ}$ = 2.5V or 3.3V | 50 | - | ns |
| $T_{MSS}$ | TMS setup time when $V_{CCJ}$ = 2.5V or 3.3V | 10 | - | ns |
| $T_{MSH}$ | TMS hold time when $V_{CCJ}$ = 2.5V or 3.3V | 25 | - | ns |
| $T_{DIS}$ | TDI setup time when $V_{CCJ}$ = 2.5V or 3.3V | 10 | - | ns |
| $T_{DIH}$ | TDI hold time when $V_{CCJ}$ = 2.5V or 3.3V | 25 | - | ns |
| $T_{DOV}$ | TDO valid delay when $V_{CCJ}$ = 2.5V or 3.3V | - | 30 | ns |

## Connecting Configuration PROMs

Connecting the FPGA device with the configuration PROM (see Figure 5 and Figure 6).

- The DATA output(s) of the PROM(s) drives the $D_{IN}$ input of the lead FPGA device.
- The Master FPGA CCLK output drives the CLK input(s) of the PROM(s) (in Master-Serial mode only).
- The $\overline{CEO}$ output of a PROM drives the $\overline{CE}$ input of the next PROM in a daisy chain (if any).
- The OE/$\overline{RESET}$ pins of all PROMs are connected to the INIT_B pins of all FPGA devices. This connection assures that the PROM address counter is reset before the start of any (re)configuration, even when a reconfiguration is initiated by a $V_{CCINT}$ glitch.
- The PROM $\overline{CE}$ input can be driven from the DONE pin. The $\overline{CE}$ input of the first (or only) PROM can be driven by the DONE output of all target FPGA devices, provided that DONE is not permanently grounded. $\overline{CE}$ can also be permanently tied Low, but this keeps the DATA output active and causes an unnecessary supply current of 10 mA maximum.

### Initiating FPGA Configuration

The Platform Flash PROMs incorporate a pin named $\overline{CF}$ that is controllable through the JTAG CONFIG instruction. Executing the CONFIG instruction through JTAG pulses the $\overline{CF}$ low once for 300-500 ns, which resets the FPGA and initiates configuration.

The $\overline{CF}$ pin must be connected to the PROG_B pin on the FPGA(s) to use this feature.

The iMPACT software can also issue a JTAG CONFIG command to initiate FPGA configuration through the “Load FPGA” setting.

## Master Serial Mode Summary

The I/O and logic functions of the Configurable Logic Block (CLB) and their associated interconnections are established by a configuration program. The program is loaded either automatically upon power up, or on command, depending on the state of the three FPGA mode pins. In Master Serial mode, the FPGA automatically loads the configuration program from an external memory. Xilinx PROMs are designed to accommodate the Master Serial mode.

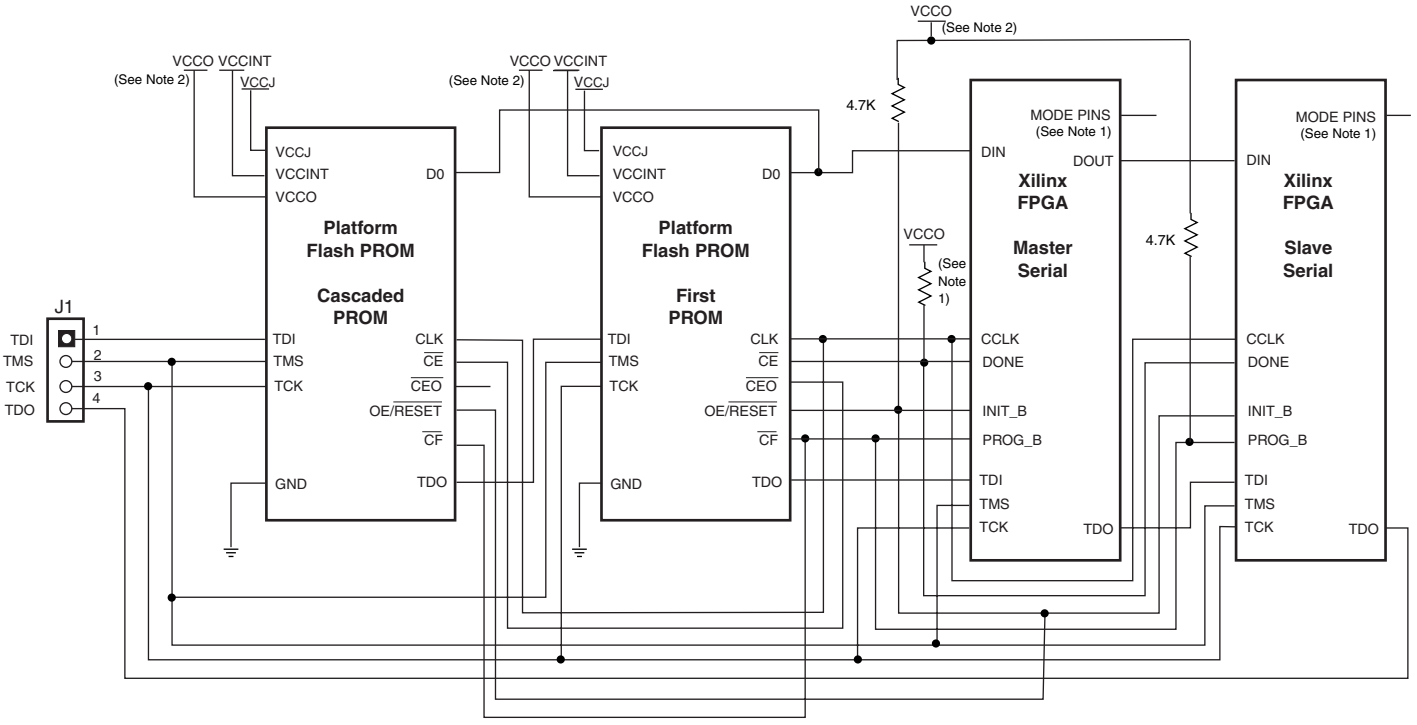
Upon power-up or reconfiguration, an FPGA enters the Master Serial mode whenever all three of the FPGA mode-select pins are Low (M0=0, M1=0, M2=0). Data is read from the PROM sequentially on a single data line. Synchronization is provided by the rising edge of the temporary signal CCLK, which is generated by the FPGA during configuration.

Master Serial Mode provides a simple configuration interface. Only a serial data line, a clock line, and two control lines are required to configure an FPGA. Data from the PROM is read sequentially, accessed via the internal address and bit counters which are incremented on every valid rising edge of CCLK.

### Cascading Configuration PROMs

For multiple FPGAs configured as a serial daisy-chain, or a single FPGA requiring larger configuration memories in a serial configuration mode, cascaded PROMs provide additional memory (Figure 5). Multiple Platform Flash PROMs can be concatenated by using the $\overline{CEO}$ output to drive the $\overline{CE}$ input of the downstream device. The clock inputs and the data outputs of all Platform Flash PROMs in the chain are interconnected. After the last data from the first PROM is read, the next clock signal to the PROM asserts its $\overline{CEO}$ output Low and drives its DATA line to a high-impedance state. The second PROM recognizes the Low level on its $\overline{CE}$ input and enables its DATA output.

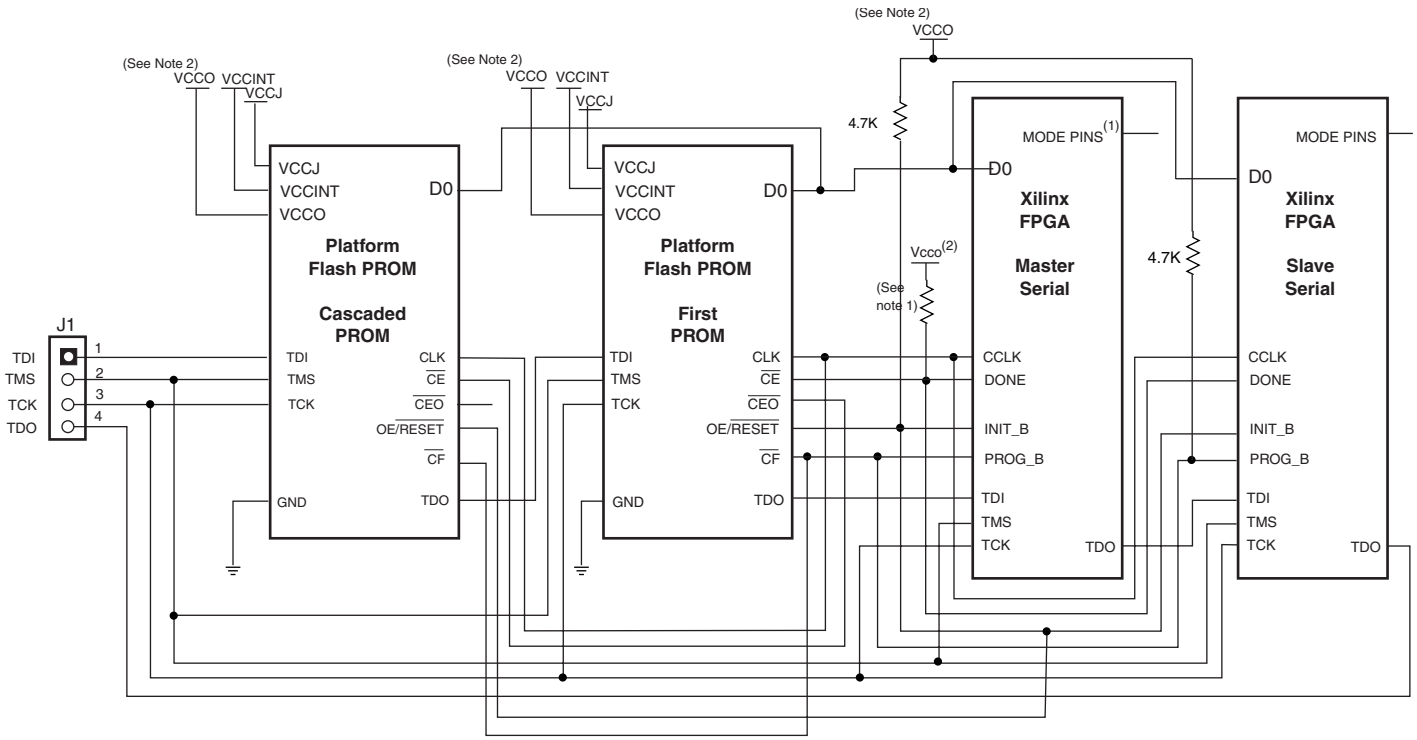
After configuration is complete, address counters of all cascaded PROMs are reset if the PROM OE/$\overline{RESET}$ pin goes Low or $\overline{CE}$ goes High.




Notes:
1 For Mode pin connections and DONE pin pullup value, refer to appropriate FPGA data sheet.
2 For compatible voltages, refer to the appropriate data sheet.

DS123_08_081103

*Figure 5:* **Configuring Multiple Devices in Master/Slave Serial Mode**




Notes:
1 For Mode pin connections and DONE pin pullup value, refer to the appropriate FPGA data sheet.
2 For compatible voltages, refer to the appropriate data sheet.

DS123_09_052903

*Figure 6:* **Configuring Multiple Devices with Identical Patterns in Master/Slave Serial Mode**

## Reset Activation

On power up, OE/$\overline{\text{RESET}}$ is held low until the Platform Flash PROM is active (1 ms). OE/$\overline{\text{RESET}}$ is connected to an external resistor to pull OE/$\overline{\text{RESET}}$ HIGH releasing the FPGA $\overline{\text{INIT}}$ and allowing configuration to begin. If the power drops below 2.0V, the PROM resets. OE/$\overline{\text{RESET}}$ polarity is *not* programmable. See Figure 7 for power-up requirements.

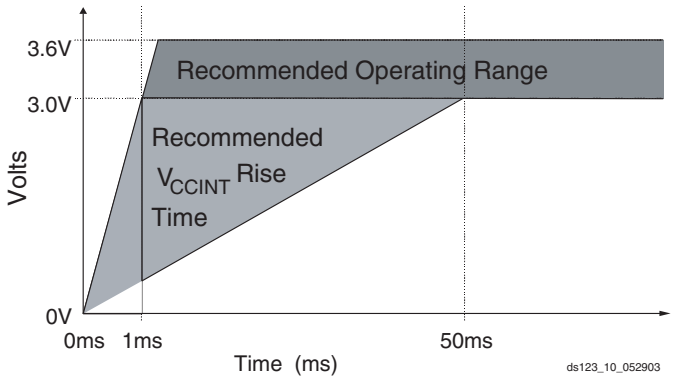



*Figure 7:* **$V_{CCINT}$ Power-Up Requirements**

## Standby Mode

The PROM enters a low-power standby mode whenever $\overline{\text{CE}}$ is asserted High. The address is reset. The output remains in a high-impedance state regardless of the state of the OE input. JTAG pins TMS, TDI and TDO can be in a high-impedance state or High. See Table 7.

## 5V Tolerant I/Os

The I/Os on each re-programmable PROM are fully 5V tolerant even when the core power supply is 3.3V if VCCO is at 3.3V or 2.5V. This allows 5V CMOS signals to connect directly to the PROM inputs without damage. In addition, the 3.3V $V_{CCINT}$ power supply can be applied before or after 5V signals are applied to the I/Os. In mixed 5V/3.3V/2.5V systems, the user pins, the core power supply ($V_{CCINT}$), and the output power supply ($V_{CCO}$) can have power applied in any order. This makes the PROM devices immune to power supply sequencing issues.

**Notes:**
1. When VCCO is set to 1.8V, the I/Os are only 3.3V tolerant.

*Table 7:* **Truth Table for PROM Control Inputs**

| Control Inputs | | Internal Address | Outputs | | |
|---|---|---|---|---|---|
| OE/RESET | CE | | DATA | CEO | $I_{CC}$ |
| High | Low | If address ≤ TC(1): increment<br>If address > TC(1): don't change | Active<br>High-Z | High<br>Low | Active<br>Reduced |
| Low | Low | Held reset | High-Z | High | Active |
| High | High | Held reset | High-Z | High | Standby |
| Low | High | Held reset | High-Z | High | Standby |

**Notes:**
1. TC = Terminal Count = highest address value. TC + 1 = address 0.

## Absolute Maximum Ratings(1,2)

| Symbol | Description | Value | Units |
|---|---|---|---|
| $V_{CCINT}$/ $V_{CCO}$/ $V_{CCJ}$ | Supply voltage relative to GND | –0.5 to +4.0 | V |
| $V_{IN}$ | Input voltage with respect to GND | –0.5 to +5.5 | V |
| $V_{TS}$ | Voltage applied to High-Z output | –0.5 to +5.5 | V |
| $T_{STG}$ | Storage temperature (ambient) | –65 to +150 | °C |
| $T_{SOL}$ | Maximum soldering temperature (10s @ 1/16 in.) | +220 | °C |
| $T_J$ | Junction temperature | +125 | °C |

**Notes:**
1. Maximum DC undershoot below GND must be limited to either 0.5V or 10 mA, whichever is easier to achieve. During transitions, the device pins can undershoot to –2.0V or overshoot to +7.0V, provided this over- or undershoot lasts less then 10 ns and with the forcing current being limited to 200 mA.
2. Stresses beyond those listed under Absolute Maximum Ratings might cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those listed under Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time might affect device reliability.

## Recommended Operating Conditions

| Symbol | Parameter | Min | Max | Units |
|---|---|---|---|---|
| $V_{CCINT}$ | Internal voltage supply | 3.0 | 3.6 | V |
| $V_{CCO}$ | Supply voltage for output drivers for $V_{CCO}$ at 3.3V operation | 3.0 | 3.6 | V |
| | Supply voltage for output drivers for $V_{CCO}$ at 2.5V operation | 2.3 | 2.7 | V |
| | Supply voltage for output drivers for $V_{CCO}$ at 1.8V operation | 1.7 | 2.0 | V |
| $V_{CCJ}$ | Supply voltage for output drivers for $V_{CCO}$ at 3.3V operation | 3.0 | 3.6 | V |
| | Supply voltage for output drivers for $V_{CCO}$ at 2.5V operation | 2.3 | 2.7 | V |
| $V_{IL}$ | Low-level input voltage for $V_{CCO}$ at 3.3V or 2.5V | 0 | 0.8 | V |
| | Low-level input voltage for $V_{CCO}$ at 1.8V | 0 | 20% $V_{CCO}$ | V |
| $V_{IH}$ | High-level input voltage for $V_{CCO}$at 3.3V or 2.5V | 2.0 | 5.5 | V |
| | High-level input voltage for $V_{CCO}$ at 1.8V | 70% $V_{CCO}$ | 3.6 | V |
| $V_O$ | Output voltage | 0 | $V_{CCO}$ | V |
| $T_{VCC}$ | $V_{CCINT}$ rise time from 0V to nominal voltage(1) | 1 | 50 | ms |
| $T_A$ | Operating ambient temperature | –40° | 85° | C |

**Notes:**
1. At power up, the device requires the $V_{CCINT}$ power supply to monotonically rise from 0V to nominal voltage within the specified $V_{CCINT}$ rise time. If the power supply cannot meet this requirement, then the device might not perform power-on-reset properly. See Figure 7.

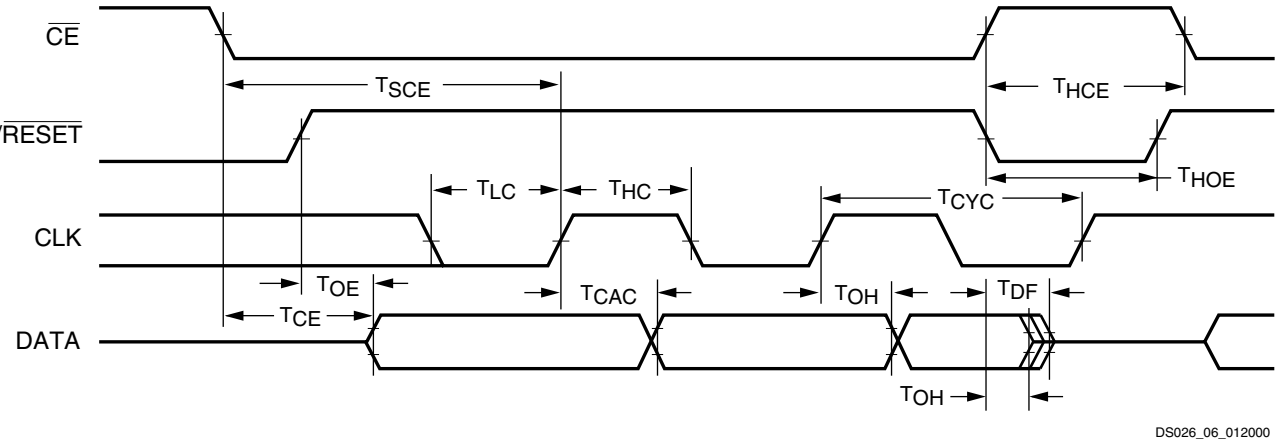
## Quality and Reliability Characteristics

| Symbol | Description | Min | Max | Units |
|---|---|---|---|---|
| $T_{DR}$ | Data retention | 20 | - | Years |
| $N_{PE}$ | Program/erase cycles (Endurance) | 20,000 | - | Cycles |
| $V_{ESD}$ | Electrostatic discharge (ESD) | 2,000 | - | Volts |

## DC Characteristics Over Operating Conditions

| Symbol | Parameter | Test Conditions | Min | Max | Units |
|---|---|---|---|---|---|
| $V_{OH}$ | High-level output voltage for 3.3V outputs | $I_{OH}$ = –4 mA | 2.4 | - | V |
| | High-level output voltage for 2.5V outputs | $I_{OH}$ = –500 µA | $V_{CCO}$ – 0.4 | - | V |
| | High-level output voltage for 1.8V outputs | $I_{OH}$ = –50 µA | $V_{CCO}$ – 0.4 | - | V |
| $V_{OL}$ | Low-level output voltage for 3.3V outputs | $I_{OL}$ = 8 mA | - | 0.4 | V |
| | Low-level output voltage for 2.5V outputs | $I_{OL}$ = 500 µA | - | 0.4 | V |
| | Low-level output voltage for 1.8V outputs | $I_{OL}$ = 50 µA | - | 0.4 | V |
| $I_{CC}$ | Supply current, active mode | 25 MHz | - | 10 | mA |
| $I_{CCS}$ | Supply current, standby mode | - | - | 1 | mA |
| $I_{ILJ}$ | JTAG pins TMS, TDI, and TDO pull-up current | $V_{CCJ}$ = MAX<br>$V_{IN}$ = GND | - | 100 | µA |
| $I_{IL}$ | Input leakage current | $V_{CCINT}$ = Max<br>$V_{IN}$ = GND or $V_{CCINT}$ | –10 | 10 | µA |
| $I_{IH}$ | Input and output High-Z leakage current | $V_{CCINT}$ = Max<br>$V_{IN}$ = GND or $V_{CCINT}$ | –10 | 10 | µA |
| $C_{IN}$ | Input capacitance | $V_{IN}$ = GND<br>f = 1.0 MHz | - | 8 | pF |
| $C_{OUT}$ | Output capacitance | $V_{IN}$ = GND<br>f = 1.0 MHz | - | 14 | pF |

# AC Characteristics Over Operating Conditions for XCF04S, XCF02S, and XCF01S

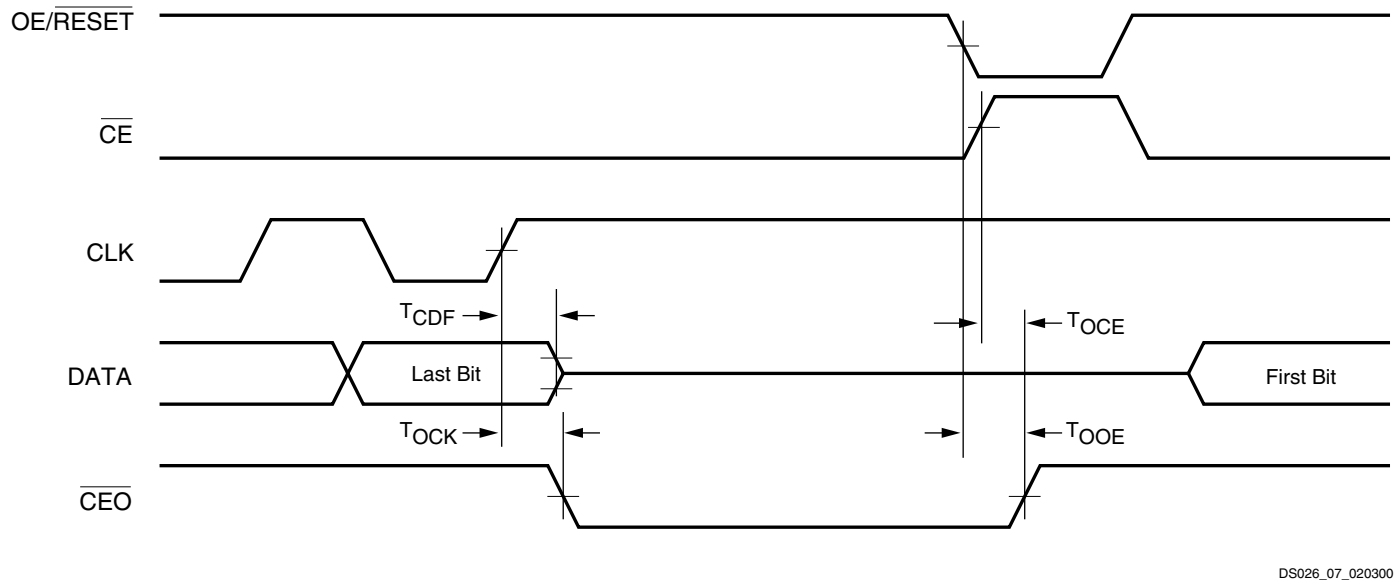



| Symbol | Description | Min | Max | Units |
|---|---|---|---|---|
| $T_{OE}$ | OE/$\overline{RESET}$ to data delay when $V_{CCO}$ = 3.3V or 2.5V | - | 10 | ns |
| | OE/$\overline{RESET}$ to data delay when $V_{CCO}$ = 1.8V | - | 30 | ns |
| $T_{CE}$ | $\overline{CE}$ to data delay when $V_{CCO}$ = 3.3V or 2.5V | - | 15 | ns |
| | $\overline{CE}$ to data delay when $V_{CCO}$ = 1.8V | - | 30 | ns |
| $T_{CAC}$ | CLK to data delay when $V_{CCO}$ = 3.3V or 2.5V | - | 15 | ns |
| | CLK to data delay when $V_{CCO}$ = 1.8V | - | 30 | ns |
| $T_{OH}$ | Data hold from $\overline{CE}$, OE/$\overline{RESET}$, or CLK when $V_{CCO}$ = 3.3V or 2.5V | 0 | - | ns |
| | Data hold from $\overline{CE}$, OE/$\overline{RESET}$, or CLK when $V_{CCO}$ = 1.8V | 0 | - | ns |
| $T_{DF}$ | $\overline{CE}$ or OE/$\overline{RESET}$ to data float delay(2) when $V_{CCO}$ = 3.3V or 2.5V | - | 25 | ns |
| | $\overline{CE}$ or OE/$\overline{RESET}$ to data float delay(2) when $V_{CCO}$ = 1.8V | - | 30 | ns |
| $T_{CYC}$ | Clock period(7) when $V_{CCO}$ = 3.3V or 2.5V | 30 | - | ns |
| | Clock period when $V_{CCO}$ = 1.8V | 67 | - | ns |
| $T_{LC}$ | CLK Low time(3) when $V_{CCO}$ = 3.3V or 2.5V | 10 | - | ns |
| | CLK Low time(3) when $V_{CCO}$ = 1.8V | 15 | - | ns |
| $T_{HC}$ | CLK High time(3) when $V_{CCO}$ = 3.3V or 2.5V | 10 | - | ns |
| | CLK High time(3) when $V_{CCO}$ = 1.8V | 15 | - | ns |
| $T_{SCE}$ | $\overline{CE}$ setup time to CLK (guarantees proper counting)(3) when $V_{CCO}$ = 3.3V or 2.5V | 20 | - | ns |
| | $\overline{CE}$ setup time to CLK (guarantees proper counting)(3) when $V_{CCO}$ = 1.8V | 30 | - | ns |
| $T_{HCE}$ (5) | $\overline{CE}$ High time (guarantees counters are reset) when $V_{CCO}$ = 3.3V or 2.5V | 250 | - | ns |
| | $\overline{CE}$ High time (guarantees counters are reset) when $V_{CCO}$ = 1.8V | 250 | - | ns |

| Symbol | Description | Min | Max | Units |
|---|---|---|---|---|
| $T_{HOE}$ (6) | OE/$\overline{RESET}$ hold time (guarantees counters are reset) when $V_{CCO}$ = 3.3V or 2.5V | 250 | - | ns |
| | OE/$\overline{RESET}$ hold time (guarantees counters are reset) when $V_{CCO}$ = 1.8V | 250 | - | ns |

**Notes:**
1. AC test load = 50 pF.
2. Float delays are measured with 5 pF AC loads. Transition is measured at ±200 mV from steady state active levels.
3. Guaranteed by design, not tested.
4. All AC parameters are measured with $V_{IL}$ = 0.0V and $V_{IH}$ = 3.0V.
5. If $T_{HCE}$ High < 2 µs, $T_{CE}$ = 2 µs.
6. If $T_{HOE}$ Low < 2 µs, $T_{OE}$ = 2 µs.
7. This is the minmum possible $T_{CYC}$. Actual $T_{CYC}$ = $T_{CAC}$ + FPGA Data setup time. If FPGA Data setup time = 15 ns, the actual $T_{CYC}$ = 15 ns +15 ns = 30 ns.

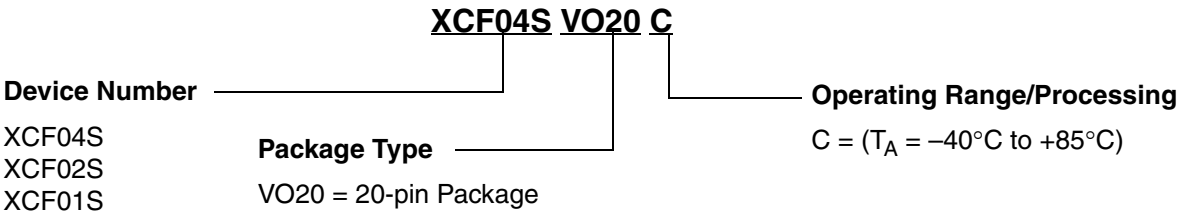
# AC Characteristics Over Operating Conditions When Cascading for XCF04S, XCF02S, and XCF01S




| Symbol | Description | Min | Max | Units |
|---|---|---|---|---|
| $T_{CDF}$ | CLK to data float delay(2,3) when $V_{CCO}$ = 2.5V or 3.3V | - | 25 | ns |
| | CLK to data float delay(2,3) when $V_{CCO}$ = 1.8V | - | 35 | ns |
| $T_{OCK}$ | CLK to $\overline{CEO}$ delay(3,5) when $V_{CCO}$ = 2.5V or 3.3V | - | 20 | ns |
| | CLK to $\overline{CEO}$ delay(3,5) when $V_{CCO}$ = 1.8V | - | 35 | ns |
| $T_{OCE}$ | CE to $\overline{CEO}$ delay(3) when $V_{CCO}$ = 2.5V or 3.3V | - | 20 | ns |
| | CE to $\overline{CEO}$ delay(3) when $V_{CCO}$ = 1.8V | - | 35 | ns |
| $T_{OOE}$ | OE/$\overline{RESET}$ to $\overline{CEO}$ delay(3) when $V_{CCO}$ = 2.5V or 3.3V | - | 20 | ns |
| | OE/$\overline{RESET}$ to $\overline{CEO}$ delay(3) when $V_{CCO}$ = 1.8V | - | 35 | ns |

**Notes:**
1. AC test load = 50 pF.
2. Float delays are measured with 5 pF AC loads. Transition is measured at ±200 mV from steady state active levels.
3. Guaranteed by design, not tested.
4. All AC parameters are measured with $V_{IL}$ = 0.0V and $V_{IH}$ = 3.0V.
5. For cascaded PROMs minimum, $T_{CYC} = T_{OCK}$ + FPGA Data setup time.

## Ordering Information

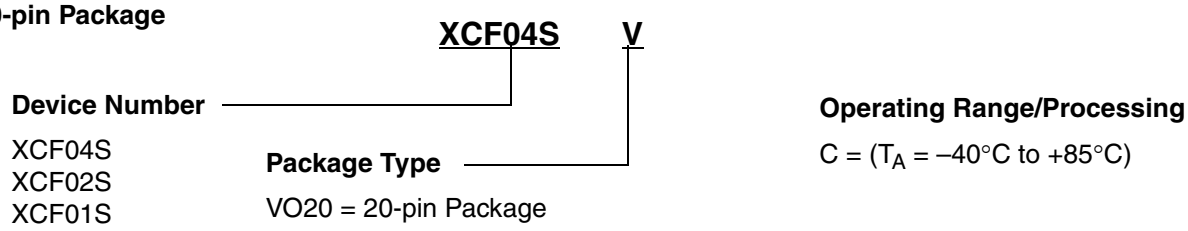



## Valid Ordering Combinations

| |
|---|
| XCF04SVO20 C |
| XCF02SVO20 C |
| XCF01SVO20 C |

## Marking Information

**20-pin Package**




## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 04/29/03 | 1.0 | Xilinx Initial Release. |
| 06/03/03 | 1.1 | Made edits to all pages. |

# Bitstream Generator (BitGen) Switches and Options

The BitGen program generates bitstream configuration files for a Spartan-3 FPGA. It can be run as BitGen in the command-line mode or run within the Project Navigator as part of the "Generate Programming File" process, where the same options are available through dialog boxes. The information below is a subset of the BitGen section of the Xilinx software documentation, which can be found on-line at http://www.xilinx.com/support/software_manuals.htm. See the on-line documentation for the complete options available for a specific version of the program. The information relevant to Spartan-3 FPGAs is included here.

This chapter contains the following sections:



## BitGen Overview

BitGen produces a bitstream for Xilinx device configuration. After the design is completely routed, it is necessary to configure the device so that it can execute the desired function. This is done using files generated by BitGen, the Xilinx bitstream generation program. BitGen takes a fully routed NCD (native circuit description) file as input and produces a configuration bitstream—a binary file with a .bit extension.

The BIT file contains all of the configuration information from the NCD file that defines the internal logic and interconnections of the FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT file is then downloaded into the FPGAs memory cells, or it is used to create a PROM file.

The following figure shows the BitGen input and output files:

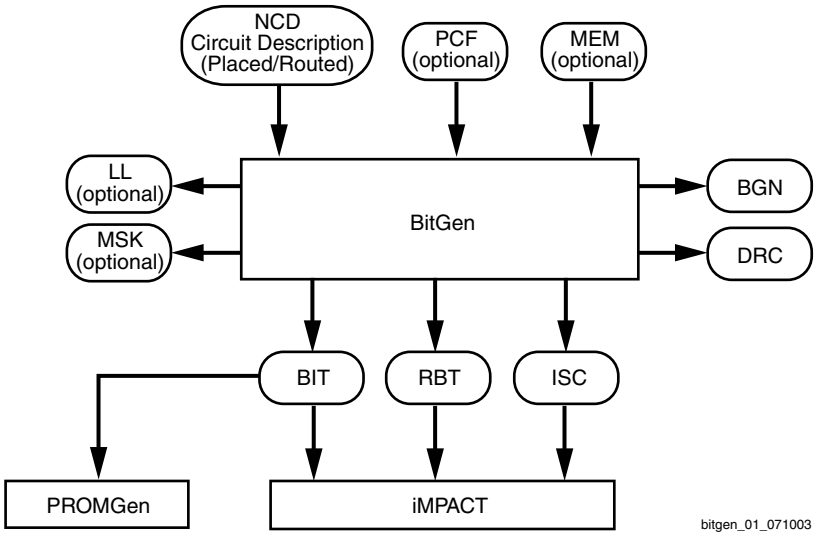



*Figure 2-1:* **BitGen input and output files**

# BitGen Syntax

The following syntax creates a bitstream from your NCD file:

```
bitgen [options] infile[.ncd] [outfile] [pcf_file.pcf]
```

*options* is one or more of the options listed in "BitGen Options".

*infile* is the name of the NCD design for which you want to create the bitstream. You may specify only one design file, and it must be the first file specified on the command line.

***Note:*** You do not have to use an extension. If you do not use an extension, then .ncd is assumed. If you do use an extension, then the extension must be .ncd.

*outfile* is the name of the output file. If you do not specify an output file name, BitGen creates a .bit file in your input file directory. If you specify any of the following options, the corresponding file is created in addition to the .bit file. If you do not specify an extension, BitGen appends the correct one for the specified option.

| Option | Output File |
|---|---|
| –l | *outfile_name*.ll |
| –m | *outfile_name*.msk |
| –b | *outfile_name*.rbt |

A report file containing all BitGen output is automatically created under the same directory as the output file. The report file has the same root name as the output file and a .bgn extension.

*Pcf_file* is the name of a physical constraints file. BitGen uses this file to interpret CONFIG constraints, which control bitstream options. These CONFIG constraints override default behavior and can be overridden by configuration options. See "–g (Set Configuration)." BitGen automatically reads the .pcf file by default. If the PCF is the second file specified on the command line, it must have a .pcf extension. If it is the third file specified, the extension is optional; .pcf is assumed. If a .pcf file name is specified, it must exist; otherwise, the input design name with a .pcf extension is assumed.

Type the following syntax to see a complete list of BitGen command line options and supported devices:

```
bitgen –h
```

# BitGen Input Files

Input to BitGen comprises the following files:

- NCD file—a physical description of the design mapped, placed and routed in the target device. The NCD file must be fully routed.
- PCF—an optional user-modifiable ASCII Physical Constraints File.

# BitGen Output Files

Output from BitGen comprises the following files:

*Table 2-1:* **BitGen Output Files**

| Output File Type | Output File Description |
|---|---|
| .bgn | Contains log information for the BitGen run, including command line options, errors, and warnings. Always produced. |
| .bin | A binary file that contains only configuration data. The .bin has no header like the .bit file. Produced when –g Binary:Yes is specified. |
| .bit | A binary file that contains proprietary header information as well as configuration data. Meant for input to other Xilinx tools, such as PROMGen and iMPACT. Always produced unless the -j option is specified. |
| .drc | A design rule check (DRC) file for the design. Contains log information or Design Rules Checker, including errors and warnings. Always produced unless the -d option is specified. |
| .isc | Contains the configuration data in IEEE1532 format. Produced when -g IEEE:1532:Yes is specified. |
| .ll | An ASCII file that contains information on each of the nodes in the design that can be captured for readback. The file contains the absolute bit position in the readback stream, frame address, frame offset, logic resource used, and name of the component in the design. Produced when the -l option is specified. |
| .msd | An ASCII file that contains only mask information for verification, including pad words and frames. No commands are included. Produced when -g Readback is specified. |

*Table 2-1:* **BitGen Output Files *(Continued)***

| Output File Type | Output File Description |
|---|---|
| .msk | A binary file that contains the same configuration commands as a .bit file, but has mask data where the configuration data is. This data should NOT be used to configure the device. If a mask bit is 0, that bit should be verified against the bitstream data. If a mask bit is 1, that bit should not be verified. Produced when the -m option is specified. |
| .rba | An ASCII file that contains readback commands, rather than configuration commands, and expected readback data where the configuration data would normally be.<br>To produce the .rba file, the –b option must be used when –g Readback is specified. |
| .rbb | The same as the .rba file, but it is a binary file.<br>Produced when –g Readback is specified. |
| .rbd | An ASCII file that contains only expected readback data, including pad words and frames. No commands are included. Produced when -g Readback is specified. |
| .rbt | An ASCII version of the bit file. Produced when the -b option is specified. |

# BitGen Options

Following is a description of the command line options and how they affect the behavior of BitGen.

## –b (Create Rawbits File)

Create a *rawbits* (*file_name*.rbt) file. If the –g Readback option is specified in combination with the –b option, an ASCII readback command file (*file_name*.rba) is also generated.

The rawbits file consists of ASCII ones and zeros representing the data in the bitstream file. If you are using a microprocessor to configure a single FPGA, you can include the rawbits file in the source code as a text file to represent the configuration data. The sequence of characters in the rawbits file is the same as the sequence of bits written into the FPGA.

## –bd (Update Block RAMs)

```
-bd file_name
```

The –bd option updates the bitstream with the block RAM content from the specified MEM file.

## –d (Do Not Run DRC)

Do not run DRC (design rule check). Without the –d option, BitGen runs a DRC and saves the DRC results in two output files: the BitGen report file (*file_name*.bgn) and the DRC file (*file_name*.drc). If you enter the –d option, no DRC information appears in the report file and no DRC file is produced.

Running DRC before a bitstream is produced detects any errors that could cause the FPGA to malfunction. If DRC does not detect any errors, BitGen produces a bitstream file (unless you use the –j option described in "–j (No BIT File)").

## –f (Execute Commands File)

**–f** *command_file*

The –f option executes the command line arguments in the specified *command_file*.

## –g (Set Configuration)

The –g option has sub-options that represent settings you use to set the configuration for a design. These options have the following syntax:

**bitgen -g** *option*:*setting design*.**ncd** *design*.***bit*** *design*.***pcf***

For example, to enable Readback, use the following syntax:

```
bitgen -g Readback
```

The following sections describe the options for the –g option.

### ActivateGCLK

Allows any partial bitstream for a reconfigurable area to have its registered elements wired to the correct clock domain.

| | |
|---|---|
| Settings: | No, Yes |
| Default: | No |

### ActiveReconfig

Prevents the assertions of GHIGH and GSR during configuration. This is required for the active partial reconfiguration enhancement features.

| | |
|---|---|
| Settings: | No, Yes |
| Default: | No |

### Binary

Creates a binary file with programming data only. Use this option to extract and view programming data. Any changes to the header will not affect the extraction process.

| | |
|---|---|
| Settings: | No, Yes |
| Default: | No |

## CclkPin

Adds an internal pull-up to the Cclk pin. The Pullnone setting disables the pullup.

| | |
|---|---|
| Settings: | Pullnone, Pullup |
| Default: | Pullup |

## Compress

This option reduces the size of the bitstream, not just the .bit file. Using the Compress option does not guarantee that the size of the bitstream will shrink. Compression is enabled by setting the BitGen option **`-g compress`**; compression is disabled by not setting it.

| | |
|---|---|
| Settings: | None |
| Default: | Off |

## ConfigRate

Spartan-3 FPGAs use an internal oscillator to generate the configuration clock, CCLK, when configuring in a master mode. Use the configuration rate option to select the rate for this clock.

| | |
|---|---|
| Settings: | 6, 3, 12, 25, 50 |
| Default: | 6 |

## CRC

The CRC option controls the generation of a Cyclic Redundancy Check value in the bitstream. When enabled, a unique CRC value is calculated based on bitstream contents. If the calculated CRC value does not match the CRC value in the bitstream, the device will fail to configure. When CRC is disabled a constant value is inserted in the bitstream in place of the CRC and the device will not calculate a CRC.

| | |
|---|---|
| Settings: | Disable, Enable |
| Default: | Enable |

## DCIUpdateMode

This option controls how often the Digitally Controlled Impedance circuit attempts to update the impedance match for DCI IOSTANDARDs.

| | |
|---|---|
| Settings: | AsRequired, continuous, quiet |
| Default: | AsRequired |

## DCMShutdown

When DCMShutdown is enabled, the digital clock manager (DCM) resets if the SHUTDOWN and AGHIGH commands are loaded into the configuration logic.

| | |
|---|---|
| Settings: | Disable, Enable |
| Default: | Disable |

## DebugBitstream

If the device does not configure correctly, you can debug the bitstream using the DebugBitstream option. A debug bitstream is significantly larger than a standard bitstream. The values allowed for the DebugBitstream option are No and Yes.

***Note:*** Use this option only if your device is configured to use slave or master serial mode

| | |
|---|---|
| Values: | No, Yes |

In addition to a standard bitstream, a debug bitstream offers the following features:

- Writes 32 0s to the LOUT register after the synchronization word
- Loads each frame individually
- Performs a cyclical redundancy check (CRC) after each frame
- Writes the frame address to the LOUT register after each frame

## DONE_cycle

Selects the Startup phase that activates the FPGA Done signal. Done is delayed when DonePipe=Yes.

| | |
|---|---|
| Settings: | 1, 2, 3, 4, 5, 6 |
| Default: | 4 |

## DonePin

Adds an internal pull-up to the DONE pin. The Pullnone setting disables the pullup.

Use this option only if you are planning to connect an external pull-up resistor to this pin. The internal pull-up resistor is automatically connected if you do not use this option.

| | |
|---|---|
| Settings: | Pullup, Pullnone |
| Default: | Pullup |

## DonePipe

This option is intended for use with FPGAs being set up in a high-speed daisy chain configuration.When set to Yes, the FPGA waits on the CFG_DONE (DONE) pin to go High and then waits for the first clock edge before moving to the Done state.

Settings: No, Yes

Default: No

## DriveDone

This option actively drives the DONE Pin High as opposed to using a pullup.

Settings: No, Yes

Default: No

## GWE_cycle

Selects the Startup phase that asserts the internal write enable to flip-flops, LUT RAMs, and shift registers. It also enables the BRAMs. Before the Startup phase both BRAM writing and reading are disabled.The Done setting asserts GWE when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes. The Keep setting is used to keep the current value of the GWE signal.

Settings: 1, 2, 3, 4, 5, 6, Done, Keep

Default: 6

## GTS_cycle

Selects the Startup phase that releases the internal 3-state control to the I/O buffers. The Done setting releases GTS when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes.

Settings: Done, 1, 2, 3, 4, 5, 6, Keep

Default: 5

## HswapenPin

Adds a pull-up, pull-down, or neither to the HSWAP_EN pin. The Pullnone option shows there is no connection to either the pull-up or the pull-down.

Settings: Pullup, Pulldown, Pullnone

Default: Pullup

### IEEE1532

Creates IEEE1532 configuration file (`.isc`).

Settings: No, Yes

Default: No

### LCK_cycle

Selects the Startup phase to wait until DLLs/DCMs lock. If NoWait is selected, the Startup sequence does not wait for DLLs/DCMs.

Settings: 0,1, 2, 3, 4, 5, 6, NoWait

Default: NoWait

### M0Pin

The M0 pin is used to determine the configuration mode. Adds an internal pull-up, pull-down or neither to the M0 pin. The following settings are available. The default is PullUp. Select Pullnone to disable both the pull-up resistor and pull-down resistor on the M0 pin.

Settings: Pullup, Pulldown, Pullnone

Default: Pullup

### M1Pin

The M1 pin determines the configuration mode. Adds an internal pull-up, pull-down or neither to the M1 pin. The following settings are available. The default is PullUp.

Select Pullnone to disable both the pull-up resistor and pull-down resistor on the M1 pin.

Settings: Pullup, Pulldown, Pullnone

Default: Pullup

### M2Pin

The M2 pin determines the configuration mode. Adds an internal pull-up, pull-down or neither to the M2 pin. The default is Pullup. Select Pullnone to disable both the pull-up resistor and pull-down resistor on the M2 pin.

Settings: Pullup, Pulldown, Pullnone

Default: Pullup

## Match_cycle

Specifies a stall in the Startup cycle until digitally controlled impedance (DCI) match signals are asserted.

Settings: Auto, NoWait, 0, 1, 2, 3, 4, 5, 6

Default: Auto

***Note:*** When the Auto setting is specified, BitGen searches the design for any DCI I/O standards. If DCI standards exist, BitGen will use the Match_cycle:2 setting, otherwise it will use the Match_cycle:NoWait setting.

## PartialGCLK

Adds the center global clock column frames into the list of frames to write out in a partial bitstream. This option is equivalent to the PartialMask0:1 option.

Default: <Not Specified> - no partial masks in use

## PartialMask0, PartialMask1, PartialMask2

Generates a bitstream comprised of only the major addresses of block type *<0, 1, or 2>* that have enabled value in the mask. The block type is all non-block ram initialization data frames in the applicable device and its derivatives. The mask is a hex value.

Settings: All columns enabled, major address mask

Default: <Not Specified> - no partial masks in use

## PartialLeft

Adds the left side frames of the device into the list of frames to write out in a partial bitstream. This includes CLB, IOB, and BRAM columns. It does not include the center global clock column.

## PartialRight

Adds the right side frames of the device into the list of frames to write out in a partial bitstream. This includes CLB, IOB, and BRAM columns. It does not include the center global clock column.

## Persist

This option is needed for Readback and Partial Reconfiguration using the SelectMAP configuration pins. If Persist is set to Yes, the pins used for SelectMAP mode are prohibited for use as user I/O. Refer to the datasheet for a description of SelectMAP mode and the associated pins.

Settings: No, Yes

Default: No

### ProgPin

Adds an internal pull-up to the PROG_B pin. The Pullnone setting disables the pullup. The pull-up affects the pin after configuration.

| | |
|---|---|
| Settings: | Pullup, Pullnone |
| Default: | Pullup |

### ReadBack

This option allows you to perform the Readback function by creating the necessary readback files.

When specifying the –g Readback option, the .rbb, .rbd, and .msd files are created.

If the –b option is used in conjunction with the –g Readback option, an ASCII readback command file (*file_name*.rba) is also generated.

### Security

Selecting Level1 disables Readback. Selecting Level2 disables Readback and Partial Reconfiguration.

| | |
|---|---|
| Settings: | None, Level1, Level2 |
| Default: | None |

### StartupClk

The startup sequence following the configuration of a device can be synchronized to either Cclk, a User Clock, or the JTAG Clock. The default is Cclk.

- Cclk

  Enter Cclk to synchronize to an internal clock provided in the FPGA device.

- JTAG Clock

  Enter JtagClk to synchronize to the clock provided by JTAG. This clock sequences the TAP controller which provides the control logic for JTAG.

- UserClk

  Enter UserClk to synchronize to a user-defined signal connected to the CLK pin of the STARTUP symbol.

| | |
|---|---|
| Settings: | Cclk (pin—see Note), UserClk (user-supplied), JtagCLK |
| Default: | Cclk |

***Note:*** In modes where Cclk is an output, the pin is driven by an internal oscillator.

### TckPin

Adds a pull-up, a pull-down or neither to the TCK pin, the JTAG test clock. Selecting one setting enables it and disables the others. The Pullnone setting shows there is no connection to either the pull-up or the pull-down.

Settings: Pullup, Pulldown, Pullnone

Default: Pullup

### TdiPin

Adds a pull-up, a pull-down, or neither to the TDI pin, the serial data input to all JTAG instructions and JTAG registers. Selecting one setting enables it and disables the others. The Pullnone setting shows there is no connection to either the pull-up or the pull-down.

Settings: Pullup, Pulldown, Pullnone

Default: Pullup

### TdoPin

Adds a pull-up, a pull-down, or neither to the TDO pin, the serial data output for all JTAG instruction and data registers. Selecting one setting enables it and disables the others. The Pullnone setting shows there is no connection to either the pull-up or the pull-down.

Settings: Pullup, Pulldown, Pullnone

Default: Pullup

### TmsPin

Adds a pull-up, pull-down, or neither to the TMS pin, the mode input signal to the TAP controller. The TAP controller provides the control logic for JTAG. Selecting one setting enables it and disables the others. The Pullnone setting shows there is no connection to either the pull-up or the pull-down

Settings: Pullup, Pulldown, Pullnone

Default: Pullup

### UnusedPin

Adds a pull-up, a pull-down, or neither to the unused device pins and the serial data output (TDO) for all JTAG instruction and data registers. Selecting one setting enables it and disables the others. The Pullnone setting shows there is no connection to either the pull-up or the pull-down.

The following settings are available. The default is Pulldown.

Settings: Pullup, Pulldown, Pullnone

Default: Pulldown

### UserID

You can enter up to an 8-digit hexadecimal code in the User ID register. You can use the register to identify implementation revisions.

| | |
|---|---|
| Settings: | 0xFFFFFFFF, [*hex string]* |
| Default: | 0xFFFFFFFF |

## –j (No BIT File)

Do not create a bitstream file (.bit file). This option is used when you want to generate a report without producing a bitstream. For example, if you wanted to run DRC without producing a bitstream file, you would use the -j option.

***Note:*** The .msk or .rbt files may still be created.

## –l (Create a Logic Allocation File)

This option creates an ASCII logic allocation file (*design*.ll) for the selected design. The logic allocation file shows the bitstream position of latches, flip-flops, IOB inputs and outputs, and the bitstream position of LUT programming and Block RAMs.

In some applications, you may want to observe the contents of the FPGA internal registers at different times. The file created by the –l option helps you identify which bits in the current bitstream represent outputs of flip-flops and latches. Bits are referenced by frame and bit number within the frame.

The iMPACT tool uses the design.ll file to locate signal values inside a readback bitstream.

## –m (Generate a Mask File)

Creates a mask file. This file determines which bits in the bitstream should be compared to readback data for verification purposes.

## –r (Create a Partial Bit File)

–r *bit_file*

The –r option is used to create a partial bit file. It takes that bit file and compares it to the .ncd file given to bitgen. Instead of writing out a full bit file, it only writes out the part of the bit file that is different from the original bit file given.

## –w (Overwrite Existing Output File)

Enables you to overwrite an existing BitGen output file. See "BitGen Output Files" for additional information.

# Printed Circuit Board Design Considerations

## Package Drawings

## Using IBIS Models for Spartan-3 FPGAs

## Using BSDL Files for Spartan-3 FPGAs


SPARTAN™-3
FPGA
MAKE IT YOUR ASIC

# Package Drawings

This section contains the package drawings for the packages used by the Spartan-3 FPGA family:

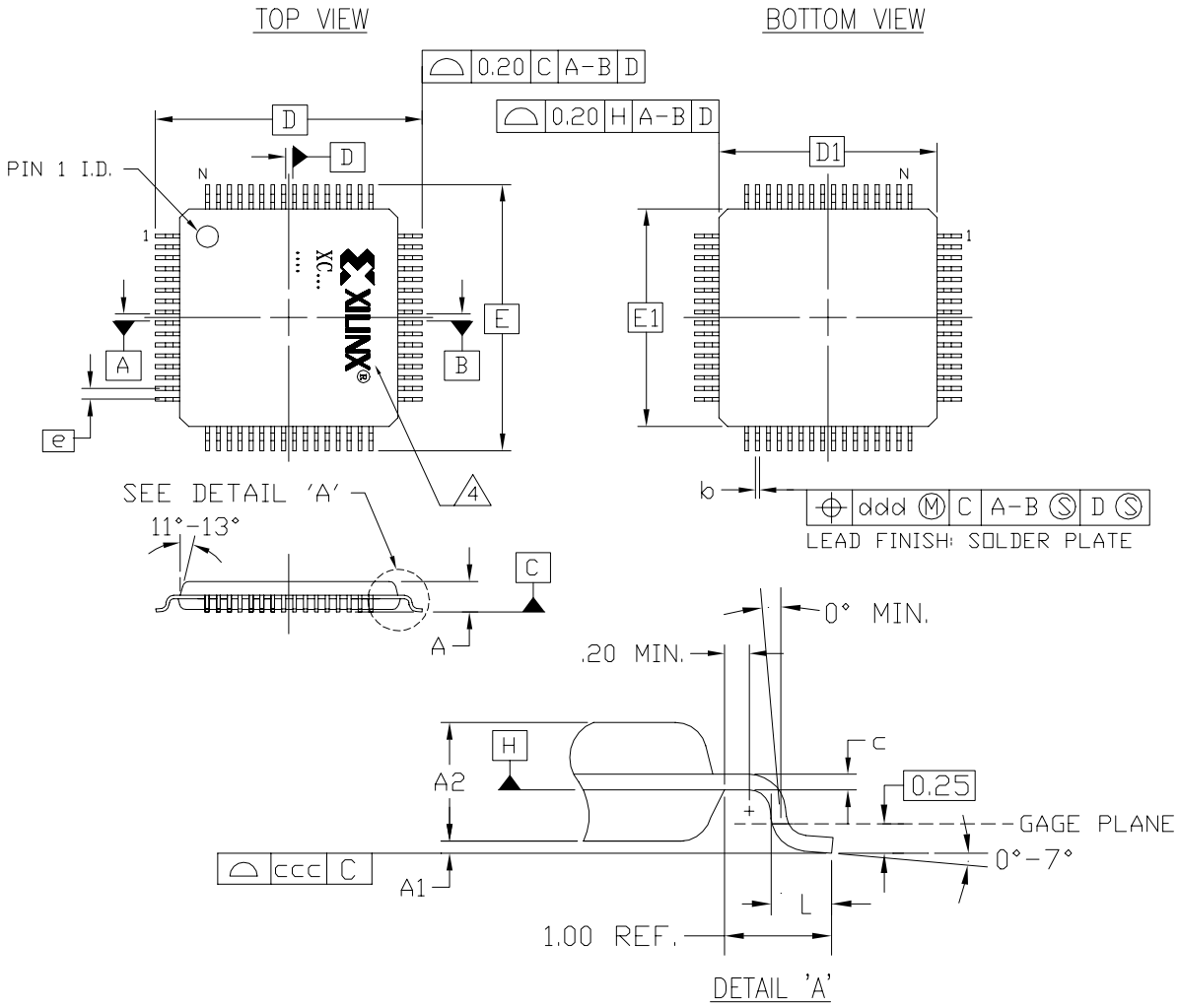
- VQ100: Very Thin Quad Flat Pack
- TQ144: Thin Quad Flat Pack
- PQ208: Plastic Quad Flat Pack
- FT256: Fine-Pitch Thin Ball Grid Array
- FG456: Fine-Pitch Ball Grid Array
- FG676: Fine-Pitch Ball Grid Array
- FG900: Fine-Pitch Ball Grid Array
- FG1156: Fine-Pitch Ball Grid Array

For the latest version of these drawings, see the following Web page:
http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Package+Drawings

XILINX®

PK012 (v1.1) October 17, 2002

# VQFP (VQ100) Package



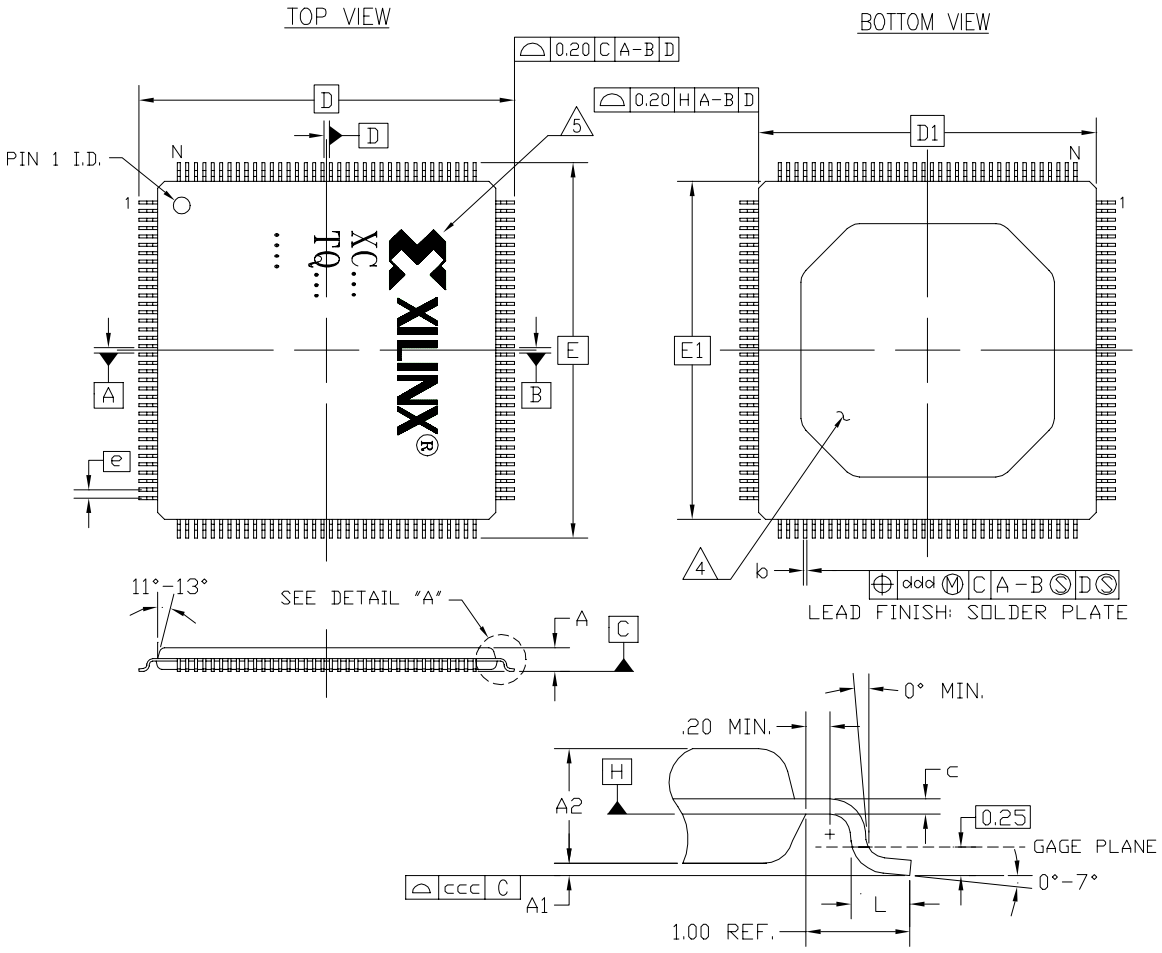

| VQ100 | | |
|---|---|---|
| MILLIMETERS | | |
| MIN. | NOM. | MAX. |
| | | 1.20 |
| 0.05 | 0.10 | 0.15 |
| 0.95 | 1.00 | 1.05 |
| 16.00 BSC. | | |
| 14.00 BSC. | | |
| 0.17 | 0.22 | 0.27 |
| 0.09 | | 0.20 |
| 0.50 BSC. | | |
| 0.45 | 0.60 | 0.75 |
| | | 0.08 |
| | | 0.08 |
| 100 | | |
| JEDEC MS-026-AED | | |

NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M-1994.
2. DIMENSIONS "D1" AND "E1" DO NOT INCLUDE MOLD PROTRUSIONS. ALLOWABLE MOLD PROTRUSIONS SHALL NOT EXCEED 0.25mm PER SIDE.
3. THE TOP OF THE PACKAGE MAY BE SMALLER THAN THE BOTTOM OF THE PACKAGE BY 0.15mm.
4. MARK ORIENTATION WITH RESPECT TO PIN-1 I.D. IF 2 OR MORE CIRCLES EXIST ON TOP OF THE PACKAGE, USE THE SMALLEST CIRCLE AS PIN-1 I.D.

PK009 (v1.1) October 17, 2002

# TQFP (TQ144) Package







DETAIL 'A'

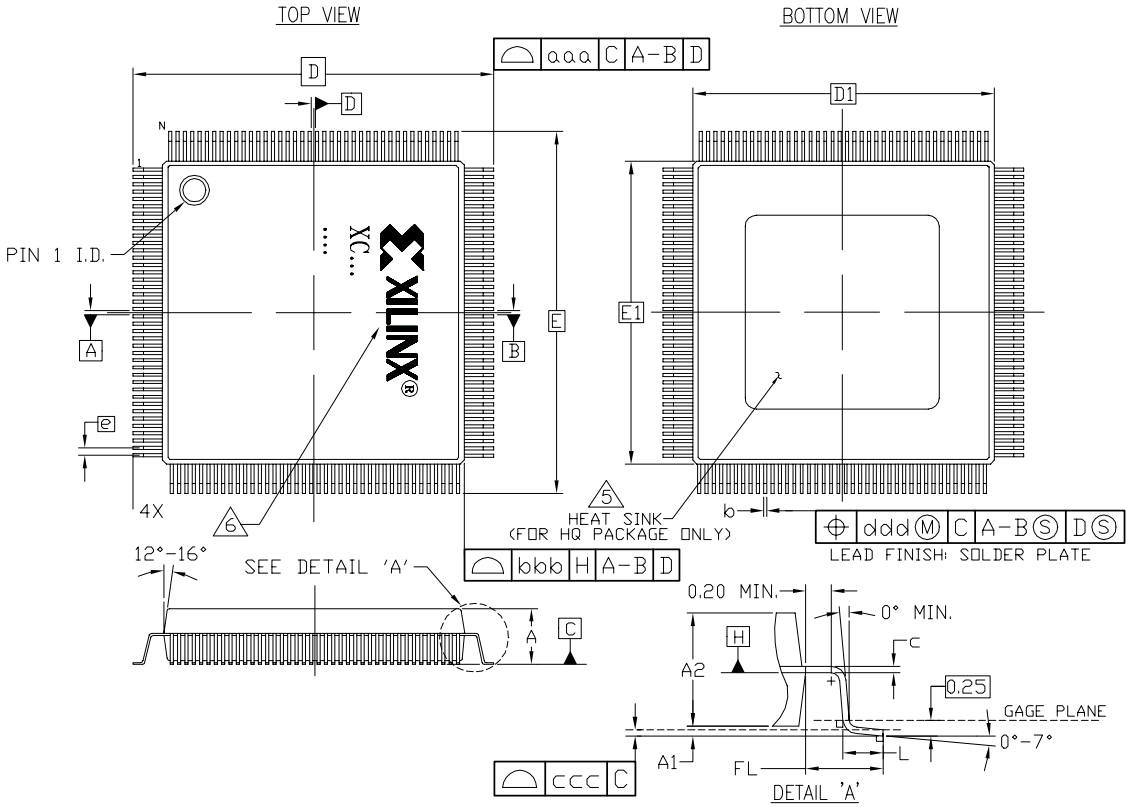
| TQ/HT144 | | |
|---|---|---|
| MILLIMETERS | | |
| MIN. | NOM. | MAX. |
| | | 1.60 |
| 0.05 | 0.10 | 0.15 |
| 1.35 | 1.40 | 1.45 |
| 22.00 BSC | | |
| 20.00 BSC | | |
| 0.45 | 0.60 | 0.75 |
| 0.50 BSC | | |
| 0.17 | 0.22 | 0.27 |
| 0.09 | | 0.20 |
| | | 0.08 |
| | | 0.08 |
| 144 | | |
| JEDEC MS-026-BFB | | |

NOTE:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5-1994
2. DIMENSION D1 AND E1 DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE MOLD PROTRUSION SHALL NOT EXCEED 0.25mm PER SIDE.
3. PACKAGE TOP DIMENSION MAY BE SMALLER THAN THE BOTTOM DIMENSION BY 0.15mm.
4. THE SAME PACKAGE DIMENSIONS APPLY FOR THERMALLY ENHANCED PRODUCTS. HEAT SINK IS ADDED. THE PACKAGE CODE IS "HT".
   HEAT SINK DIMENSIONS: HT100=10.0mm SQ. REF.
   HT144=15.5mm SQ. REF.
   HT176=19.4mm SQ. REF.
5. MARK ORIENTATION WITH RESPECT TO PIN-1 I.D. IF 2 OR MORE CIRCLES EXIST ON TOP OF THE PACKAGE, USE THE SMALLEST CIRCLE AS PIN-1 I.D.

XILINX®
PK007 (v1.1) April 6, 2001

# PQFP (PQ208) Package



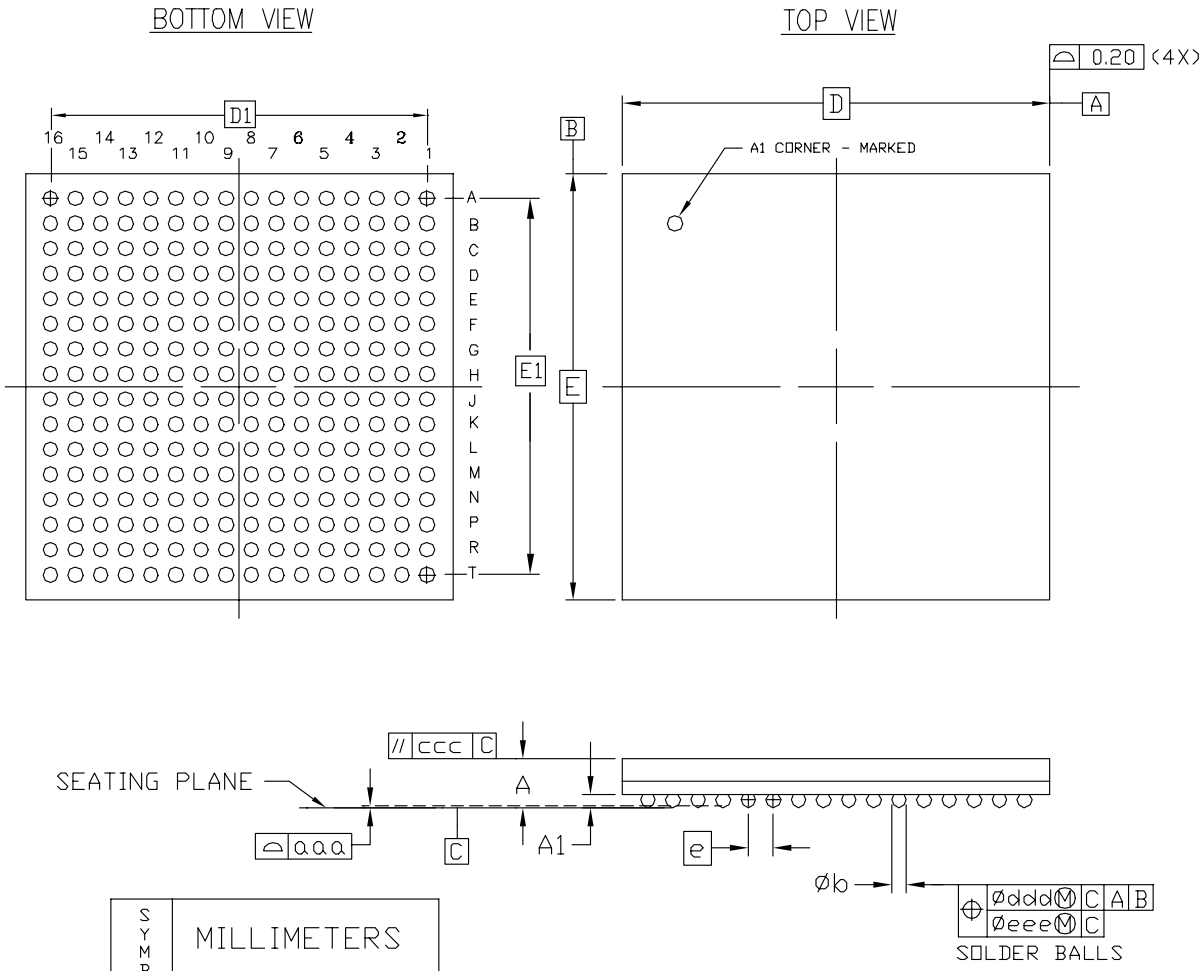

| PQ/HQ208 | | |
|---|---|---|
| MILLIMETERS | | |
| MIN. | NOM. | MAX. |
| | 3.70 | 4.10 |
| 0.25 | 0.33 | 0.50 |
| 3.20 | 3.40 | 3.60 |
| 30.60 BSC | | |
| 28.00 BSC | | |
| 1.30 REF. | | |
| 0.50 | 0.60 | 0.75 |
| 0.50 BSC. | | |
| 0.17 | 0.22 | 0.27 |
| 0.09 | | 0.20 |
| | | 0.25 |
| | | 0.20 |
| | | 0.08 |
| | | 0.08 |
| 208 | | |
| JEDEC MS-029-FA-1 | | |

NOTES:
1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M-1994.
2. DIMENSIONS D1 AND E1 DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE MOLD PROTRUSION SHALL NOT EXCEED 0.25mm PER SIDE.
3. PACKAGE TOP DIMENSIONS MAY BE SMALLER THAN THE BOTTOM DIMENSIONS BY 0.20mm.
4. THE SAME PACKAGE DIMENSIONS APPLY FOR THERMALLY ENHANCED PRODUCTS. HEAT SINK IS ADDED. THE PACKAGE CODE IS "HQ".
5. HEAT SINK DIMENSIONS: HQ160/HQ208=21mm SQ. REF.; HQ240=24mm SQ. REF.
6. MARK ORIENTATION WITH RESPECT TO PIN-1 I.D. IF 2 OR MORE CIRCLES EXIST ON TOP OF THE PACKAGE, USE THE SMALLEST CIRCLE AS PIN-1 I.D.

XILINX®

PK053 (v1.0) April 6, 2001

# Fine Pitch Thin BGA (FT256) Package




| SYMBOL | MILLIMETERS | | |
|---|---|---|---|
| | MIN. | NOM. | MAX. |
| A | --- | 1.40 | 1.55 |
| $A_1$ | 0.35 | 0.40 | 0.50 |
| D/E | 17.00 BSC | | |
| $D_1/E_1$ | 15.00 REF | | |
| e | 1.00 BSC | | |
| Øb | 0.40 | 0.50 | 0.60 |
| aaa | --- | --- | 0.15 |
| ccc | --- | --- | 0.15 |
| ddd | --- | --- | 0.25 |
| eee | --- | --- | 0.10 |
| M | 16 | | |

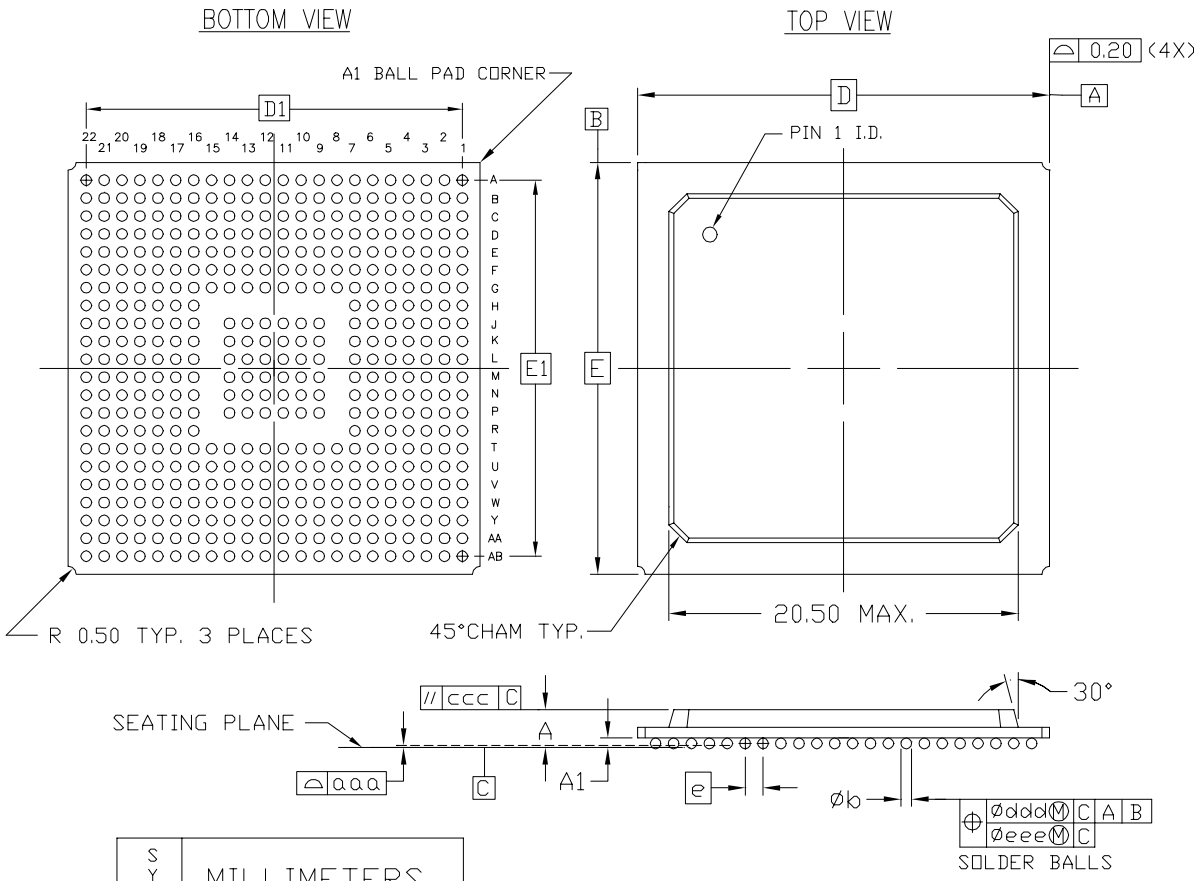
NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ASME Y14.5M-1994
2. SYMBOL 'M' IS THE BALL MATRIX SIZE.
3. CONFORMS TO JEDEC MS-034 AAF-1 EXCEPT FOR THE SIZE OF BALL

256-BALL FINE PITCH THIN BGA (FT256)

XILINX®
PK034 (v1.1) April 6, 2001

# Fine Pitch BGA (FG456) Package




| SYMBOL | MILLIMETERS | | |
|---|---|---|---|
| | MIN. | NOM. | MAX. |
| A | | 2.20 | 2.60 |
| $A_1$ | 0.40 | 0.50 | 0.60 |
| D/E | 23.00 BSC | | |
| $D_1/E_1$ | 21.00 REF | | |
| e | 1.00 BSC | | |
| øb | 0.50 | 0.60 | 0.70 |
| aaa | | | 0.20 |
| ccc | | | 0.35 |
| ddd | | | 0.30 |
| eee | | | 0.10 |
| M | 22 | | |

NOTES:

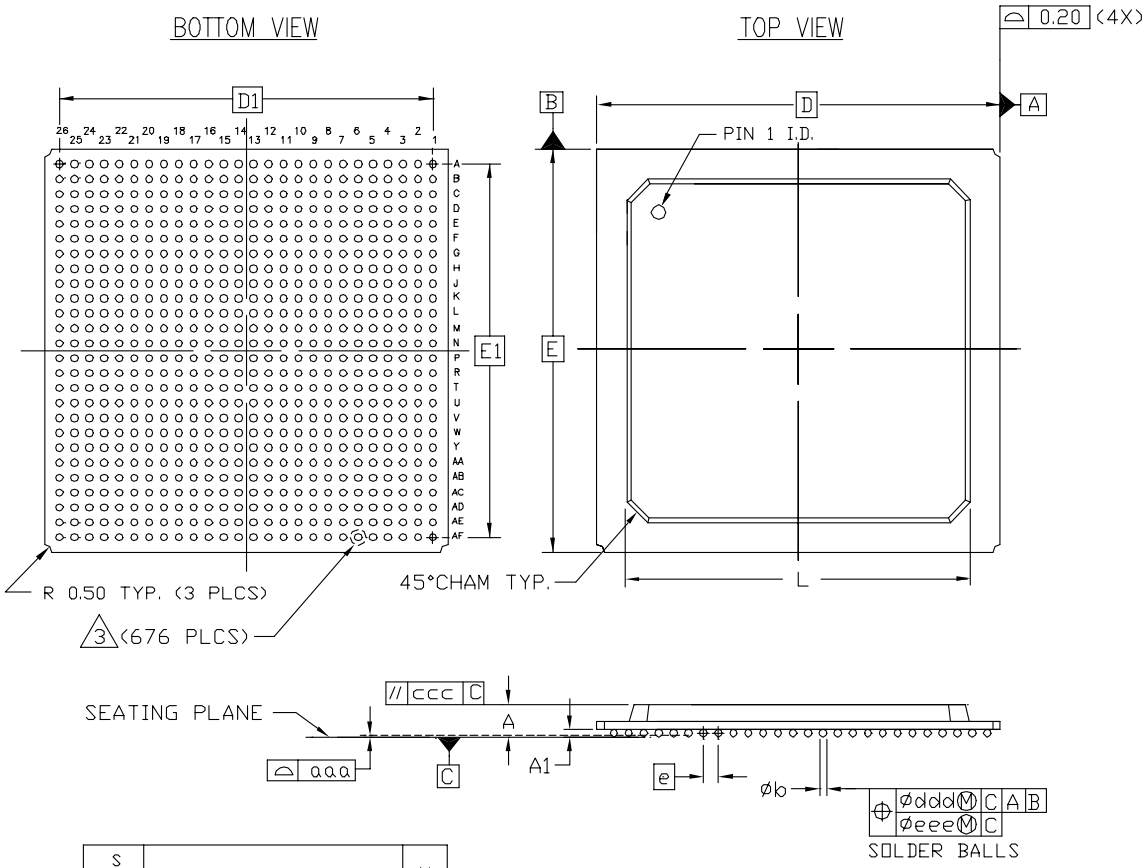
1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M–1994
2. SYMBOL 'M' IS THE BALL MATRIX SIZE.
3. CONFORMS TO JEDEC MS–034–AAJ–1 (DEPOPULATED)

456–BALL FINE PITCH BGA (FG456)

XILINX®

PK035 (v1.1) April 6, 2001

# Fine Pitch BGA (FG676) Package




| SYMBOL | MILLIMETERS | | | NOTE |
|---|---|---|---|---|
| | MIN. | NOM. | MAX. | |
| A | | 2.25 | 2.60 | |
| $A_1$ | 0.40 | 0.50 | 0.60 | |
| D/E | 27.00 BSC | | | |
| $D_1/E_1$ | 25.00 REF | | | |
| e | 1.00 BSC | | | |
| øb | 0.50 | 0.60 | 0.70 | |
| aaa | | | 0.20 | |
| ccc | | | 0.35 | |
| ddd | | | 0.30 | |
| eee | | | 0.10 | |
| L | | | 25.70 | |
| M | 26 | | | 2 |
| REF. | JEDEC MS-034-AAL-1 | | | |

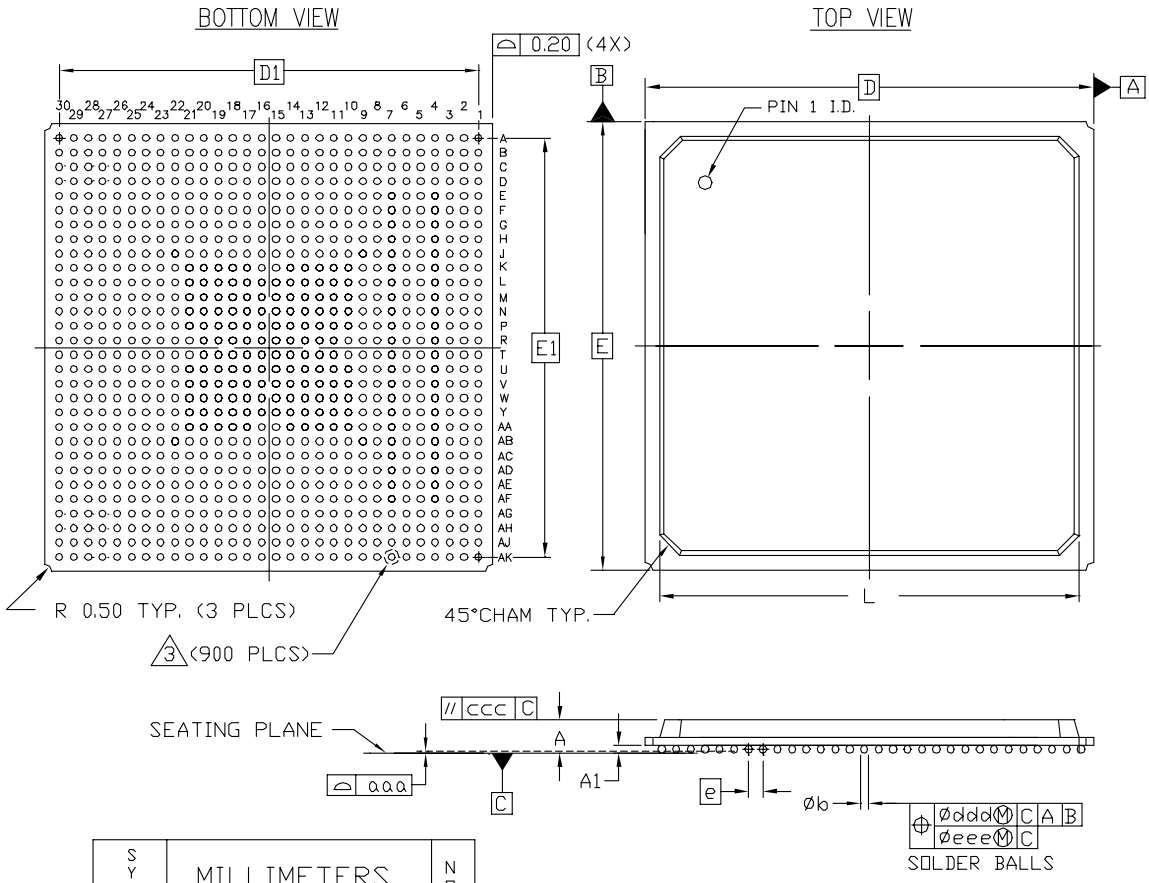
NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M-1994
2. SYMBOL 'M' IS THE BALL MATRIX SIZE.
3. LAND PAD OPENING – SOLDER MASK DEFINED ø0.485mm (0.019")

676-BALL FINE PITCH BGA (FG676)

XILINX®
PK038 (v1.1) April 6, 2001

# Fine Pitch BGA (FG900) Package




| SYMBOL | MILLIMETERS | | | NOTE |
|---|---|---|---|---|
| | MIN. | NOM. | MAX. | |
| A | | 2.25 | 2.60 | |
| $A_1$ | 0.40 | 0.50 | 0.60 | |
| D/E | 31.00 BSC | | | |
| $D_1/E_1$ | 29.00 REF | | | |
| e | 1.00 BSC | | | |
| øb | 0.50 | 0.60 | 0.70 | |
| aaa | | | 0.20 | |
| ccc | | | 0.35 | |
| ddd | | | 0.30 | |
| eee | | | 0.10 | |
| L | | | 29.05 | |
| M | 30 | | | 2 |
| REF. | JEDEC MS-034-AAN-1 | | | |

NOTES:

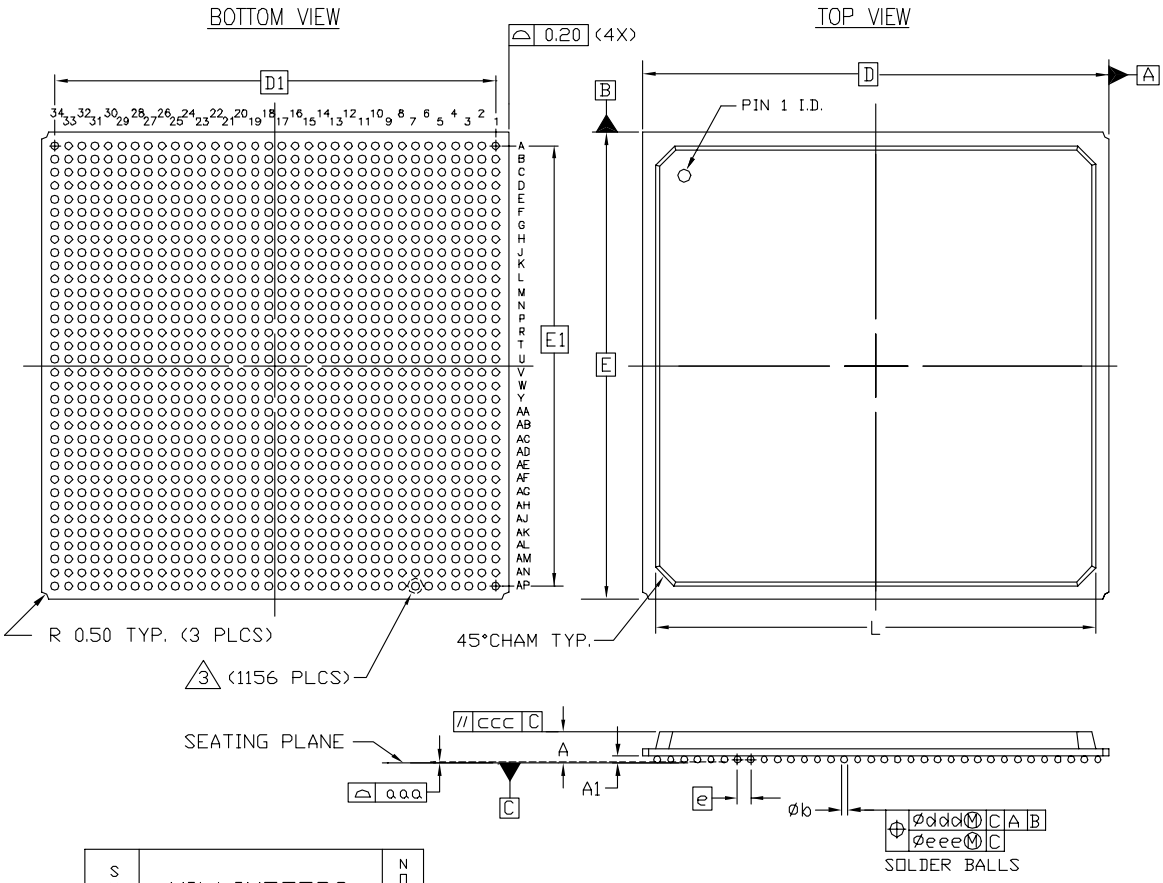
1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M-1994
2. SYMBOL 'M' IS THE BALL MATRIX SIZE.
3. LAND PAD OPENING – SOLDER MASK DEFINED ø0.485mm (0.019")

900-BALL FINE PITCH BGA (FG900)

XILINX®

PK039 (v1.1) April 6, 2001

# Fine Pitch BGA (FG1156) Package




| SYMBOL | MILLIMETERS | | | NOTE |
|---|---|---|---|---|
| | MIN. | NOM. | MAX. | |
| A | | 2.33 | 2.60 | |
| $A_1$ | 0.40 | 0.50 | 0.60 | |
| D/E | 35.00 BSC | | | |
| $D_1/E_1$ | 33.00 REF | | | |
| e | 1.00 BSC | | | |
| øb | 0.50 | 0.60 | 0.70 | |
| aaa | | | 0.20 | |
| ccc | | | 0.35 | |
| ddd | | | 0.30 | |
| eee | | | 0.10 | |
| L | | | 33.05 | |
| M | 34 | | | 2 |
| REF. | JEDEC MS-034-AAR | | | |

NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M-1994
2. SYMBOL 'M' IS THE BALL MATRIX SIZE.
3. LAND PAD OPENING - SOLDER MASK DEFINED ø0.485mm (0.019")

1156-BALL FINE PITCH BGA (FG1156)

XAPP475 (v1.0) June 21, 2003

# Using IBIS Models for Spartan-3 FPGAs


## Summary

Input/Output Buffer Information Specification (IBIS) models are industry-standard descriptions used to simulate I/O characteristics in board-level design simulation. IBIS models for Spartan™-3 devices are available at http://www.xilinx.com/support/sw_ibis.htm. The models can be used with third-party simulation tools to verify proper signal integrity characteristics in board designs.


## Introduction

As I/O switching frequencies have increased and voltage levels have decreased, accurate analog simulation of I/Os has become an essential part of modern high-speed digital system design. By accurately simulating the I/O buffers, termination, and circuit board traces, designers can significantly shorten their time-to-market of new designs. Identifying signal integrity related issues at the beginning of the design cycle decreases the required number of board fixes and increases quality.

The device data sheets provide basic information about guaranteed DC and switching characteristics of the I/Os. However, the data sheet does not include all the information required to determine the best board layout for a particular application, such as slew rates and drive strength, which are included in the IBIS model. Designers can use IBIS models for system-level analysis of signal integrity issues, such as ringing, ground bounce, crosstalk, and RFI/EMI. Complete designs can be simulated and evaluated before going through the expensive and time consuming process of producing prototype PCBs. This type of pre-layout simulation can reduce considerably the development cost and time to market, while increasing the reliability of the I/O operation.

## IBIS Advantages over SPICE

Traditionally SPICE analysis has been used extensively in areas like IC design, where a high level of accuracy is required. However in the PCB and systems domain, there are several disadvantages to the SPICE method, both for the device vendor and the user.

Since SPICE simulations model a circuit at transistor level, it is necessary for the SPICE models to contain detailed information about the circuit and process parameters. For most IC vendors, this type of information is regarded as proprietary.

Although SPICE simulation accuracy is typically very good, a significant limitation with any simulation method is simulation speed. Simulation speeds are particularly slow for transient simulation analysis, which is most often used when evaluating signal integrity performance. SPICE simulation has a further disadvantage in that not all SPICE simulators are fully compatible. Often, default simulator options are not the same in different SPICE simulators. As there are some very powerful options that control accuracy, convergence and the algorithm type, any options that are not consistent might give rise to poor correlation in simulation results across different simulators. Also, because of the different variants of SPICE, these models are often incompatible between simulators, thus models must be extracted for a specific simulator.

Xilinx SPICE models are only available upon completion of a Non-Disclosure Agreement (NDA) and application evaluation process; therefore it is recommended that IBIS models be used wherever possible. See more information on SPICE at: http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?ipoid=66136&sSecondaryNavPick=Design+Tools&sGlobalNavPick=PRODUCTS

## IBIS Background

IBIS, originally developed by Intel, is an alternative to SPICE simulation. The IBIS specification now is maintained by the EIA/IBIS Open Forum, which has members from a large number of IC and EDA vendors. IBIS is the ANSI/EIA-656 and IEC 62014-1 standard. For more information about the IBIS specification, see http://www.eigroup.org/ibis/ibis.htm.

The core of the IBIS model consists of a table of current versus voltage and timing information. This is very attractive to the IC vendor as the I/O internal circuit is treated as a black box. This way, transistor-level information about the circuit and process details is not revealed.

IBIS models can be used to model best-case and worst-case conditions (best-case = strong transistors, low temperature, high voltage; worst-case = weak transistors, high temperature, low voltage). The “fast/strong” model represents best-case conditions, while the "slow/weak" model represents worst-case conditions. The "typical" model represents typical behavior.

IBIS cannot be used for internal timing information (propagation delays and skew); the timing models instead provide that information. IBIS also does not model power and ground structures or pin-to-pin coupling. The implications are that ground bounce, power supply droop, and simultaneous switching output (SSO) noise cannot be simulated with IBIS models. Instead, Xilinx provides device/package-dependent SSO guidelines once extensive lab measurements are completed. IBIS models also do not provide detailed package parasitic information. Package parasitics usually are provided in the form of lumped RLC data, which loses its accuracy at higher speeds. To model the package parasitics accurately, include a transmission line with a delay of 25 ps to 100 ps and an impedance of 65Ω.

Using IBIS models has a great advantage to the user in that simulation speed is significantly increased over SPICE, while accuracy is only slightly decreased. Non-convergence, which can be a problem with SPICE models and simulators, is eliminated in IBIS simulation. Virtually all EDA vendors presently support IBIS models, and ease of use of these IBIS simulators is generally very good. IBIS models for most devices are freely available over the Internet making it easy to simulate several different manufacturers’ devices on the same board. Several different IBIS simulators are available today, and each simulator provides different results. An overshoot or undershoot of ±10% of the measured result is tolerable. Differences between the model and measurements occur because not all parameters are modeled. Simulators for IBIS models are provided by Cadence, Avanti Corporation, Hyperlynx, Mentor, Microsim, Intusoft, Veribest, and Viewlogic. See the links to third-party IBIS tools at: http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=si_simulation.

## Xilinx Support of IBIS

Xilinx provides IBIS models for all current products; they are downloaded easily from our website at http://www.xilinx.com/support/sw_ibis.htm. The models also are made available in the development system. The Preliminary models are based initially on simulation and then verified against the silicon.

An IBIS file contains two sections, the header and the model data for each component. One IBIS file can describe several devices. The following is the content list in a typical IBIS file:

- IBIS Version
- File Name
- File Revision
- Component
- Package R/L/C

- Pin name, model, R/L/C
- Model (i.e., 3-state)
- Temperature Range (typical, minimum, and maximum)
- Voltage Range (typical, minimum, and maximum)
- Pull-Up Reference
- Pull-Down Reference
- Power Clamp Reference
- Ground Clamp Reference
- I/V Tables for:
  - Pull-Up
  - Pull-Down
  - Power Clamp
  - Ground Clamp
- Rise and Fall dV/dt for minimum, typical, and maximum conditions (driving 50Ω)

## IBIS I/V and dV/dt Curves

A digital buffer can be measured in receive (3-state) mode and drive mode. IBIS I/V curves are based on the data of both these modes. The transition between modes is achieved by phasing in/out the difference between the driver and the receiver models, while keeping the receiver model constantly in the circuit.

The I/V curve range required by the IBIS specification is $-V_{CC}$ to (2x $V_{CC}$). This wide voltage range exists because the theoretical maximum overshoot due to a full reflection is twice the signal swing. The ground clamp I/V curve must be specified over the range $-V_{CC}$ to $V_{CC}$, and the power clamp I/V curve must be specified from $V_{CC}$ to (2x $V_{CC}$).

The three supported conditions for the IBIS buffer models are typical values (required), minimum values (optional), and maximum values (optional). For CMOS buffers, the minimum condition is defined as high temperature and low supply voltage, and the maximum condition is defined as low temperature and high supply voltage.

An IBIS model of a digital buffer has four I/V curves:

- The pull-down I/V curve contains the mode data for the driver driving low. The origin of the curve is at 0V for CMOS buffers.
- The pull-up I/V curve contains the mode data for the driver driving high. The origin of the curve is at the supply voltage ($V_{CC}$).
- The ground clamp I/V curve contains receive (3-state) mode data. The origin of the curve is at 0V for CMOS buffers.
- The power clamp I/V curve contains receive (3-state) mode data. The origin of the curve is at the supply voltage ($V_{CC}$).

## Ramp and dV/dt Curves

The Ramp keyword contains information on how fast the pull-up and pull-down transistors turn on/off. The dV/dt curves give the same information, while including the effects of die capacitance (C_comp). C_comp is the total die capacitance as seen at the die pad, excluding the package capacitance.

dV/dt curves describe the transient characteristics of a buffer more accurately than ramps. A minimum of four dV/dt curves are required to describe a CMOS buffer: pull-down ON, pull-up OFF, pull-down OFF, and pull-up ON. dV/dt curves incorporate the clock-to-out delay, and the length of the dV/dt curve corresponds to the clock speed at which the buffer is used. Each dV/dt curve has t = 0, where the pulse crosses the input threshold.

## Xilinx IBIS Package Parasitic Modeling

Xilinx IBIS modeling previously used a simple RCL model for the pin and bond wire parasitics. Due to the fast rise and fall times of many of the supported I/O standards, it was deemed necessary to improve the package parasitic modeling. The latest IBIS 3.2 specification has a complex parasitic package model, which incorporates a transmission line and lumped RCL model. Unfortunately, IBIS 3.2 still is not widely supported by simulators.

For these reasons, **the old lumped package parasitic parameters have been removed from the latest models, and the user must add manually an external transmission line**. A 65Ω ideal transmission line, with the delay set between 25 ps to 100 ps, is recommended. This configuration works in conjunction with a revised lumped model (included inside the IBIS model). For critical applications, both extremes (25 ps and 100 ps) should be checked; however, for most I/O applications this difference is very small.

## IBISWriter

A Xilinx IBIS file downloaded from the Web contains a collection of IBIS models for all I/O standards available in the targeted device. ISE can generate IBIS models specific to your design via the IBISWriter tool, simplifying design export into signal integrity analysis tools. IBISWriter associates IBIS buffer models to each pin of the customer design according to the design specification for each I/O buffer. IBISWriter outputs an IBS file that can be used directly as an input file to your signal integrity analysis tool.

Generating design-specific IBIS files requires only three easy steps:

1. Implement your design in Project Navigator.
2. In the Process View window, under Implement Design/Place & Route, select Generate IBIS Model and click Run. A design-specific file is generated where all input/output pins are associated with an IBIS model.
3. Incorporate this file onto your favorite signal integrity analysis tool to perform the desired simulations.

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 06/21/03 | 1.0 | Initial Xilinx release. |

# Using BSDL Files for Spartan-3 FPGAs

XAPP476 (v1.0) July 10, 2003


## Summary

BSDL (Boundary Scan Description Language) files are provided for every part and package combination of IEEE 1149.1 (JTAG) compatible devices produced by Xilinx, including all the Spartan-3 FPGAs. Third-party boundary scan tools use these files to generate test vectors and perform the tests. Xilinx programming software also uses BSDL files when configuring devices through Boundary Scan.


## Boundary Scan Overview

Boundary Scan testing is used to identify faulty board-level connections, such as unconnected or shorted pins. Boundary Scan tests allow designers to quickly identify manufacturing or layout problems, which otherwise could be nearly impossible to isolate, especially with high-count ball-grid packages. More recently, PLD vendors such as Xilinx have made use of boundary scan as a convenient way of configuring devices, including the Spartan-3 FPGA family.

## IEEE Standards

JTAG (Joint Test Action Group) is the commonly used name for IEEE standard 1149.1, which defines a method for Boundary Scan. JTAG compliant devices have dedicated hardware that comprises a state machine and several registers to allow boundary scan operations. This dedicated hardware interprets instructions and data provided by four dedicated signals: TDI (Test Data In), TDO (Test Data Out), TMS (Test Mode Select), and TCK (Test Clock). The JTAG hardware interprets instructions and data on the TDI and TMS signals, and drives data out on the TDO signal. The TCK signal is used to clock the process.

IEEE 1532 is a superset of the IEEE 1149.1 JTAG standard. IEEE 1532 provides additional flexibility for configuring programmable logic devices. IEEE Std 1532 enables designers to concurrently program multiple devices, minimize programming times with enhanced silicon features, and produce robust systems that are more easily maintained. This standard defines the three additional items required to configure in-system programmable logic devices:

- Device architectural components for configuration
- Algorithm description framework
- Configuration data file

General information on the IEEE 1532 JTAG standard is available on the Xilinx website at http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=isp_standards_specs#1532.

## Boundary Scan Tools

Boundary Scan testing requires specialized test equipment and software. The Boundary Scan test software is used to generate test vectors, which are typically delivered to the boundary scan chain using a test pod connected to a PC.

To develop vectors for boundary scan testing, the test software must be provided with information about the scan chain:

1. The composition of the scan chain - how many devices, what type, and so forth.

The chain composition can be either specified by the user or automatically detected by the boundary scan software.

2. The Boundary Scan architecture of each device - the Instruction Register length, opcodes, number of I/Os, and how each of those I/Os behaves.

   The Boundary Scan architecture of each device is defined in a Boundary Scan Description Language (BSDL) file.

3. How the device I/Os are connected to each other.

   This information typically is extracted from a board-level netlist.

## BSDL Files

Any manufacturer of a JTAG-compliant device must provide a BSDL file for that device. The BSDL file contains information on the function of each of the pins on the device - which are used as I/Os, which are power or ground, and so forth. All Xilinx BSDL files have file extensions of `.bsd`.

BSDL files for Xilinx devices are available in the development system and on the Xilinx website at http://www.xilinx.com/support/sw_bsdl.htm. BSDL files for other manufacturers typically can be found on the manufacturer's website.

Prototype files for the IEEE 1532 extension to the BSDL files are also available for Xilinx products. They are included in the development system and are available at http://www.xilinx.com/isp/1532download.htm.

IEEE 1149.1 BSDL files appear as: `<device_name>.bsd`

For example: `xc3s50_pq208.bsd`

IEEE 1532 BSDL files appear as: `<device_name>_1532.bsd`

For example: `xc3s50_pq208_1532.bsd`

Note that prototype IEEE Std 1532 BSDL files should not be used in place of or alongside 1149.1 BSDL files.

### BSDL File Composition

BSDL files describe the Boundary Scan architecture of a JTAG-compliant device, and are written in VHDL. There are eight main parts of a BSDL file:

1. Entity Declaration

   The entity declaration is a VHDL construct used to identify the name of the device that is described by the BSDL file.

   Example (from the `xc3s50_pq208.bsd` file):

   ```
   entity XC3S50_PQ208 is
   ```

2. Generic Parameter

   The Generic parameter specifies which package the BSDL file describes.

   Example (from the `xc3s50_pq208.bsd` file):

   ```
   generic (PHYSICAL_PIN_MAP : string := "PQ208" );
   ```

3. Logical Port Description

   The Logical Port Description lists all of the pads on a device, and states whether that pin is an input (`in bit;`), output (`out bit;`), bidirectional (`inout bit;`) or unavailable for boundary scan (`linkage bit;`).

Example (from the xc3s50_pq208.bsd file):

```
port (
  GND: linkage bit_vector (1 to 28);
  CCLK_P104: inout bit;
  DONE_P103: inout bit;
  HSWAP_EN_P206: in bit;
  M0_P55: in bit;
  M1_P54: in bit;
  M2_P56: in bit;
  PROG_B: in bit;
  TCK: in bit;
  TDI: in bit;
  TDO: out bit;
  TMS: in bit;
  VCCAUX: linkage bit_vector (1 to 8);
  VCCINT: linkage bit_vector (1 to 4);
  VCCO0: linkage bit_vector (1 to 2);
  IO_P2: inout bit; --  PAD124
  IO_P3: inout bit; --  PAD123
```

4. Package Pin Mapping

   The Package Pin Mapping shows how the pads on the device die are wired to the pins on the device package.

   Example (from the xc3s50_pq208.bsd file):

```
constant PQ208: PIN_MAP_STRING:=
  "GND:(P1,P8,P14,P25,P30,P41,P47,P53,P59,P66," &
    "P75,P82,P91,P99,P105,P112,P118,P129,P134,P145," &
    "P151,P157,P163,P170,P179,P186,P195,P202)," &
  "CCLK_P104:P104," &
  "DONE_P103:P103," &
  "HSWAP_EN_P206:P206," &
  "M0_P55:P55," &
  "M1_P54:P54," &
  "M2_P56:P56," &
  "PROG_B:P207," &
  "TCK:P159," &
  "TDI:P208," &
  "TDO:P158," &
  "TMS:P160," &
  "VCCAUX:(P17,P38,P69,P89,P121,P142,P173,P193)," &
  "VCCINT:(P70,P88,P174,P192)," &
  "VCCO0:(P188,P201)," &
  "IO_P2:P2," &
  "IO_P3:P3," &
```

5. use statements

   The use statement calls VHDL packages that contain attributes, types, constants, and others that are referenced in the BSDL File.

   Example (from the xc3s50_pq208.bsd file):

```
  use STD_1149_1_1994.all;
```

6. Scan Port Identification

   The Scan Port Identification identifies the JTAG pins: TDI, TDO, TMS, TCK, and TRST (if used). TRST is an optional JTAG pin that is not used by Xilinx devices.

Example (from the xc3s50_pq208.bsd file):

```
attribute TAP_SCAN_IN    of TDI : signal is true;
attribute TAP_SCAN_MODE  of TMS : signal is true;
attribute TAP_SCAN_OUT   of TDO : signal is true;
attribute TAP_SCAN_CLOCK of TCK : signal is (33.0e6, BOTH);
```

7. TAP description

   The TAP description provides additional information on the device's JTAG logic. Some of this information includes the Instruction Register length, Instruction Opcodes, and device IDCODE. These characteristics are device specific, and may vary widely from device to device.

   Examples (from the xc3s50_pq208.bsd file):

```
attribute COMPLIANCE_PATTERNS of XC3S50_PQ208 : entity is
      "(PROG_B) (1)";
attribute INSTRUCTION_LENGTH of XC3S50_PQ208 : entity is 6;
attribute INSTRUCTION_OPCODE of XC3S50_PQ208 : entity is
      "EXTEST    (000000)," &
attribute INSTRUCTION_CAPTURE of XC3S50_PQ208 : entity is
      "XXXX01";
attribute IDCODE_REGISTER of XC3S50_PQ208 : entity is
  "XXXX" &      -- version
  "0001010" &   -- family
  "000001100" &    -- array size
  "00001001001" &  -- manufacturer
  "1";          -- required by 1149.1
```

8. Boundary Register description

   The Boundary Register description gives the structure of the Boundary Scan cells on the device. Each pin on a device may have up to three Boundary Scan cells, each cell consisting of a register and a latch. Boundary Scan test vectors are loaded into or scanned from these registers.

   Example (from the xc3s50_pq208.bsd file):

```
attribute BOUNDARY_REGISTER of XC3S50_PQ208 : entity is
    "   0 (BC_2, *, controlr, 1)," &
    "   1 (BC_2, IO_P161, output3, X, 0, 1, PULL0)," & --  PAD30
    "   2 (BC_2, IO_P161, input, X)," & --  PAD30
```

## BSDL File Verification

Xilinx verification of the supplied BSDL files has two levels. Preliminary files are generated using an automated, Xilinx-standard, BSDL generation process. The process is script-based and extracts information directly from the device design files, which fully describe the architecture and pinout. The quality of "Preliminary" BSDL files is very high, and the syntax is always tested. Files marked "Final" have undergone all the tests for syntax and hardware verification. To guarantee that the BSDL files exactly describe the operation of each pin, Xilinx uses an independent third-party boundary-scan tool vendor — Intellitech — to verify the actual silicon against the BSDL.

Xilinx BSDL files are checked for 1149.1 compliance with the Intellitech Eclipse product using 'strict' BSDL syntax checking. Every semantic check described in the IEEE 1149.1b-1994 (the standard for BSDL syntax) is performed using strict parsing. Test patterns then are generated from the BSDL file that include unique tests for every I/O pin. Each Xilinx device/package combination is tested on the Intellitech RCT (Reduced Contact Tester). The test patterns include verification of Test-Logic-Reset and TAP controller operation, BYPASS/IDCODE/USERCODE instructions and registers, and pin mapping of the boundary register to every input/output/bidir/clock pin and control cell. Finally, each device is tested for

1149.1 compliance after the device is programmed by downloading a design and using the RCT tester to verify post configuration compliance.

### Using BSDLAnno for Post-Configuration Boundary Scan Behavior

Whenever possible, boundary scan tests should be performed on an unconfigured Spartan-3 device. Unconfigured devices allow for better test coverage, since all I/Os are available for bidirectional scan vectors. Boundary Scan tests should be performed after configuration when configuration cannot be prevented and when differential signaling standards are used. If the differential signals are located between Xilinx devices, both devices can be tested pre-configuration. Each side of the differential pair will behave as a single-ended signal.

The BSDL files provided by Xilinx reflect the Boundary Scan behavior of an unconfigured device. After configuration, the boundary scan behavior of a device changes. I/O pins that were bidirectional before configuration may now be input-only, output-only, bidirectional, or unavailable. Boundary Scan test vectors typically are derived from BSDL files, so if boundary scan tests are going to be performed on a configured Xilinx device, the BSDL file must be modified to reflect the device's configured Boundary Scan behavior.

The boundary scan architecture changes after the device is configured because the boundary scan registers sit behind the I/O buffer and sense amplifier. The hardware is arranged in this way so that the boundary scan logic operates at the I/O standard specified by the design. This allows boundary scan testing across the entire range of available I/O standards.

Because certain connections between the boundary scan registers and pad may change, the boundary scan architecture is effectively changed when the device is configured. These changes often need to be communicated to the boundary scan tester through a post-configuration BSDL file. If the changes to the boundary scan architecture are not reflected in the BSDL file, boundary scan tests may fail.

Xilinx offers the BSDLAnno utility to automatically modify the BSDL file for post-configuration testing. BSDLAnno obtains the necessary design information from the routed `.ncd` file and generates a BSDL file that reflects the post-configuration boundary scan architecture of the device.

Use the following syntax to generate a post-configuration BSDL file with BSDLAnno:

```
bsdlanno [options] infile[.ncd] outfile[.bsd]
```

The `infile` is the routed (post-PAR) NCD design source file for the specified design. The `outfile[.bsd]` is the destination for the design-specific BSDL file. The `.bsd` extension is optional. For more details on BSDLanno, see Answer 15346 at http://www.xilinx.com/xlnx/xil_ans_display.jsp?i&getPagePath=15346.

## Software Support

Xilinx offers several tools for generating device files and for device programming. Boundary Scan test functionality is available from several third-party vendors, as noted under Configuration Solutions on www.xilinx.com.

### iMPACT

iMPACT is a full featured software tool used for configuration and programming of all Xilinx FPGAs, CPLDs, and PROMs. It features a series of "wizard" dialogs that easily guide the user through the every step of the configuration process. iMPACT supports a host of output file types including SVF. iMPACT configuration software enables users to easily configure all Xilinx FPGAs using three different modes; slave serial, SelectMAP (Slave Parallel), and JTAG IEEE 1149.1. iMPACT supports the Parallel Cable IV and MultiPRO cables.

iMPACT features a special function in the JTAG mode to test both the operation of the cable and the robustness of the JTAG chain. The user can test chain operation by instructing iMPACT to write to and read back from the user code location multiple thousands of times. It then counts

the number of errors that occur in this operation. This gives the user the opportunity to evaluate the relative robustness of the JTAG chain and the susceptibility to noise and other influences like board layout.

### SVF Files

Serial Vector Format (SVF) is an industry-standard file format that is used to describe JTAG chain operations in a compact, portable fashion. SVF files capture all of the device specific programming information within the SVF instructions. SVF files are useful because intricate knowledge of the device is not needed. The capability to create SVF files is included in the iMPACT tool. See the *iMPACT User Guide* for more details.

### J Drive Engine for IEEE 1532 Programming

Xilinx has developed and introduced the world's first IEEE Std 1532 Programming Engine: J Drive™ Engine. Using this engine and a simple cable connected to the parallel port of any PC, users can easily configure Xilinx IEEE Std 1532 compatible PLDs. The designer provides J Drive Engine with the data and 1532 BSDL files for the device(s) to be programmed using a command line interface to configure the PLDs in the JTAG chain. For more information, see http://www.xilinx.com/xlnx/xil_prodcat_systemsolution.jsp?title=isp_jdrivemain_page.

### Using the BSCAN_SPARTAN3 Macro

BSCAN_SPARTAN3 provides access to the BSCAN sites on a Spartan-3 device. It is used to create internal boundary scan chains. The four-pin JTAG interface (TDI, TDO, TCK, and TMS) contains dedicated pins in Spartan-3 FPGAs. To use normal JTAG for boundary scan purposes, just hook up the JTAG pins to the port and go. The pins on the BSCAN_SPARTAN3 symbol do not need to be connected unless those special functions are needed to drive an internal scan chain.

Spartan-3 FPGAs provide hooks for two user-definable scan chains through the USER1 and USER2 instructions. These instructions may be used to provide access to the user design through the JTAG interface. To take advantage of the optional USER1 and USER2 instructions, the designer must instantiate the BSCAN_SPARTAN3 macro in the source code, and wire it to the user-defined scan chain.

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 07/10/03 | 1.0 | Initial Xilinx release. |

# Appendices

## Xilinx XAPP Application Notes

## Glossary


SPARTAN™-3
FPGA
MAKE IT YOUR ASIC

XILINX®

*Appendix A*

# Xilinx XAPP Application Notes

This appendix summarizes all application notes that are relevant to the Spartan-3 FPGA family. The latest versions of these documents are available on the Xilinx website at http://www.xilinx.com/apps/sp3app.htm.

*Table A-1:* **Application Note Summary for Spartan-3 FPGA Family**

| Application Note Number | Title | Description |
|---|---|---|
| **Memory** | | |
| XAPP201 | An Overview of Multiple CAM Designs | Flexible CAMs (Content Addressable Memory) are implemented by taking advantage of the reprogrammability of the basic LUT as a Shift Register or a SelectRAM memory and the fast carry logic chain. |
| XAPP228 | Quad-Port Memories | Describes how the dual-port block memories can be used as Quad-Port memories. This essentially involves a data access time (halved) versus functionality (doubled) trade-off. The overall bandwidth of the block memory in terms of bits per second will remain the same. |
| XAPP258 | FIFOs Using Block RAM | Describes a way to create a common-clock (synchronous) version and an independent-clock (asynchronous) version of a 511 x 36 FIFO, with the depth and width being adjustable within the Verilog or VHDL code. |
| XAPP260 | Using Block RAM for High-Performance Read/Write CAMs | Content Addressable Memory (CAM) offers increased data search speed. This innovative design is suited for small embedded Cams with high-speed match and write requirements. The reference design is built using the true dual-port block SelectRAM+™ feature. |
| XAPP261 | Data-Width Conversion FIFOs Using Block RAM | Describes how to create a common-clock (synchronous) version and an independent-clock (asynchronous) version of a FIFO for data-width conversion with different width read and write data ports. |

*Table A-1:* **Application Note Summary for Spartan-3 FPGA Family**

| Application Note Number | Title | Description |
|---|---|---|
| XAPP291 | Self-Addressing FIFO | A self-addressing FIFO reference design uses block memories to store both data and address information in a single memory location. No external counters are required; only flag and status information logic is used. Their advantage is in using only one clock load. In addition, the status mechanism is very simple, making FIFOs more suitable for data throttling in continuous data systems instead of the full or empty detection required in frame based data systems. |
| **LFSRs** | | |
| XAPP211 | Pseudo-Random Noise Generators Using the SRL Macro | Pseudo-random Noise (PN) generators are at the heart of every spread spectrum system. Many PN generators are required within CDMA base stations. PN generators are based upon LFSRs. Xilinx devices implement efficient LFSRs and deliver a significant reduction in resource utilization when compared with alternative flip-flop only PLD structures. For example, a 16-stage LFSR can be realized in just one LUT. |
| XAPP217 | Gold Code Generators | Gold code generators are used extensively in CDMA systems to generate code sequences with good correlation properties. Describes the implementation of Gold code generators in Spartan-II devices. The Gold code generators use efficiently implemented LFSRs using the SRL16 macro. |
| XAPP220 | LFSRs as Functional Blocks in Wireless Applications | Describes two implementations of an LFSR using the SRL16 primitive for area-efficient designs. The first LFSR implementation describes the parallel output access and parity calculation; the second describes the multi-cycle output access and sequential parity calculation. |
| **Multipliers** | | |
| XAPP284 | Matrix Math, Graphics, and Video | Describes a unique way to implement a 3 x 3-matrix multiplier using a Virtex™-II device. By running multipliers at multiples of the system clock rate in slower applications, silicon resources can be leveraged. |
| XAPP636 | Optimal Pipelining of the I/O Ports of Virtex-II Multipliers | This application note and reference design describes a high-speed, optimized implementation of a Virtex-II pipelined multiplier primitive. |

*Table A-1:* **Application Note Summary for Spartan-3 FPGA Family**

| Application Note Number | Title | Description |
|---|---|---|
| **Microcontrollers** | | |
| XAPP213 | PicoBlaze™ 8-Bit Microcontroller | The Constant Coded Programmable State Machine (PicoBlaze) solution is a fully embedded 8-bit microcontroller macro. The module is remarkably small at just 35 CLBs. This PicoBlaze solution provides 49 different instructions at 35 million instructions per second. This performance exceeds that of traditional discrete microcontroller devices, providing a cost-attractive solution for data processing as well as control algorithms. Fully embedded including the program memory, the PicoBlaze solution can be connected to many other functions and peripherals tuned to a specific design. |
| **Configuration** | | |
| XAPP501 | Configuration Quick Start Guidelines | Discusses configuration and programming options for Xilinx CPLD, FPGA, and PROM families and demonstrates some of the most popular configuration methods. |
| XAPP502 | Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Modes | In embedded systems designers can reduce component count and increase flexibility by using a microprocessor to configure an FPGA. C code illustrates using either Slave Serial or SelectMAP mode. CPLD design files illustrate a synchronous interface between processor and FPGA. |
| **Power** | | |
| XAPP623 | Power Distribution System (PDS) Design: Using Bypass/Decoupling Capacitors | Covers principles of power distribution systems and bypass or decoupling capacitors. A step-by-step process is described where a power distribution system can be designed and verified. The final section discusses additional sources of power supply noise and provides resolutions. |
| XAPP653 | 3.3V PCI Reference Design | 3.3V regulator reference design to work with 3.3V I/O pins using the PCI I/O standard. |

*Table A-1:* **Application Note Summary for Spartan-3 FPGA Family**

| Application Note Number | Title | Description |
|---|---|---|
| **Timing** | | |
| XAPP259 | System Interface Timing Parameters | Defines timing parameters required for the analysis of source synchronous and system synchronous applications. Explains the DCM clock phase accuracy parameters, system-synchronous pin-to-pin setup/hold with DCM parameters, and all source-synchronous parameters. Memory interface analyses are provided as examples. |
| XAPP268 | Active Phase Alignment | The DCM allows fine phase adjustment of an incoming clock. Normally the DCM is set up to provide a constant phase shift that allows the incoming data to be correctly clocked in. This phase shift is corrected for both temperature and voltage, but can vary slightly across different devices and wafer lots, thus effectively reducing slightly the receiver window or "eye". One way of correcting for this is to set up the DCM phase shift dynamically via training either at device reset or on a continuous basis. Systems using both single and double data rate reception can use the concepts in this application note. |
| XAPP622 | SDR LVDS Transmitter/Receiver | Describes Single Data Rate (SDR) transmitter and receiver interfaces using 17 LVDS pairs (one clock and 16 data channels). Describes a method of implementing an SDR interface at clock frequencies higher than the maximum operating frequency of the DCM. |

XILINX®

*Appendix B*

# Glossary

## Numeric

**3-state**

Includes high, low and disabled states, also known as three-state

**90nm**

Feature size in Spartan-3 process technology, 1 nanometer (nm) is 1 billionth of a meter

## A

**advance specification**

Initial estimates of characteristics based on simulation, subject to change.

**AllianceCORE**

IP created and supported by third parties

**AllianceEDA**

Third-party partners supplying Electronic Design Automation tools

**Alliance**

Xilinx ISE development system for interfacing to third-party partners

**ASIC**

Application Specific Integrated Circuit

**ASSP**

Application Specific Standard Product

## B

**bank**

Groups of I/Os with a common $V_{REF}$ and/or $V_{CCO}$

**BaseX**

Xilinx low-cost ISE development system

**BGA**

Ball Grid Array package

**BitGen**

Bitstream Generator tool

**bitstream**

FPGA configuration file

**block RAM**

Dedicated embedded 18K memory blocks

**BLVDS**

Bus LVDS for bidirectional signals

**boundary scan**

Serial test protocol defined by IEEE 1149.1 standard

**BRAM**

See "block RAM"

**BSP**

Board Support Package

**BUFG**

Global clock buffer

**BUFGCE**

Global clock buffer with enable input

**BUFGMUX**

Global clock multiplexer buffer

## C

**CAM**

Content-Addressable Memory

**CCLK**

Configuration Clock

**ChipScope**

In-circuit real-time debugging tool

**CLB**

Configurable Logic Block

**CLKIN**

Clock Input to DCM

**commercial**

Junction temperature operating range 0-85 °C

**configuration**

Process of programming an FPGA or the program itself

**constraints**

Implementation parameters defined by user

**CORE Generator**

Software tool providing a set of ready-made functions as IP

**CPLD**

Complex Programmable Logic Device

## D

**daisy chain**

FPGAs connected in series in order to configure from a single source

**Data2BRAM**

Block RAM contents definition tool

**DCI**

Digitally Controlled Impedance on-chip termination, also known as XCITE

**DCM**

Digital Clock Manager

**DCM Wizard**

DCM design entry tool

**DDR**

Double Data Rate I/O registers

**DFS**

Digital Frequency Synthesizer part of the DCM

**differential**

Signaling scheme using two complementary signals to transmit data, such as LVDS

**DIN**

Data Input, FPGA configuration input

**direct line**

Routing that connects adjacent CLBs

**distributed RAM**

16x1 RAM blocks built from LUTs

**DLL**

Delay Locked Loop part of the DCM

**D-MIPS**

Dhrystone MIPS or Dhrystone Million Instructions Per Second processor performance benchmark

**DONE**

Device pin that indicates end of configuration process

**double line**

Routing that connects to one of every two CLBs

**DOUT**

Data Out, FPGA configuration output

**DSP**

Digital Signal Processing

**dual port**

Two independent data ports allowing access to same memory

## E

**ECS**

Engineering Capture System; Xilinx schematic entry tool

**EDIF**

Electronic Design Interchange Format, industry standard for specifying a logic design in text

**EDK**

Embedded Development Kit for MicroBlaze embedded CPU

**ESD**

Electro-Static Discharge

**EST**

Embedded Systems Tools

## F

**F5MUX**

CLB multiplexer that combines two LUTs to create any 5-input function

**FAE**

Field Application Engineer

**FDDR**

DDR Flip-flop

**FF**

Flip-Flop, edge-sensitive storage element

**FG or FBGA**

Fine-pitch ball grid array package

**FIFO**

First In, First Out memory

**FiMUX**

F6MUX, F7MUX, or F8MUX CLB multiplexer

**Flash**

Reprogrammable, non-volatile memory technology

**Floorplanner**

Tool for graphical analysis and specification of placement

**Forge**

High-level logic description language using Java syntax

**Foundation**

Xilinx ISE development system with integrated design entry and verification tools

**FPC**

Field-Programmable Controller (eg. MicroBlaze or PicoBlaze FPC)

**FPGA**

Field-Programmable Gate Array

**FPGA Editor**

Tool for graphical analysis and specification of placement and routing

**FSM**

Finite State Machine

**FT or FTBGA**

Fine-pitch thin ball grid array package

**function generator**

4-input look-up table, also known as LUT

## G

**GCLK**

Global Clock input

**GNU**

Pronounced "guh-NEW". Free embedded development tools

**GSR**

Global Set/Reset

**GTL**

Gunning Transceiver Logic I/O standard

**GTLP**

GTL Plus I/O standard

**GTS**

Global Three-State

**GWE**

Global Write Enable

## H

**HDL**

Hardware Description Language

**hex line**

Routing that connects to one of every three CLBs

**Hoplite**

A soldier in the Spartan army

**HSTL**

High-Speed Transceiver Logic I/O standard

**HSWAP_EN**

Disables pull-ups on I/Os during configuration

**HyperTransport**

I/O standard selected via LDT option

## I

**IBIS**

I/O Buffer Information Specification

**IC**

Integrated Circuit

**ICR**

In-Circuit Reconfigurable

**ILA**

Integrated Logic Analyzer. See “ChipScope”

**iMPACT**

Xilinx programming tool

**industrial**

Junction temperature operating range -40-100C

**INIT**

1. INITialization pin

2. INITial state control

**I/O**

Input/Ouput

**IOB**

Input/Output Block

**IP**

1. Intellectual Property, usually referring to predefined core designs
2. Internet Protocol

**ISE**

Integrated Software Environment; Xilinx development system

**ISP**

1. In-System Programmable
2. Internet Service Provider

## J

**jitter**

In a periodic signal, the delay between the expected and actual transition

**JTAG**

Joint Test Action Group, common name for IEEE 1149.1 boundary scan standard

## L

**LDT**

Lightning Data Transport, previous name to HyperTransport

**lock**

When DLL has brought clock output in phase with clock intput

**logic cell**

Equivalent to a 4-input LUT and a flip-flop, Spartan-3 CLBs have additional logic making them equivalent to nine logic cells

**LogiCORE**

IP created and supported by Xilinx

**long line**

Routing that connects to one of every six CLBs

**LUT**

Look-Up Table, also known as function generator

**LVCMOS**

Low Voltage CMOS I/O standard

**LVDS**

Low Voltage Differential Signaling I/O standard

**LVDSEXT**

LVDS EXTended I/O standard

**LVTTL**

Low Voltage TTL I/O standard

## M

**map**

Translate a logic design to a physical design

**master**

Configuration mode where the FPGA supplies the configuration clock

**MicroBlaze**

32-bit soft RISC processor core

**module**

1. Block of a design
2. Section of a data sheet

**MultiPRO**

High-speed desktop download tool that programs all Xilinx devices

**mux**

Multiplexer

## N

**NC**

No Connect

**NCD**

Native Circuit Description, physical FPGA file

**no change**

Block RAM mode where data is written and outputs maintain current state

**NRE**

Non-Recurring Engineering or mask charges associated with gate arrays, ASICs, and SOCs. NRE charges do not apply to FPGAs.

## P

**PACE**

Pinout and Area Constraints Editor

**PAR**

Place And Route

**PCI**

Peripheral Component Interconnect local bus standard

**PicoBlaze**

Fuly embedded 8-bit RISC controller optimized for Xilinx FPGAs

**Platform Flash**

Cost-effective, in-system reprogrammable Flash for storing FPGA configuration bitstreams

**PLD**

Programmable Logic Device, generic name for FPGAs and CPLDs

**POR**

Power-On Reset

**PQ or PQFP**

Plastic quad flat pack package

**preliminary specification**

Characteristics not expected to change, based on characterization

**primitive**

Design library element

**production specification**

Final characteristics

**PROG**

PROGram pin

**Project Navigator**

Xilinx user interface to manage the design process

**PROMgen**

PROM file generator

**PS**

Phase Shifter part of DCM

## R

**RAM**

Writeable Random Access Memory

**readback**

Reading configuration data out of an FPGA

**read first**

Block RAM mode in which data is written and previous data is sent to outputs

**register**

A set of flip-flops

**RISC**

Reduced Instruction Set Computer

**ROM**

Read-Only Memory

**RPM**

Relationally Placed Macro

**RSDS**

Reduced Swing Differential Signaling I/O standard

**RTL**

Register Transfer Level or Language, depending on context

**RTOS**

Real-Time Operating System

## S

**schematic**

Graphical representation of a design

**SDR**

1. Single Date Rate

2. Software Defined Radio

**SelectIO**

I/O with programmable functionality and interface standards

**SelectMAP**

Port used in Master and Slave Parallel configuration modes

**SelectRAM**

Hierarchical memory system including distributed RAM and block RAM

**single ended**

Signaling scheme using one signal to transmit data, such as LVCMOS

**single port**

One data port allowing access to a memory

**skew**

Difference in timing

**slave**

Configuration mode with external clock source

**slew rate**

I/O transition speed

**slice**

One-fourth of a CLB, contains two LUTs

**SLICEL**

Slice for Logic only, right-hand LUTs in CLB

**SLICEM**

Slice for Memory or logic, left-hand LUTs in CLB

**SMT**

Surface Mount Technology

**SOC**

System On a Chip. Typically, a single device that integrates a processor, memory, and logic

**Spartan-3**

Family of low-cost Xilinx FPGAs ranging from the XC3S50 to the XC3S5000

**Spartan Series**

All families of low-cost Xilinx FPGAs: Spartan, Spartan-XL, Spartan-II, Spartan-IIE, and Spartan-3

**SPICE**

Transistor-level circuit model

**SPROM**

Serial PROM

**SR**

Set/Reset

**SRL**

Shift Register LUT

**SSTL**

Stub Series Terminated Logic I/O standard

**staggered pads**

Dual rows of I/O pads

**startup**

Device transition from configuration to operation

**synthesis**

Translation from behavioral HDL to structural netlist

**system gate**

Equivalent 2-input function required to implement an FPGA resource

### System Generator

Software tool for designing, simulating, and implementing FPGA-based DSP designs

## T

### TBUF

Three-state BUFfer

### Three-state

Includes high, low, and disabled states, also known as 3-state

### TJ

Junction Temperature

### TQ or TQFP

Thin quad flat pack package

### TRACE

Timing Reporter And Circuit Evaluator

### True Dual-Port RAM

RAM with either of two ports configurable as input or output

## U

### UCF

User Constraints File

### ULVDS

Ultra LVDS

## V

### VCCAUX

VCC AUXiliary

### VCCINT

VCC INTernal

### VCCO

VCC Output/input

### Verilog

A high-level design language

### VHDL

VHSIC HDL high-level design language

### VQ or VQFP

Very thin quad flat pack package

**VREF**

Voltage REFerence

**VRN**

Voltage Reference Negative for DCI

**VRP**

Voltage Reference Positive for DCI

**VTT**

Board termination voltage

## W

**WebPACK**

Xilinx free downloadable ISE development system

**write first**

Block RAM mode in which data is written and sent to outputs

## X

**XAPP**

Xilinx Application Note

**XCITE**

Xilinx Controlled Impedance TEchnology, also known as DCI

**XMD**

Xilinx Microprocessor Debug

**XPower**

Xilinx Power estimator tool

**XPS**

Xilinx Platform Studio

**XST**

Xilinx Synthesis Technology

**XtremeDSP**

Xilinx DSP solution including silicon, cores, development tools, and support


SPARTAN™-3
FPGA
MAKE IT YOUR ASIC

# *Index & Sales Office Listing*


SPARTAN™-3
FPGA
MAKE IT YOUR ASIC

# Index

## S



## T



## V



## W



## X

# Xilinx Sales Offices

## Headquarters

2100 Logic Drive
San Jose, CA 95124

Tel: (408) 559-7778
Fax: (408) 559-7114
TWX: (510) 600-8750
Web: http://www.xilinx.com

## North America

***Phoenix, AZ***
Tel: (480) 753-4503
Fax: (480) 753-4504

***Irvine, CA***
Tel: (949) 471-1300
Fax: (949) 727-3128

***San Diego, CA***
Tel: (858) 558-5974
Fax: (858) 558-6418

***Sunnyvale, CA***
Tel: (408) 470-0000
Fax: (408) 245-9865

***Greenwood Village, CO***
Tel: (303) 220-7541
Fax: (303) 220-8641

***Winter Park, FL***
Tel: (407) 673-8661
Fax: (407) 673-8663

***Schaumburg, IL***
Tel: (847) 605-1972
Fax: (847) 605-1976

***Deephaven, MN***
Tel: (952) 473-4816
Fax: (952) 473-5060

***Marriottsville, MD***
Tel: (410) 442-9748
Fax: (410) 442-9749

***Nashua, NH***
Tel: (603) 521-1200
Fax: (603) 891-0890

***Fairfield, NJ***
Tel: (973) 808-2780
Fax: (973) 808-2738

***Raleigh, NC***
Tel: (919) 455-2980
Fax: (919) 846-8316

***Brecksville, OH***
Tel: (440) 526-1991
Fax: (440) 526-5426

***Portland, OR***
Tel: (503) 293-9016
Fax: (503) 293-3858

***Dallas, TX***
Tel: (972) 246-0220
Fax: (972) 960-0927

***Salt Lake City, UT***
Tel: (801) 281-2245
Fax: (801) 281-2369

***Bellevue, WA***
Tel: (425) 451-7000
Fax: (425) 990-8989

***Oakville, Ontario Canada***
Tel: (905) 337-5850
Fax: (905) 337-3554

## European Headquarters

Benchmark House, 203
Brooklands Rd.
Weybridge Surrey KT13 0RH
United Kingdom

Tel: +44-870-7350-600
Fax: +44-870-7350-601

***Benelux***
Tel : +32-53-848310
Fax: +32-53-848311

***France and Spain***
Tel: +33-1-34-63-01-01
Fax: +33-1-34-63-01-09

***Germany, Switzerland, and Austria***
Tel: +49-89-93088-0
Fax: +49-89-93088-188

***Israel***
Tel: +972-3-9295-318
Fax: +972-3-9295-319

***Italy***
Tel: +39-02-487-12-101
Fax: +39-02-400-94-700

***Sweden, Norway, Denmark, and Finland***
Tel: +46-8-594-61-660
Fax: +46-8-594-61-661
e-mail: xilinx-nordic@xilinx.com

## Japan

Tel: +81-3-5321-7711
Fax: +81-3-5321-7765
Web: http://www.xilinx.co.jp

## Asia Pacific Headquarters

Hong Kong

Tel: +852-2-424-5200
Fax: +852-2-494-7159
e-mail: hongkong@xilinx.com

***Korea***
Tel : +822-761-4277
Fax : +822-761-4278
e-mail: ask-korea@xilinx.com

***Shanghai***
Tel: +86-21-6886-2323, 2322
Fax: +86-21-6886-2333
e-mail: ask-china@xilinx.com
Web: http://www.xilinx-china.com

***Shenzhen***
Tel: +86-755-2588-2622
Fax: +86-755-2583-0986
e-mail: ask-china@xilinx.com
Web: http://www.xilinx-china.com

***Taiwan***
Tel: +886-2-2739-6765
Fax: +886-2-2739-8423
e-mail: ask-taiwan@xilinx.com

For updates and addresses, see http://www.xilinx.com/company/sales/offices.htm

For information on Xilinx North American Sales Representative offices, see http://www.xilinx.com/company/sales/na_reps.htm

For information on Xilinx International Sales Representative offices, see http://www.xilinx.com/company/sales/int_reps.htm