Virtex-II Platform FPGA User Guide

UG002 (v2.0) 23 March 2005

XILINX®

Virtex-II™ Platform FPGA User Guide
UG002 (v2.0) 23 March 2005

The following table summarizes changes made to each version of this document. Publication versions are electronic only (PDF) unless otherwise stated.

Date	Version	Revision
12/06/00	1.0	(Printed version with data sheet modules 1-3 in Part I) Initial Release.
04/02/01	1.1	Misc updates throughout.
10/12/01	1.2	Misc technical edits throughout.
12/03/01	1.3	(Printed version with data sheet modules 1-3 in Part I) Misc updates throughout.
11/01/02	1.4	Misc updates throughout.
12/02/02	1.5	Misc updates throughout.
04/21/03	1.6	• Added HDL code for resetting the DCM after configuration in section "External Feedback" on page 84. • Corrected Location Constraints syntax, multiple instances. • Added LVDS_25 and LVDSEXT_25 to list of I/O standards that support DCI in section "DCI in Virtex-II Hardware" on page 220. • Corrected and updated section "Using LVPECL I/O" on page 247. • Corrected command-line equivalent statements for bitstream encryption key files in section "Creating Keys" on page 251. • Added new section "Abort" on page 268 to section SelectMAP Programming Modes. • Added LVDS_25 and LVDSEXT_25 to list of I/O standards that support DCI in section "DCI in Virtex-II Hardware" on page 220. • Revised description of the JTAG output in section "Test Access Port" on page 298. • Updated configuration parameters in Table 4-12 on page 309.
08/01/03	1.6.1	Table 4-3 on page 276: Corrected bias voltage on dedicated pins from 2.5V to 3.3V. Other minor edits to this table and the text above it.
02/04/04	1.7	• Figure 3-61 on page 146: Added missing connection between second LUT of Slice 3 and first LUT of Slice 2. • Figure 3-73 on page 168: Corrected bit numbers on inputs to adder. • Added section "Routing with BlockRAM" on page 171. • Table 3-29 on page 175: Corrected second grouping of HSTL standards to HSTL_18. • Section "DCI I/O Buffer Library Components" on page 216, added IBUFG_LVDS components to list. • Section "Location Constraints" on page 235: Added constraint on locating a DDR register next to an SDR register using a different clock. • Table 3-60 on page 238: Changed names of primitives in first row of table to IBUF_LVDS, OBUF_LVDS, etc. Underneath table, deleted text referring to primitives used in Virtex-E and earlier designs. • Table 4-2 on page 266: Updated all configurationbitstream lengths. • Table 4-3 on page 276: Corrected directionality of BUSY/DOUT to Output, INIT_B to Input/Output.

Date	Version	Revision
02/04/04 (cont'd)	1.7 (cont'd)	• The following parameter specification tables were removed from Chapter 4, Configuration (formerly Chapter 3): - *Table 3-3*, Power-Up Timing Configuration Signals - *Table 3-7*, Master/Slave Serial Mode Programming Switching - *Table 3-8*, SelectMAP Write Timing Characteristics - *Table 3-12*, Boundary-Scan Port Timing Specifications See the **Virtex-II Platform FPGA Data Sheet, DC and Switching Characteristics** for these and all other parameter specifications. • Section "Mixed Voltage Environments" on page 277: Restructured with new headings. Reworded Footnote (1) in Figure 4-4. • Added footnote to Figure 4-8 on page 283 and Figure 4-10 on page 285 clarifying that DOUT transitions on the falling edge of CCLK. • Second paragraph below Figure 4-12 on page 289: Corrected maximum no-handshake SelectMAP configuration speed from 5 MHz to 50 MHz. • Section "Master SelectMAP Data Loading" on page 289: Changed wording to emphasize that if RDWR_B is toggled while CS_B is still asserted, a configuration abort will occur. • Section "Express-Style Loading" on page 286: Added new text to clarify the sequencing of signals before, during, and after data loading. • Section "Test Access Port" on page 298: Added mention of implementation tool pull-up, pull-down, and float options on TMS and TDI pins. • Table 4-12 on page 309: Updated XC2V8000 configuration bitstream length. • Table 5-3 on page 421: Changed Theta-JC to 0.5°C/Watt for all FF packages.
04/16/04	1.8	• End of Table 3-37 on page 191, added reference to XAPP689 for detailed ground bounce discussion. • Section "DCI I/O Buffer Library Components" on page 216, deleted IBUF_LVDS_33_DCI, IBUFG_LVDS_33_DCI, IBUF_LVDSEXT_33_DCI, and IBUFG_LVDSEXT_33_DCI. • Section "DCI in Virtex-II Hardware" on page 220, deleted LVDS_33_DCI and LVDSEXT_33_DCI. • Section "Using Bitstream Encryption" on page 250, added reference to Appendix C, Choosing the Battery for V_{BATT}. • End of section "Creating Keys" on page 251, added paragraph advising use of a different CBC initial value for each design to insure security. • Table 4-3 on page 276, added instruction to connect V_{BATT} to V_{CCAUX} or GND when bitstream encryption is not used. • Section "Frame Length Register (FLR)" on page 310, corrected definition of the value loaded into this register by adding "minus one word."

Date	Version	Revision
08/05/04	1.9	• Revised throughout with updated tables, text, and package diagrams to include new Pb-free packaging options CSG144, FGG256, FGG456, FGG676, BGG575, and BGG728. • Section "Daisy-Chain Configuration" on page 281: Revised text to clarify daisy-chain options for strings of different Xilinx FPGA devices. • Section "RDWR_B" on page 283: Revised text to clarify the status of the data bus when RDWR_B is High and Low. • Table 4-15 on page 307: Corrected Footnote (1) to say "data and instruction".
03/23/05	2.0	• **Chapter 4, Configuration**: This chapter has been expanded and completely updated with new material. Since most of the material is newly written, specific changes in this chapter are not recorded in the Revision History. • Section "BUFG Exclusivity" on page 67: Corrected numbering of exclusive buffer pairs. • Figure 3-23 on page 82: Added note on when to reset the DCM. • Table 3-9 on page 104: From "Description" for CLKIN_DIVIDE_BY_2, removed text specifying an upper frequency limit for using this attribute to divide the DCM input clock by 2. • Section "Using Embedded Multipliers" on page 167: Deleted subsection Two Multipliers in a Single Primitive. Deleted references to DUAL_MULT_* templates. • Table 3-29 on page 175: Added Footnote (1) for GTL and GTL+ standards. Corrected voltage parameters for SSTL18_I and SSTL18_II standards, and added Footnote (2). • Section "Overview of Supported I/O Standards" on page 175: Added section SSTL18 - Stub Series Terminated Logic for 1.8V. • Table 3-30, Table 3-32, Table 3-34, and Table 3-35: Added symbols for SSTL18_I and SSTL18_2. • Section "Output Drive Source Voltage (V_{CCO}) Pins" on page 186: Added text recommending that V_{CCO} be powered for all I/O standards. • Table 3-36: Added entries for SSTL18_I, SSTL18_II, SSTL18_I_DCI, and SSTL18_II_DCI. Corrected parameters for HSTL_II_DCI, HSTL_I_18, HSTL_II_18, HSTL_III_18, HSTL_IV_18, HSTL_I_DCI_18, HSTL_II_DCI_18, HSTL_III_DCI_18, and HSTL_IV_DCI_18. • Section "DCI in Virtex-II Hardware" on page 220: Added steps 7 and 8 to list. • Added Figure 3-111 on page 226. • Code template DDR_input.v: Removed `assign data_out = q1 & q2` just before `endmodule`. • Section "Using LVDS I/O" on page 241: Removed table Available Virtex-II LVDS Primitives and associated text. • Section "Loading Encrypted Bitstreams" on page 253: Corrected reference to JTAG JSTART instruction to JTAG JPROG_B. • Added new section "Temperature-Sensing Diode (DXP/DXN)" on page 253. • Section "Design Examples" on page 485 (Appendix C): Deleted text and illustration that incorrectly suggested different battery voltage compliance requirements for Virtex-II and Virtex-II Pro BBRAM power pin.

Contents

Chapter 1: Contents

Preface: About This Guide

Additional Resources 15

Chapter 2: Introduction to the Virtex-II FPGA Family

Virtex-II Platform 19
Virtex-II Target Applications 19
Interconnect Engine for Fast, Wide Busses in Networking Applications 20
Complete Solution For Rapid Time-to-Production 20

Chapter 3: Timing Models

Summary 21
Introduction 21
CLB / Slice Timing Model 22
Introduction 22
General Slice Timing Model and Parameters 22
Slice Distributed RAM Timing Model and Parameters 25
Slice SRL Timing Model and Parameters 28
Block SelectRAM Timing Model 30
Embedded Multiplier Timing Model 34
IOB Timing Model 36
IOB Input Timing Model and Parameters 37
IOB Output Timing Model and Parameters 41
IOB 3-State Timing Model and Parameters 44
Pin-to-Pin Timing Model 48
Introduction 48
Global Clock Input to Output 49
Global Clock Setup and Hold 51
Digital Clock Manager Timing Model 53
Operating Frequency Ranges 53
Input Clock Tolerances 54
Output Clock Precision 55
Miscellaneous DCM Timing Parameters 56

Chapter 4: Design Considerations

Summary 59
Introduction 59
Using Global Clock Networks 60
Introduction 60
Clock Distribution Resources 60
Power Consumption 70

Library Primitives and Submodules 71
Characteristics 74
Location Constraints 75
Secondary Clock Network 75
VHDL and Verilog Instantiation 75
Using Digital Clock Managers (DCMs) 80
Overview 80
Clock De-Skew 81
Frequency Synthesis 89
Phase Shifting 93
DCM Waveforms 109
Using Block SelectRAM™ Memory 112
Introduction 112
Synchronous Dual-Port and Single-Port RAM 112
Characteristics 116
Library Primitives 116
VHDL and Verilog Instantiation 118
Port Signals 118
Address Mapping 119
Attributes 120
Initialization in VHDL or Verilog Codes 121
Location Constraints 121
Applications 122
VHDL and Verilog Templates 122
Using Distributed SelectRAM Memory 129
Introduction 129
Characteristics 130
Library Primitives 131
VHDL and Verilog Instantiation 132
Ports Signals 132
Attributes 132
Initialization in VHDL or Verilog Codes 133
Location Constraints 133
Applications 135
VHDL and Verilog Templates 136
Using Look-Up Tables as Shift Registers (SRLs) 139
Introduction 139
Shift Register Operations 139
Characteristics 141
Library Primitives and Submodules 141
Initialization in VHDL and Verilog Code 144
Port Signals 144
Attributes 144
Location Constraints 145
Fully Synchronous Shift Registers 146
Static-Length Shift Registers 147
VHDL and Verilog Instantiation 148
Designing Large Multiplexers 150
Introduction 150
Virtex-II CLB Resources 150
Wide-Input Multiplexers 154

Characteristics 154
Library Primitives and Submodules 155
Port Signals 155
Applications 156
VHDL and Verilog Instantiation 156
Implementing Sum of Products (SOP) Logic 160
Introduction 160
Virtex-II CLB Resources 160
VHDL Parameters 161
Applications 161
VHDL and Verilog Instantiation 162
Using Embedded Multipliers 167
Introduction 167
Two's-Complement Signed Multiplier 167
Library Primitives and Submodules 167
VHDL and Verilog Instantiation 170
Port Signals 171
Location Constraints 171
Routing with BlockRAM 171
VHDL and Verilog Templates 172
Using Single-Ended SelectI/O-Ultra Resources 174
Summary 174
Introduction 174
Fundamentals 174
Overview of Supported I/O Standards 175
Library Symbols 177
Design Considerations 185
5V Tolerance in Virtex-II Devices 209
Using Digitally Controlled Impedance (DCI) 210
Introduction 210
Xilinx DCI 210
Software Support 216
DCI in Virtex-II Hardware 220
Using Double-Data-Rate (DDR) I/O 227
Introduction 227
Data Flow 227
Characteristics 232
Library Primitives 233
VHDL and Verilog Instantiation 233
Port Signals 234
Initialization in VHDL or Verilog 235
Location Constraints 235
Applications 235
VHDL and Verilog Templates 235
Using LVDS I/O 241
Introduction 241
Creating an LVDS Output Buffer 242
Creating an LVDS Output 3-State Buffer 244
Creating a Bidirectional LVDS Buffer 245
LDT 246

LDT Implementation 246
Using LVPECL I/O 247
Introduction 247
Creating an LVPECL Input/Clock Buffer 247
Creating an LVPECL Output Buffer 248
Using Bitstream Encryption 250
What DES Is 250
How Triple DES is Different 250
Classification and Export Considerations 251
Creating Keys 251
Loading Keys 253
Loading Encrypted Bitstreams 253
V_{BATT} 253
Temperature-Sensing Diode (DXP/DXN) 253
Using the CORE Generator System 255
Introduction 255
The CORE Generator System 255
CORE Generator Design Flow 256
Core Types 257
Xilinx IP Solutions and the IP Center 259
CORE Generator Summary 261
Virtex-II IP Cores Support 261

Chapter 5: Configuration

Summary 269
Introduction 269
Configuration Modes 270
Configuration Process and Flow 272
Configuration Pins 276
Mixed Voltage Environments 277
Configuration Solutions 278
Configuration PROMs 278
Flash PROMs With a CPLD Configuration Controller 279
Embedded Solutions 280
PROM Selection Guide 281
Software Support and Data Files 282
iMPACT Software 282
Programming Cables 282
Boundary Scan Interconnect Testing for Virtex-II Devices 282
In-System Programming Data Files 282
Serial Programming Modes 283
Master Serial Mode 283
Slave Serial Mode 284
SelectMAP Programming Modes 286
Master SelectMAP Mode 288
Slave SelectMAP Mode 289
SelectMAP ABORT Sequence and ABORT Recovery 294
Triggering an ABORT 294
ABORT Status Word 295

ABORT Recovery 296
Internal Configuration Access Port (ICAP) 297
JTAG / Boundary Scan Programming Mode 298
Introduction 298
Boundary-Scan for Virtex-II Devices Using IEEE Standard 1149.1 298
Using Boundary Scan in Virtex-II Devices 305
Boundary-Scan for Virtex-II Devices Using IEEE Standard 1532 310
Configuration Flows Using JTAG 312
Configuration Details 314
Configuration Memory: Columns and Frames 314
Configuration Memory Addressing 316
Bitstream Packets 317
Configuration Control Logic 318
Configuration 331
Partial Reconfiguration 336
Readback 342
Preparing a Design for Readback 342
Readback Command Sequences 343
Readback Files 353
Verifying Readback Data 353
Readback Capture 356
Using ChipScope ILA 357

Chapter 6: PCB Design Considerations

Summary 359
Pinout Information 360
Introduction 360
Pin Definitions 360
FG256 Fine-Pitch BGA Package 362
Pinout Diagrams 369
CS144 Chip-Scale BGA Composite Pinout Diagram (XC2V250) 370
FG256 Fine-Pitch BGA Composite Pinout Diagram (XC2V1000) 371
FG456 Fine-Pitch BGA Composite Pinout Diagram (XC2V1000) 375
FG676 Fine-Pitch BGA Composite Pinout Diagram (XC2V3000) 379
BG575 Standard BGA Composite Pinout Diagram (XC2V2000) 383
BG728 Standard BGA Composite Pinout Diagram (XC2V3000) 387
FF896 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V2000) 391
FF1152 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V8000) 395
FF1517 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V8000) 399
BF957 Flip-Chip BGA Composite Pinout Diagram (XC2V6000) 403
FG456 - FG676 Pinout Compatibility Diagram 406
FF896 - FF1152 Pinout Compatibility Diagram 407
Package Specifications 409
CS144 Chip-Scale BGA Package (0.80 mm Pitch) 410
FG256 Fine-Pitch BGA Package (1.00 mm Pitch) 411
FG456 Fine-Pitch BGA Package (1.00 mm Pitch) 412
FG676 Fine-Pitch BGA Package (1.00 mm Pitch) 413
BG575 Standard BGA Package (1.27 mm Pitch) 414
BG728 Standard BGA Package (1.27 mm Pitch) 415
FF896 Flip-Chip Fine-Pitch BGA Package (1.00 mm Pitch) 416

FF1152 Flip-Chip Fine-Pitch BGA Package (1.00 mm Pitch) 417
FF1517 Flip-Chip Fine-Pitch BGA Package (1.00 mm Pitch) 418
BF957 Flip-Chip BGA Package (1.27 mm Pitch) 419
Flip-Chip Packages 420
Advantages of Flip-Chip Technology 420
Thermal Data 421
Thermal Considerations 421
Thermal Management Options 422
Printed Circuit Board Considerations 423
Layout Considerations 423
V_{CC} Decoupling 423
Board Routability Guidelines 428
Board-Level BGA Routing Challenges 428
Board Routing Strategy 429
Power Consumption 450
CLB Logic Power 450
Block SelectRAM Power 453
Digital Clock Management Power 454
Non-Registered Multiplier Power 454
Registered Multiplier Power 455
Input/Output Power 455
Results 456
IBIS Models 458
Using IBIS Models 458
IBIS Generation 458
Advantages of IBIS 458
IBIS File Structure 459
IBIS I/V and dV/dt Curves 459
Ramp and dV/dt Curves 460
IBIS Simulations 460
IBIS Simulators 462
Xilinx IBIS Advantages 462
IBIS Reference Web Site 462
BSDL and Boundary Scan Models 463
BSDL Files 463

Appendix A: BitGen and PROMGen Switches and Options

Using BitGen 465
BitGen Syntax 466
BitGen Files 466
BitGen Options 467
Using PROMGen 473
PROMGen Syntax 474
PROMGen Files 474
PROMGen Options 475
Examples 477

Appendix B: Platform Flash Family PROMs

PROM Package Specifications 479
FS48 Package Specification 480
VO48 Package Specification 481
VO20 Package Specification 482

Appendix C: Choosing the Battery for V_{BATT}

Battery Types and Chemistry Choices 483
Primary or Secondary 483
Battery Cost and Care 484
Battery Summary 484
Design Examples 485
Case #1: 485
Case #2: 485
Case #3: 485
Case #4: 486

Glossary 487

Index 497

About This Guide

This document describes the function and operation of Virtex™-II devices and also includes information on FPGA configuration techniques and PCB design considerations. For Virtex-II device specifications, refer to the **Virtex-II Data Sheet (DS031)**.

This guide provides details on the following topics:

- Chapter 1: Introduction to the Virtex-II FPGA Family
- Chapter 2: Timing Models
- Chapter 3: Design Considerations
- Chapter 4: Configuration
- Chapter 5: PCB Design Considerations
- Appendix A: BitGen and PROMGen Switches and Options
- Appendix B: Platform Flash Family PROMs
- Appendix C: Choosing the Battery for V_{BATT}
- Glossary

Additional Resources

The following table lists URLs for resources available on the web. For additional information, go to **http://www.xilinx.com**.

Resource	Description/URL
Data Sheets	Xilinx data sheets describe device-specific operating characteristics, architecture, and pinouts/packaging. **http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp**
User Guides	Xilinx user guides contain detailed, device-specific operating theory and generic design examples for various device functions. **http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=User+Guides**
Application Notes	Xilinx application notes describe design techniques and approaches to general design as well as specific applications. Many application notes feature complete reference designs including source code. **http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Application+Notes**
Xcell Journals	This site contains quarterly journals for Xilinx programmable logic users. **http://www.xilinx.com/xcell/xcell.htm**

Resource	Description/URL
Tech Tips	See this site for the latest news, design tips, and patch information on the Xilinx design environment. http://www.xilinx.com/support/techsup/journals/index.htm
Answers Database	This database provides a current listing of solution records for Xilinx software tools. Search this database using the search function at: http://www.xilinx.com/support/searchtd.htm

Conventions

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

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *`design_name`*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values. (In angle brackets.)	**`ngdbuild`** *`<design_name>`*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** *`[option_name] design_name`*
Braces { }	A list of items from which you must choose one or more	**`lowpwr =`**{**`on`**|**`off`**}
Vertical bar |	Separates items in a list of choices	**`lowpwr =`**{**`on`**|**`off`**}

Convention	Meaning or Use	Example

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *`block_name loc1 loc2 ... locn;`*

Online Document

The following conventions are used in this document:

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

XILINX®

Chapter 1

Introduction to the Virtex-II FPGA Family

Virtex-II Platform

The Virtex-II Platform FPGA solution is the result of the largest silicon and software R&D effort in the history of programmable logic, with the goal of revolutionizing the design of complex single-chip sub-systems in terms of engineering productivity, silicon efficiency, and system flexibility.

The Virtex-II product family provides IP-Immersion™ technology which incorporates an abundance of on-chip memory options and advanced routing resources for supporting complex designs that use IP (intellectual property), such as on-chip hard-macro building blocks and a rapidly growing library of soft-IP blocks. For the first time in the programmable logic industry, innovative Virtex-II features enable system designers to:

- Eliminate external termination resistors with on-chip precision-controlled output impedance
- Manage 16 pre-engineered low-skew clock domains, with on-chip frequency and phase control
- Protect chip designs with bit-stream encryption

These unique capabilities increase engineering productivity and time-to-production by supply pre-engineered solutions for signal integrity and RF noise challenges, as well as providing a secure means to deliver designs rapidly to production.

The Virtex-II Platform FPGA family is a complete programmable solution that allows digital system designers to rapidly implement a single-chip solution with density up to 10 million system gates, in weeks rather than months or years. The inherent flexibility of Xilinx FPGA devices allows unlimited design changes throughout the development and production phases of the system, with important benefits in improved productivity, reduced design risk, and higher system flexibility. This further accelerates the industry -- from custom ASICs to FPGAs -- in fields such as optical networking systems, gigabit routers, wireless cellular base stations, modem arrays, and professional video broadcast systems.

Virtex-II Target Applications

The Virtex-II solution is developed specifically to enable rapid development of two of the most technically challenging digital system applications: data communications and digital signal processing (DSP) systems. High logic integration, fast and complex routing of wide busses, and extensive pipeline and FIFO memory requirements characterize these systems.

The Virtex-II family incorporates high logic capacity, up to 10 million system gates, a new Active Interconnect™ architecture optimized for predictable routing delays, an advanced

memory array architecture with up to 4.5Mbits of on-chip memory, and built-in support for high-speed I/O standards at up to 1108 user pins.

Applications incorporating DSP functionality, such as echo cancellation, forward error-correction, and image compression/decompression, benefit from the abundance of embedded high-speed 18-bit x 18-bit multiplier blocks within the Virtex-II solution.

The unique features of the revolutionary Virtex-II architecture make it ideal for optical networking products, storage area networks (SANs), Voice-over-Internet-Protocol (VoIP), video broadcasting, medical imaging, wireless base-stations, and Internet infrastructure products, as well as many other products.

Interconnect Engine for Fast, Wide Busses in Networking Applications

The Virtex-II architecture incorporates a number of novel features specifically to support wide data widths in complex networking and transmission systems. Modern complex systems operate with multiple clock domains, with large IP-based subsystems operating independently. Large, wide FIFOs and buffer memories are needed for handling fast and wide inter-subsystem data transfer. These wide busses are required both internally for intra-chip communications and externally for switched fabric communications.

For example, wide 32-bit and larger data busses can drive multiple Ultra Low-Voltage Differential Signal (ULVDS) high-speed interface standards for data transfer across a backplane or for point-to-point communications, or be used for implementing high-speed multi-cast bus standards.

These requirements challenge and exceed the capabilities of current programmable logic devices, which lack the gate capacity, memory and routing resources, performance, and architecture flexibility to fully support these designs. The Virtex-II solution is the first platform FPGA specifically targeted to improve the “ease of speed” in the development and production of these complex systems.

Complete Solution For Rapid Time-to-Production

The Virtex-II solution combines the most flexible FPGA architecture, advanced process technology, powerful software synthesis technology, and robust IP library, to provide the most complete system integration solution today. In addition, the Virtex-II solution provides powerful features, such as Xilinx Digitally Controlled Impedance (DCI) technology, digital clock manager to help designers further reduce overall system cost and design development cycle, making Virtex-II the ideal solution for tomorrow’s high-performance system designs.

Chapter 2

Timing Models

Summary

The following topics are covered in this chapter:

- CLB / Slice Timing Model
- Block SelectRAM Timing Model
- Embedded Multiplier Timing Model
- IOB Timing Model
- Pin-to-Pin Timing Model
- Digital Clock Manager Timing Model

Introduction

Due to the large size and complexity of Virtex-II FPGAs, understanding the timing associated with the various paths and functional elements has become a difficult and important problem. Although it is not necessary to understand the various timing parameters in order to implement most designs using Xilinx, Inc. software, a thorough timing model can assist advanced users in analyzing critical paths, or planning speed-sensitive designs.

The Timing Model chapter is broken up into five sections consisting of three basic components:

- Functional Element Diagram - basic architectural schematic illustrating pins and connections.
- Timing Parameters - **Virtex-II Data Sheet (DS031)** timing parameter definitions.
- Timing Diagram - illustrates functional element timing parameters relative to each other.

This chapter was written with the Xilinx Timing Analyzer software (TRCE) in mind. All pin names, parameter names, and paths are consistent with Post Route Timing and Pre-Route Static Timing reports. Use the models in this chapter in conjunction with both the Timing Analyzer software and the section on switching characteristics in the **Virtex-II Data Sheet (DS031)**. Most of the timing parameters found in the section on switching characteristics are described in this chapter.

CLB / Slice Timing Model

Introduction

The three sections below describes all timing parameters reported in the **Virtex-II Data Sheet** that are associated with slices and Configurable Logic Blocks (CLBs). The sections correspond to their respective (switching characteristics) sections in the data sheet:

- General Slice Timing Model and Parameters (CLB Switching Characteristics)
- Slice Distributed RAM Timing Model and Parameters (CLB Distributed RAM Switching Characteristics)
- Slice SRL Timing Model and Parameters (CLB SRL Switching Characteristics)

General Slice Timing Model and Parameters

Figure 2-1 illustrates the details of a Virtex-II slice.

Note: Some elements of the Virtex-II slice have been omitted for clarity. Only the elements relevant to the timing paths described in this section are shown.

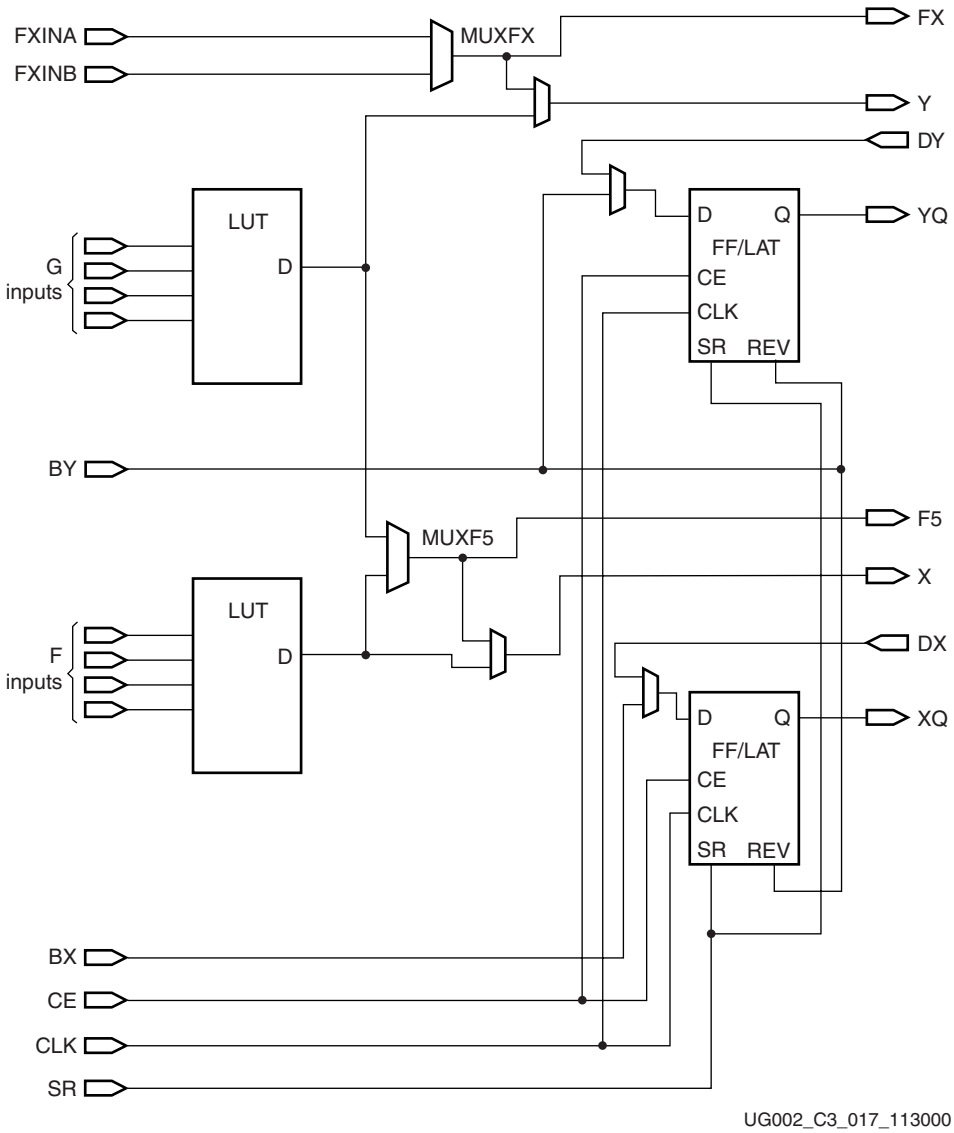


Figure 2-1: **General Slice Diagram**

Timing Parameters

Table 2-1: **General Slice Tming Parameters**

Parameter	Function	Control Signal	Description
Combinatorial Delays			
T_{ILO}	F/G inputs to X/Y outputs		Propagation delay from the F/G inputs of the slice, through the look-up tables (LUTs), to the X/Y outputs of the slice.
T_{IF5}	F/G inputs to F5 output		Propagation delay from the F/G inputs of the slice, through the LUTs and MUXF5 to the F5 output of the slice.
T_{IF5X}	F/G inputs to X output		Propagation delay from the F/G inputs of the slice, through the LUTs and MUXF5 to the X output of the slice.
T_{IFXY}	FXINA/FXINB inputs to Y output		Propagation delay from the FXINA/FXINB inputs, through MUXFX to the Y output of the slice.
T_{IFNCTL}	Transparent Latch input to XQ/YQ outputs		Incremental delay through a transparent latch to XQ/YQ outputs.
Sequential Delays			
T_{CKO}	FF Clock (CLK) to XQ/YQ outputs		Time after the clock that data is stable at the XQ/YQ outputs of the slice sequential elements (configured as a flip-flop).
T_{CKLO}	Latch Clock (CLK) to XQ/YQ outputs		Time after the clock that data is stable at the XQ/YQ outputs of the slice sequential elements (configured as a latch).
Setup and Hold for Slice Sequential Elements			
T_{xxCK} = Setup time (before clock edge) T_{CKxx} = Hold time (after clock edge)			The following descriptions are for setup times only.
T_{DICK}/T_{CKDI}	BX/BY inputs		Time before Clock (CLK) that data from the BX or BY inputs of the slice must be stable at the D-input of the slice sequential elements (configured as a flip-flop).
T_{DYCK}/T_{CKDY}	DY input		Time before Clock (CLK) that data from the DY input of the slice must be stable at the D-input of the slice sequential elements (configured as a flip-flop).
T_{DXCK}/T_{CKDX}	DX input		Time before Clock (CLK) that data from the DX input of the slice must be stable at the D-input of the slice sequential elements (configured as a flip-flop).
T_{CECK}/T_{CKCE}	CE input		Time before Clock (CLK) that the CE (Clock Enable) input of the slice must be stable at the CE-input of the slice sequen-tial elements (configured as a flip-flop).
T_{RCK}/T_{CKR}	SR/BY inputs		Time before CLK that the SR (Set/Reset) and the BY (Rev) inputs of the slice must be stable at the SR/Rev-inputs of the slice sequential elements (configured as a flip-flop). Synchronous set/reset only.

Table 2-1: **General Slice Tming Parameters *(Continued)***

Parameter	Function	Control Signal	Description
Clock CLK			
T_{CH}			Minimum Pulse Width, High.
T_{CL}			Minimum Pulse Width, Low.
Set/Reset			
T_{RPW}			Minimum Pulse Width for the SR (Set/Reset) and BY (Rev) pins.
T_{RQ}			Propagation delay for an asynchronous Set/Reset of the slice sequential elements. From SR/BY inputs to XQ/YQ outputs.
F_{TOG}			Toggle Frequency - Maximum Frequency that a CLB flip-flop can be clocked: $1/(T_{CH}+T_{CL})$

Timing Characteristics

Figure 2-2 illustrates general timing characteristics of a Virtex-II slice.

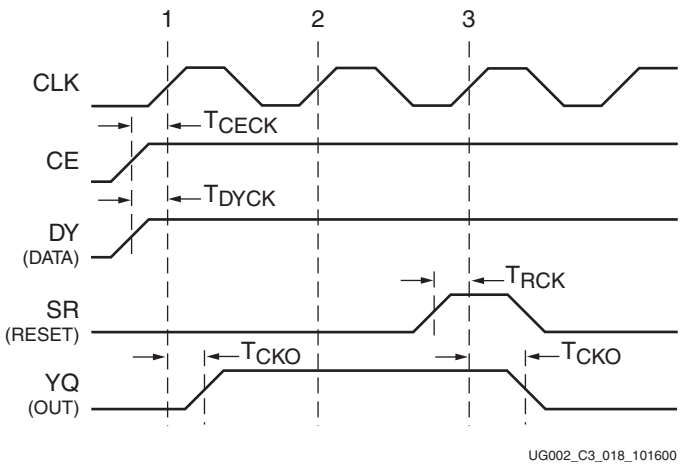


Figure 2-2: **General Slice Timing Diagram**

- At time T_{CECK} before Clock Event 1, the Clock-Enable signal becomes valid-high at the CE input of the slice register.
- At time T_{DYCK} before Clock Event 1, data from the DY input becomes valid-high at the D input of the slice register and is reflected on the YQ pin at time T_{CKO} after Clock Event 1*.
- At time T_{RCK} before Clock Event 3, the SR signal (configured as synchronous reset in this case) becomes valid-high, resetting the slice register, and this is reflected on the YQ pin at time T_{CKO} after Clock Event 3.

  * NOTE: In most cases software uses the DX/DY inputs to route data to the slice registers when at all possible. This is the fastest path to the slice registers and saves other slice routing resources.

Slice Distributed RAM Timing Model and Parameters

Figure 2-3 illustrates the details of distributed RAM implemented in a Virtex-II slice.

Note: Some elements of the Virtex-II slice have been omitted for clarity. Only the elements relevant to the timing paths described in this section are shown.

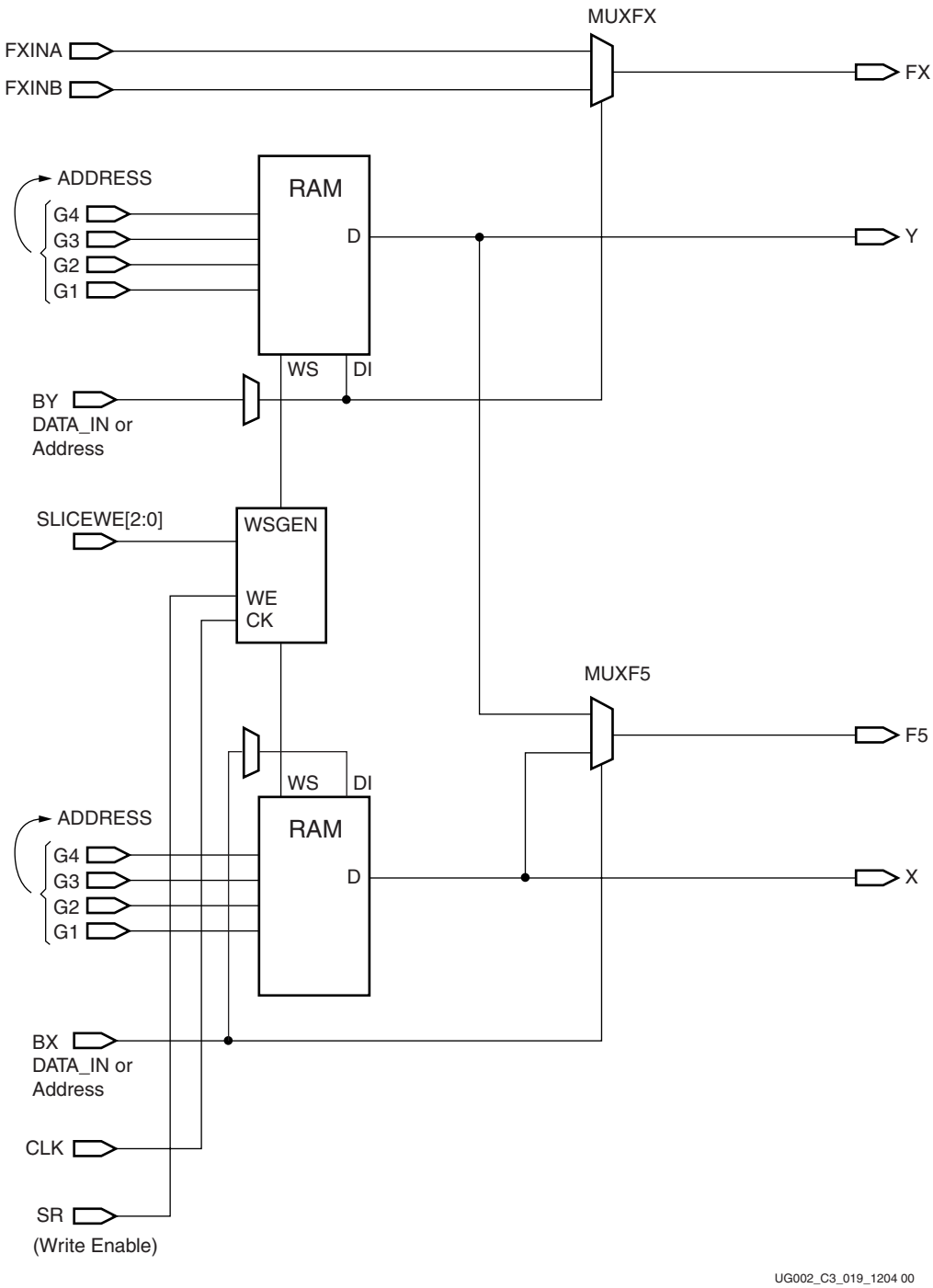


Figure 2-3: **Slice Distributed RAM Diagram**

Timing Parameters

Table 2-2: **Slice Distributed RAM Timing Parameters**

Parameter	Function	Control Signal	Description
Sequential Delays for Slice LUT Configured as RAM (Distributed RAM)			
$T_{SHCKO16}$	CLK to X/Y outputs (WE active) in 16x1 mode		Time after the Clock (CLK) of a WRITE operation that the data written to the distributed RAM (in 16x1 mode) is stable on the X/Y outputs of the slice.
$T_{SHCKO32}$	CLK to X/Y outputs (WE active) in 32x1 mode		Time after the Clock (CLK) of a WRITE operation that the data written to the distributed RAM (in 32x1 mode) is stable on the X/Y outputs of the slice.
$T_{SHCKOF5}$	CLK to F5 output (WE active)		Time after the Clock (CLK) of a WRITE operation that the data written to the distributed RAM is stable on the F5 output of the slice.
Setup and Hold for Slice LUT Configured as RAM (Distributed RAM)			
T_{xS} = Setup time (before clock edge) T_{xH} = Hold time (after clock edge)			The following descriptions are for setup times only.
T_{DS}/T_{DH}	BX/BY Data inputs (DI)		Time before the clock that data must be stable at the DI input of the slice LUT (configured as RAM), via the slice BX/BY inputs.
T_{AS}/T_{AH}	F/G Address inputs		Time before the clock that address signals must be stable at the F/G inputs of the slice LUT (configured as RAM).
T_{WES}/T_{WEH}	WE input (SR)		Time before the clock that the Write Enable signal must be stable at the WE input of the slice LUT (configured as RAM).
Clock CLK			
T_{WPH}			Minimum Pulse Width, High (for a Distributed RAM clock).
T_{WPL}			Minimum Pulse Width, Low (for a Distributed RAM clock).
T_{WC}			Minimum clock period to meet address write cycle time.

Timing Characteristics

Figure 2-4 illustrates the timing characteristics of a 16-bit distributed RAM implemented in a Virtex-II slice (LUT configured as RAM).

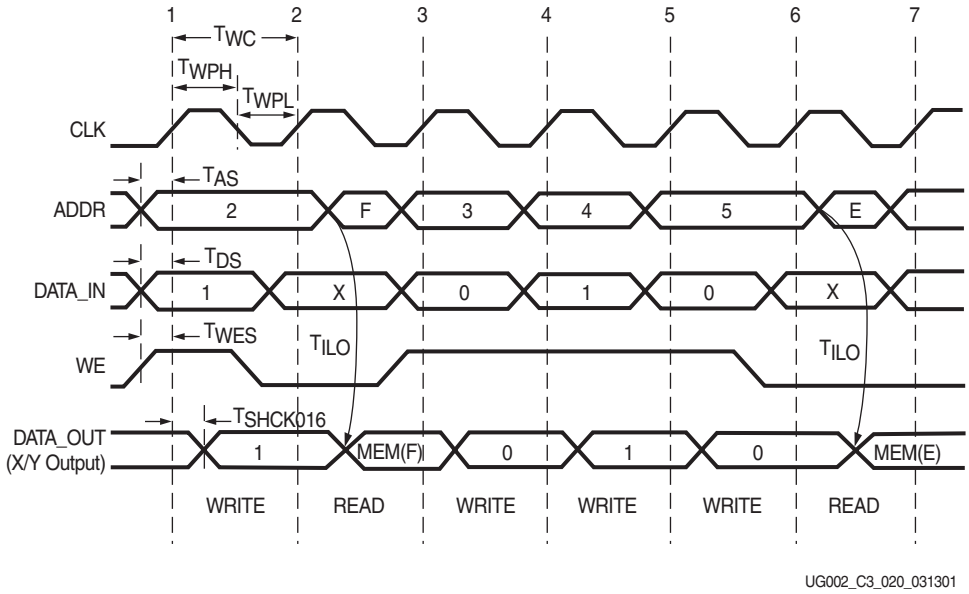


Figure 2-4: **Slice Distributed RAM Timing Diagram**

Clock Event 1: WRITE Operation

During a WRITE operation, the contents of the memory at the address on the ADDR inputs is changed. The data written to this memory location is reflected on the X/Y outputs synchronously.

- At time T_{WES} before Clock Event 1, the Write Enable signal (WE) becomes valid-high, enabling the RAM for the following WRITE operation.
- At time T_{AS} before Clock Event 1, the address (2) becomes valid at the F/G inputs of the RAM.
- At time T_{DS} before Clock Event 1, the DATA becomes valid (1) at the DI input of the RAM and is reflected on the X/Y output at time $T_{SHCKO16}$ after Clock Event 1.

Clock Event 2: READ Operation

All READ operations are asynchronous in distributed RAM. As long as write-enable (WE) is Low, the address bus can be asserted at any time, and the contents of the RAM at that address are reflected on the X/Y outputs after a delay of length T_{ILO} (propagation delay through a LUT). Note that the Address (F) is asserted *after* Clock Event 2, and that the contents of the RAM at that location are reflected on the output after a delay of length T_{ILO}.

Slice SRL Timing Model and Parameters

Figure 2-5 illustrates shift register implementation in a Virtex-II slice.

Note: Some elements of the Virtex-II slice have been omitted for clarity. Only the elements relevant to the timing paths described in this section are shown.

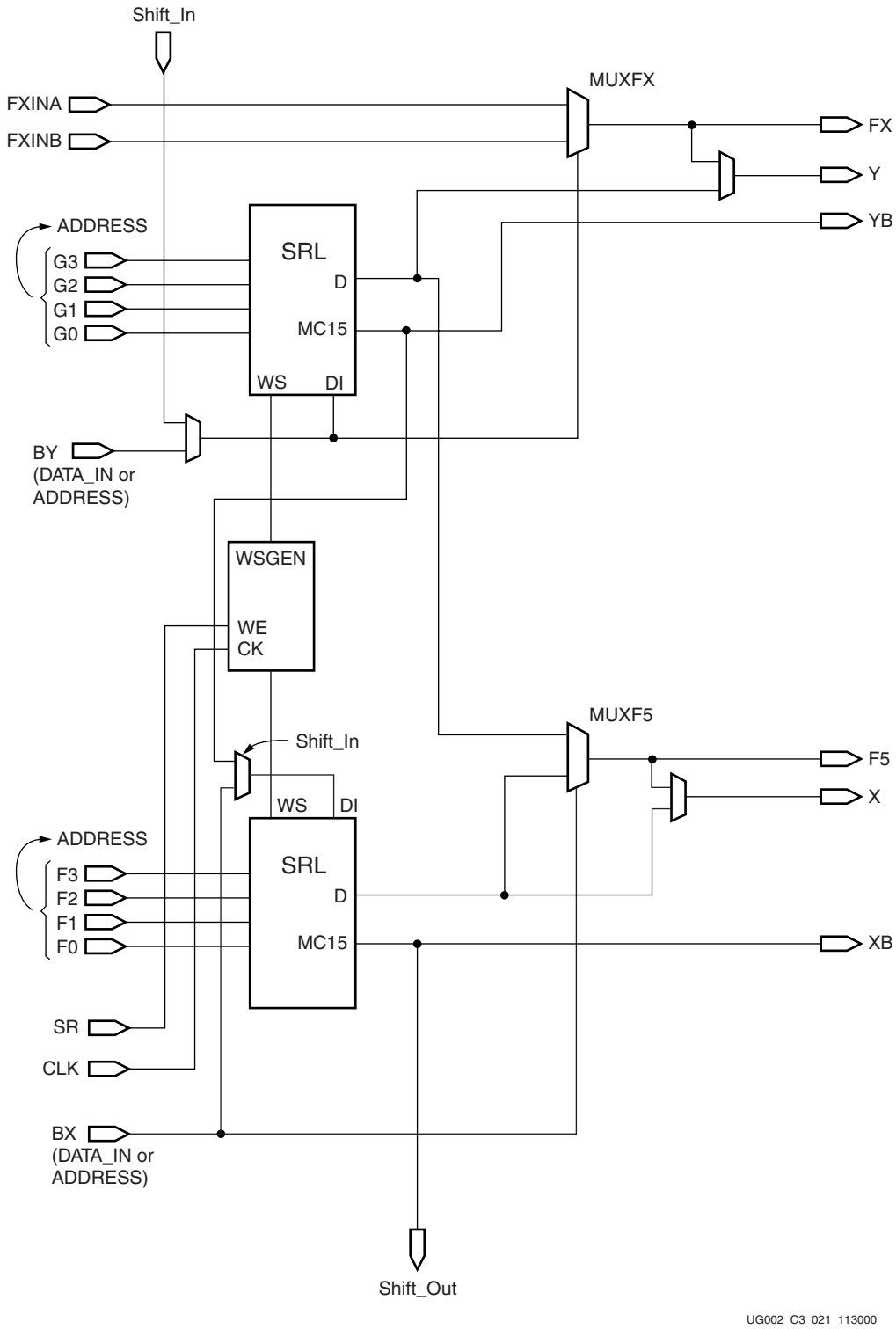


Figure 2-5: **Slice SLR Diagram**

Timing Parameters

Table 2-3: **Slice SRL Timing Parameters**

Parameter	Function	Control Signal	Description
Sequential Delays for Slice LUT Configured as SRL (Select Shift Register)			
T_{REG}	CLK to X/Y outputs		Time after the Clock (CLK) of a WRITE operation that the data written to the SRL is stable on the X/Y outputs of the slice.
T_{CKSH}	CLK to Shiftout		Time after the Clock (CLK) of a WRITE operation that the data written to the SRL is stable on the Shiftout or XB/YB outputs of the slice.
T_{REGF5}	CLK to F5 output		Time after the Clock (CLK) of a WRITE operation that the data written to the SRL is stable on the F5 output of the slice.
Setup/Hold for Slice LUT Configured as SRL (Select Shift Register)			
T_{xxS} = Setup time (before clock edge) T_{xxH}= Hold time (after clock edge)			The following descriptions are for setup times only.
T_{SRLDS}/ T_{SRLDH}	BX/BY Data inputs (DI)		Time before the clock that data must be stable at the DI input of the slice LUT (configured as SRL), via the slice BX/BY inputs.
T_{WSS}/T_{WSH}	CE input (WE)		Time before the clock that the Write Enable signal must be stable at the WE input of the slice LUT (configured as SRL).
Clock CLK			
T_{SRPH}			Minimum Pulse Width, High (for an SRL clock).
T_{SRPL}			Minimum Pulse Width, Low (for an SRL clock).

Timing Characteristics

Figure 2-6 illustrates the timing characteristics of a 16-bit shift register implemented in a Virtex-II slice (LUT configured as SRL).

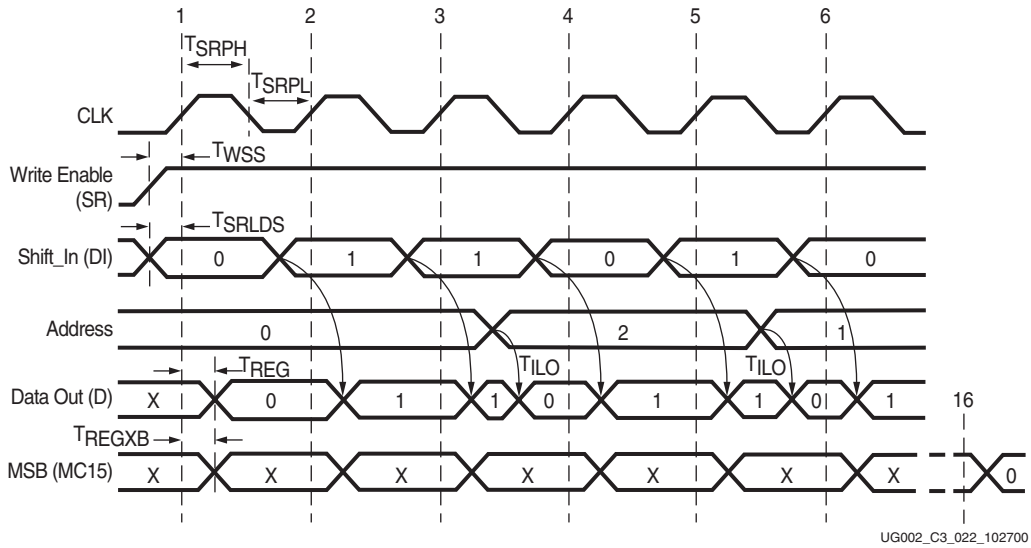


Figure 2-6: **Slice SLR Timing Diagram**

Clock Event 1: Shift_In

During a WRITE (Shift_In) operation, the single-bit content of the register at the address on the ADDR inputs is changed, as data is shifted through the SRL. The data written to this register is reflected on the X/Y outputs synchronously, if the address is unchanged during the clock event. If the ADDR inputs are changed during a clock event, the value of the data at the addressable output (D) is invalid.

- At time T_{WSS} before Clock Event 1, the Write Enable signal (SR) becomes valid-high, enabling the SRL for the WRITE operation that follows.
- At time T_{SRLDS} before Clock Event 1 the data becomes valid (0) at the DI input of the SRL and is reflected on the X/Y output after a delay of length T_{REG} after Clock Event 1*.

  * Note: Since the address 0 is specified at Clock Event 1, the data on the DI input is reflected at the D output, because it is written to Register 0.

Clock Event 2: Shift_In

- At time T_{SRLDS} before Clock Event 2, the data becomes valid (1) at the DI input of the SRL and is reflected on the X/Y output after a delay of length T_{REG} after Clock Event 2*.

  * Note: Since the address 0 is still specified at Clock Event 2, the data on the DI input is reflected at the D output, because it is written to Register 0.

Clock Event 3: Shift_In / Addressable (Asynchronous) READ

All READ operations are asynchronous. If the address is changed (between clock events), the contents of the register at that address are reflected at the addressable output (X/Y outputs) after a delay of length T_{ILO} (propagation delay through a LUT).

- At time T_{SRLDS} before Clock Event 3 the Data becomes valid (1) at the DI input of the SRL, and is reflected on the X/Y output T_{REG} time after Clock Event 3.
- Notice that the address is changed (from 0 to 2) some time after Clock Event 3. The value stored in Register 2 at this time is a 0 (in this example, this was the first data shifted in), and it is reflected on the X/Y output after a delay of length T_{ILO}.

Clock Event 16: MSB (Most Significant Bit) Changes

- At time T_{REGXB} after Clock Event 16, the first bit shifted into the SRL becomes valid (logical 0 in this case) on the XB output of the slice via the MC15 output of the LUT (SRL).

Block SelectRAM Timing Model

Introduction

This section describes the timing parameters associated with the block SelectRAM (illustrated in Figure 2-7) in Virtex-II FPGA devices. This section is intended to be used with the section on switching characteristics in the **Virtex-II Data Sheet (DS031)** and the Timing Analyzer (TRCE) report from Xilinx software. For specific timing parameter values, refer to the switching characteristics section in the *Virtex-II Data Sheet*.

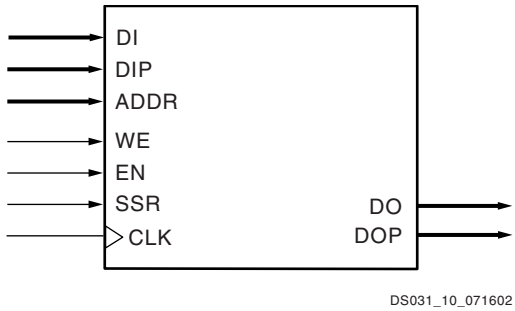


Figure 2-7: **Block SelectRAM Block Diagram**

Timing Parameters

Table 2-4: **Block SelectRAM Timing Parameters**

Parameter	Function	Control Signal	Description
Setup and Hold Relative to Clock (CLK)			
T_{BxCK} = Setup time (before clock edge) T_{BCKx} = Hold time (after clock edge)			The following descriptions are for setup times only.
T_{BACK}/T_{BCKA}	Address inputs	ADDR	Time before the clock that address signals must be stable at the ADDR inputs of the block RAM.
T_{BDCK}/T_{BCKD}	Data inputs	DI	Time before the clock that data must be stable at the DI inputs of the block RAM.
T_{BECK}/T_{BCKE}	Enable	EN	Time before the clock that the enable signal must be stable at the EN input of the block RAM.
T_{BRCK}/T_{BCKR}	Synchronous Set/Reset	SSR	Time before the clock that the synchronous set/reset signal must be stable at the SSR input of the block RAM.
T_{BWCK}/T_{BCKW}	Write Enable	WE	Time before the clock that the write enable signal must be stable at the WE input of the block RAM.
Clock to Out			
T_{BCKO}	Clock to Output	CLK to DO	Time after the clock that the output data is stable at the DO outputs of the block RAM.
Clock			
T_{BPWH}	Clock	CLK	Minimum pulse width, high.
T_{BPWL}	Clock	CLK	Minimum pulse width, low.

Timing Characteristics

The timing diagram in Figure 2-8 describes a single-port block RAM in Write-First mode. The timing for Read-First and No-Change modes are similar (see chapter 2, block RAM section.)

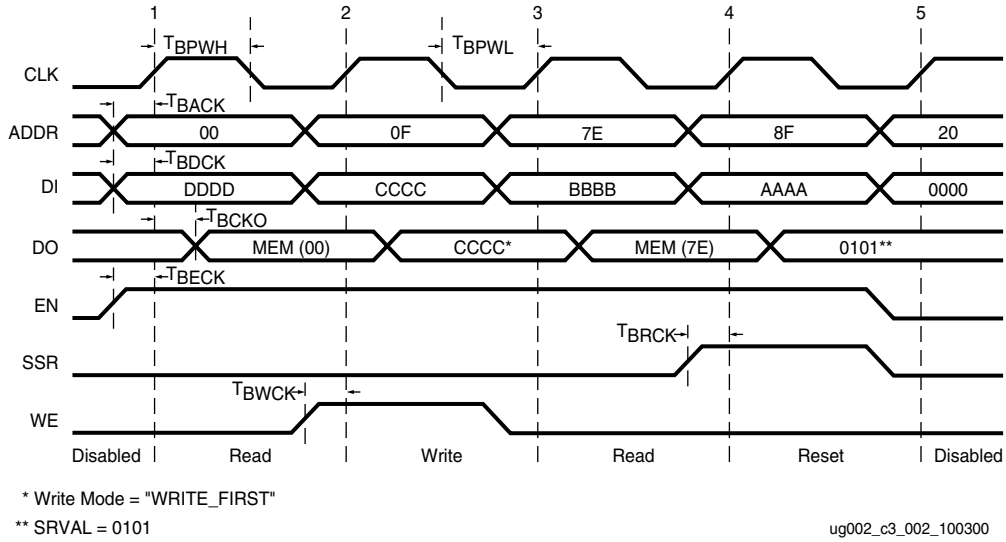


Figure 2-8: **Block SelectRAM Timing Diagram**

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

Clock Event 1: READ Operation

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

- T_{BACK} before Clock Event 1, address 00 becomes valid at the ADDR inputs of the block RAM.
- At time T_{BECK} before Clock Event 1, Enable goes High at the EN input of the block RAM, enabling the memory for the READ operation that follows.
- At time T_{BCKO} after Clock Event 1, the contents of the memory at address 00 become stable at the DO pins of the block RAM.

Clock Event 2: WRITE Operation

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

- At time T_{BACK} before Clock Event 2, address 0F becomes valid at the ADDR inputs of the block RAM.
- At time T_{BDCK} before Clock Event 2, data CCCC becomes valid at the DI inputs of the block RAM.
- At time T_{BWCK} before Clock Event 2, Write Enable becomes valid at the WE following the block RAM.
- At time T_{BCKO} after Clock Event 2, data CCCC becomes valid at the DO outputs of the block RAM.

Clock Event 4: SSR (Synchronous Set/Reset) Operation

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

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

Clock Event 5: Disable Operation

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

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

Timing Model

Figure 2-9 illustrates the delay paths associated with the implementation of block SelectRAM. This example takes the simplest paths on and off chip (these paths can vary greatly depending on the design). This timing model demonstrates how and where the block SelectRAM timing parameters are used.

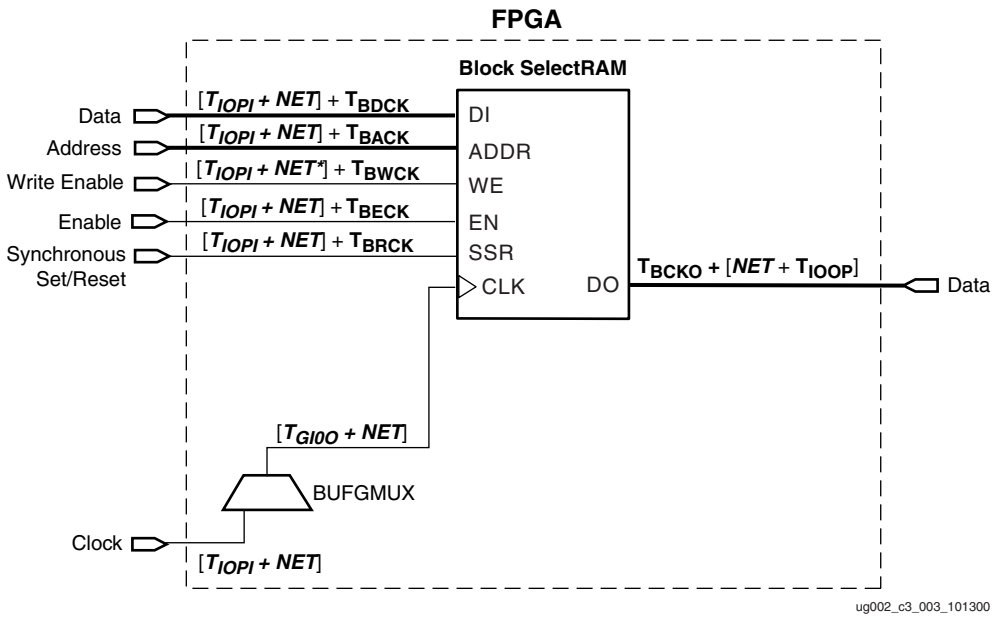


Figure 2-9: **Block SelectRAM Timing Model**

NET = Varying interconnect delays

$\mathbf{T_{IOPI}}$ = Pad to I-output of IOB delay

$\mathbf{T_{IOOP}}$ = O-input of IOB to pad delay

$\mathbf{T_{GI0O}}$ = BUFGMUX delay

Embedded Multiplier Timing Model

Introduction

This section explains all timing parameters associated with the use of embedded 18-bit x 18-bit multipliers in Virtex-II FPGAs (see Figure 2-10). The propagation delays through the embedded multiplier differ based on the size of the multiplier function implemented. The longest delay through the multiplier is to the highest order bit output (P35). Therefore, if an 18-bit x 18-bit signed multiplier is implemented, the worst-case delay for this function is the longest delay associated with the embedded multiplier block. If smaller (LSB) multipliers are used, shorter delays can be realized.

This section is intended to be used in conjunction with the section on switching characteristics in the **Virtex-II Data Sheet (DS031)** and the Timing Analyzer (TRCE) report from Xilinx software. For specific timing parameter values, refer to the *Virtex-II Data Sheet*.

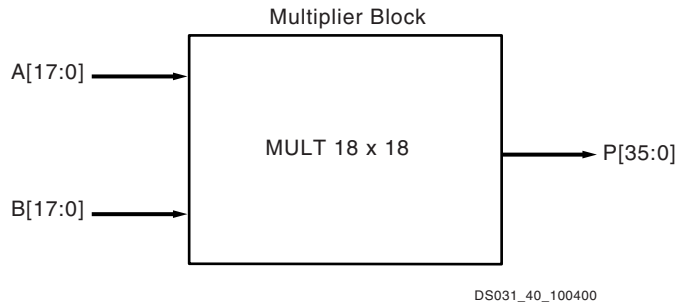


Figure 2-10: **Embedded 18-bit x 18-bit Multiplier Block**

Timing Parameters

Worst-Case Propagation Delays

The T_{MULT} timing parameter is reported by the Timing Analyzer software. These values correspond to the propagation delay through the multiplier to a specific output pin of the multiplier block. The shortest delay is to pin 0, the longest to pin 35. These parameters can be found in the table entitled "Multiplier Switching Characteristics" in Module 3 of the **Virtex-II Platform FPGA Data Sheet**. The delay-to-pin ratio is essentially linear (see Figure 2-11). This implies that smaller multiply functions are faster than larger ones. This is true as long as the LSB inputs are used.

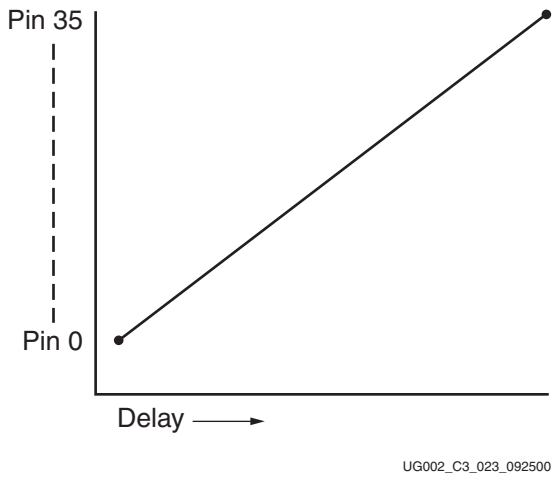


Figure 2-11: **Pin-to-Delay Ratio Curve**

Timing Characteristics

Figure 2-12 illustrates the result (outputs) of a 4-bit x 4-bit unsigned multiply implemented in an embedded multiplier block.

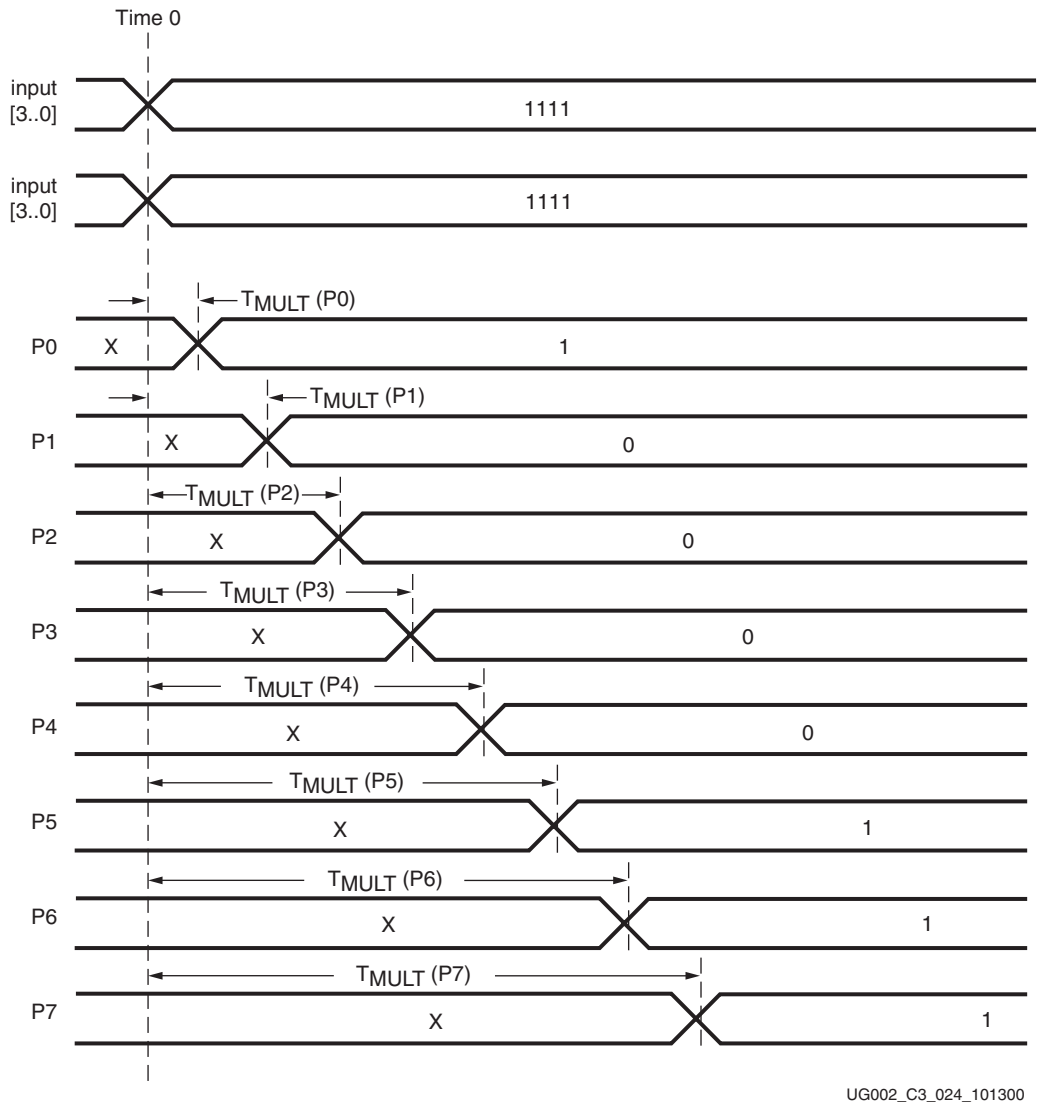


Figure 2-12: **Embedded Multiplier Block Timing Diagram**

At time 0 the two 4-bit numbers to be multiplied become valid at the A[0..3], B[0..3] inputs to the embedded multiplier. The result appears on the output pins P[0..7] in a staggered fashion. First, P0 becomes valid at time T_{MULT}(P0), followed by each subsequent output pin, until P7 becomes valid at time T_{MULT} (P7). In this case, the delay for this multiply function should correspond to that of Pin 7. In other words, the result is not valid until all output pins become valid.

IOB Timing Model

The following three sections describe all timing parameters associated with the Virtex-II IOB. These three sections are:

- IOB Input Timing Model and Parameters
- IOB Output Timing Model and Parameters
- IOB 3-State Timing Model and Parameters

This section is intended to be used in conjunction with the section on switching characteristics in the **Virtex-II Data Sheet (DS031)** and the Timing Analyzer (TRCE) report from Xilinx software. For specific timing parameter values, refer to the *Virtex-II Data Sheet*.

A Note on I/O Standard Adjustments:

The "IOB Input and Output Switching Characteristics Standard Adjustments" tables in the switching characteristics section of the *Virtex-II Data Sheet* are delay adders (+/-) to be added to all timing parameter values associated with the IOB and the Global Clock (see "Pin-to-Pin Timing Model" on page 48), if an I/O standard other than LVTTL is used.

All values specified in the *Virtex-II Data Sheet* for the parameters covered in this section are specified for LVTTL. If another I/O standard is used, these delays change. However, there are several exceptions. The following parameters associated with the pad going to high-impedance (3-State buffer OFF) should NOT be adjusted:

- T_{IOTHZ}
- $T_{IOTLPHZ}$
- T_{GTS}
- T_{IOCKHZ}
- T_{IOSRHZ}

IOB Input Timing Model and Parameters

Figure 2-13 illustrates IOB inputs.

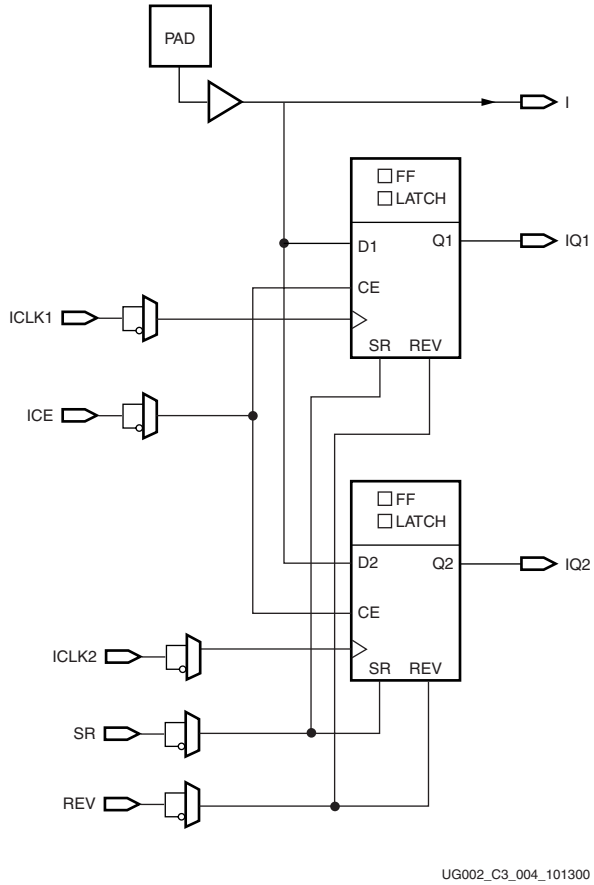


Figure 2-13: **Virtex-II IOB Input Diagram**

Timing Parameters

Table 2-5: **IOB Input Timing Parameters**

Parameter	Function	Control Signal	Description
Propagation Delays			
T_{IOPI}			Propagation delay from the pad to I output of the IOB with no delay adder.
T_{IOPID}			Propagation delay from the pad to I output of the IOB with the delay adder.
T_{IOPLI}			Propagation delay from the pad to IQ output of the IOB via transparent latch with no delay adder.
T_{IOPLID}			Propagation delay from the pad to IQ output of the IOB via transparent latch with the delay adder.

Table 2-5: **IOB Input Timing Parameters *(Continued)***

Parameter	Function	Control Signal	Description
Setup and Hold With Respect to Clock at IOB Input Register			
T_{xxCK} = Setup time (before clock edge) T_{xxCKxx} = Hold time (after clock edge)			The following descriptions are for setup times only.
T_{IOPICK}/T_{IOICKP}	ID input with NO delay		Time before the clock that the input signal from the pad must be stable at the ID input of the IOB Input Register, with no delay.
$T_{IOPICKD}/T_{IOICKPD}$	ID input with delay		Time before the clock that the input signal from the pad must be stable at the ID input of the IOB Input Register, with delay.
$T_{IOICECK}/T_{IOCKICE}$	ICE input		Time before the clock that the Clock Enable signal must be stable at the ICE input of the IOB Input Register.
$T_{IOSRCKI}$	SR input (IFF, synchronous)		Time before the clock that the Set/Reset signal must be stable at the SR input of the IOB Input Register.
Clock to Out			
T_{IOCKIQ}	Clock (CLK) to (IQ) output		Time after the clock that the output data is stable at the IQ output of the IOB Input Register.
Set/Reset Delays			
T_{IOSRIQ}	SR Input to IQ (asynchronous)		Time after the Set/Reset signal of the IOB is toggled that the output of the IOB input register (IQ) reflects the signal.
T_{GSRQ}	GSR to output IQ		Time after the Global Set/Reset is toggled that the output of the IOB input register (IQ) reflects the set or reset.

Timing Characteristics

Figure 2-14 illustrates IOB input register timing.

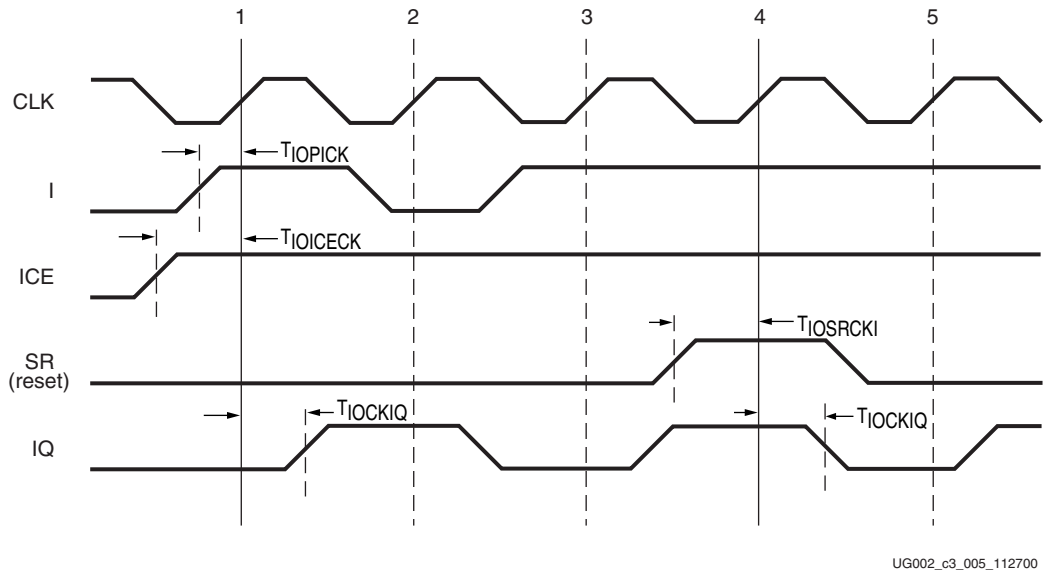


Figure 2-14: **IOB Input Register Timing Diagram**

Clock Event 1

- At time $T_{IOICECK}$ before Clock Event 1, the input clock enable signal becomes valid-high at the ICE input of the input register, enabling the input register for incoming data.
- At time T_{IOPICK} before Clock Event 1, the input signal becomes valid-high at the I input of the input register and is reflected on the IQ output of the input register at time T_{IOCKIQ} after Clock Event 1.

Clock Event 4

- At time $T_{IOSRCKI}$ before Clock Event 4 the SR signal (configured as synchronous reset in this case) becomes valid-high resetting the input register and reflected at the IQ output of the IOB at time T_{IOCKIQ} after Clock Event 4.

Timing Characteristics, DDR

Figure 2-15 illustrates IOB DDR input register timing.

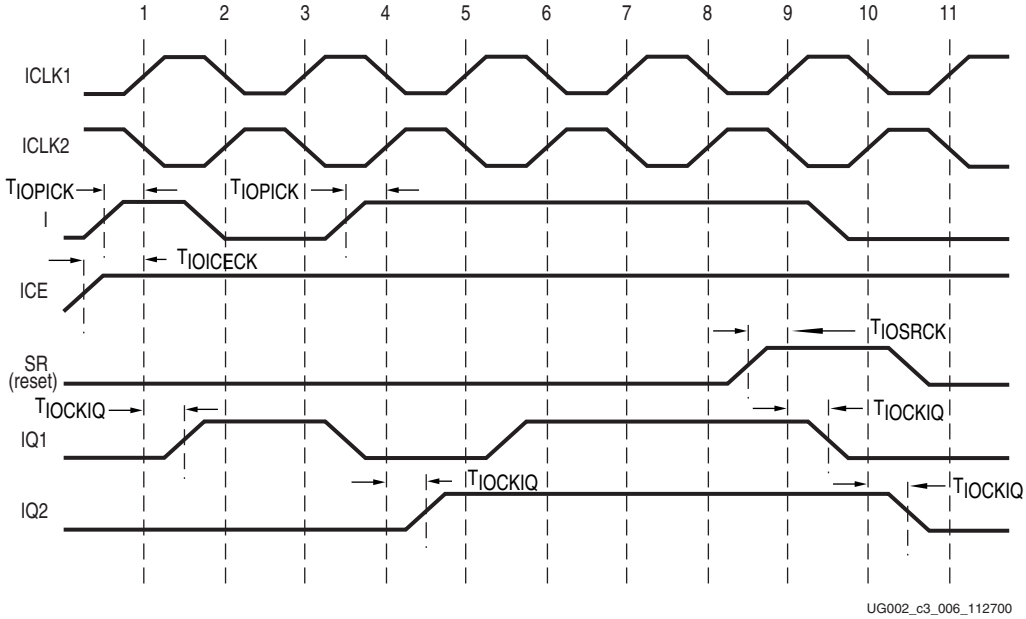


Figure 2-15: **IOB DDR Input Register Timing Diagram**

Clock Events

Clock Event 1

- At time $T_{IOICECK}$ before Clock Event 1 the input clock enable signal becomes valid-high at the ICE input of both of the DDR input registers, enabling them for incoming data. Since the ICE and I signals are common to both DDR registers, care must be taken to toggle these signals between the rising edges of ICLK1 and ICLK2 as well as meeting the register setup-time relative to both clocks.
- At time T_{IOPICK} before Clock Event 1 (rising edge of ICLK1) the input signal becomes valid-high at the I input of both registers and is reflected on the IQ1 output of input-register 1 at time T_{IOCKIQ} after Clock Event 1.

Clock Event 2

At time T_{IOPICK} before Clock Event 2 (rising edge of ICLK2) the input signal becomes valid-low at the I input of both registers and is reflected on the IQ2 output of input-register 2 at time T_{IOCKIQ} after Clock Event 2 (no change in this case).

Clock Event 9

At time $T_{IOSRCKI}$ before Clock Event 9 the SR signal (configured as synchronous reset in this case) becomes valid-high resetting input-register 1 (IQ1) at time T_{IOCKIQ} after Clock Event 9, and input-register 2 (IQ2) at time T_{IOCKIQ} after Clock Event 10.

IOB Output Timing Model and Parameters

Figure 2-16 illustrates IOB outputs.

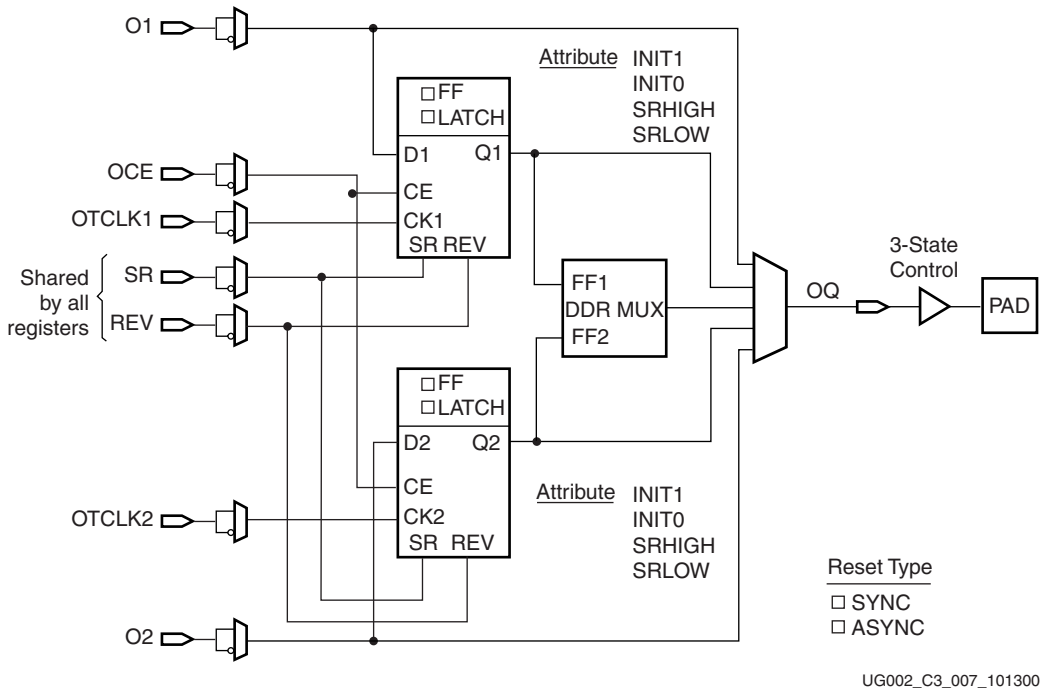


Figure 2-16: **Virtex-II IOB Output Diagram**

Timing Parameters

Table 2-6: **IOB Output Timing Parameters**

Parameter	Function	Control Signal	Description
Propagation Delays			
T_{IOOP}			Propagation delay from the O input of the IOB to the pad.
T_{IOOLP}			Propagation delay from the O input of the IOB to the pad via transparent latch.
Setup and Hold With Respect to Clock at IOB Output Register			
T_{xxCK} = Setup time (before clock edge) T_{xxCKxx} = Hold time (after clock edge)			The following descriptions are for setup times only.
T_{IOOCK}/T_{IOCKO}	O input		Time before the clock that data must be stable at the O input of the IOB Output Register.
$T_{IOOCECK}/T_{IOCKOCE}$	OCE input		Time before the clock that the Clock Enable signal must be stable at the OCE input of the IOB Output Register.
$T_{IOSRCKO}/T_{IOCKOSR}$	SR input (OFF)		Time before the clock that the Set/Reset signal must be stable at the SR input of the IOB Output Register.

Table 2-6: **IOB Output Timing Parameters *(Continued)***

Parameter	Function	Control Signal	Description
Clock to Out			
T_{IOCKP}	Clock (CLK) to pad		Time after the clock that the output data is stable at the pad.
Set/Reset Delays			
T_{IOSRP}	SR Input to pad (asynchronous)		Time after the Set/Reset input of the IOB is toggled that the pad reflects the set or reset.
T_{IOGSRQ}	GSR to pad		Time after the Global Set/Reset is toggled that the pad reflects the set or reset.

Timing Characteristics

Figure 2-17 illustrates IOB output register timing.

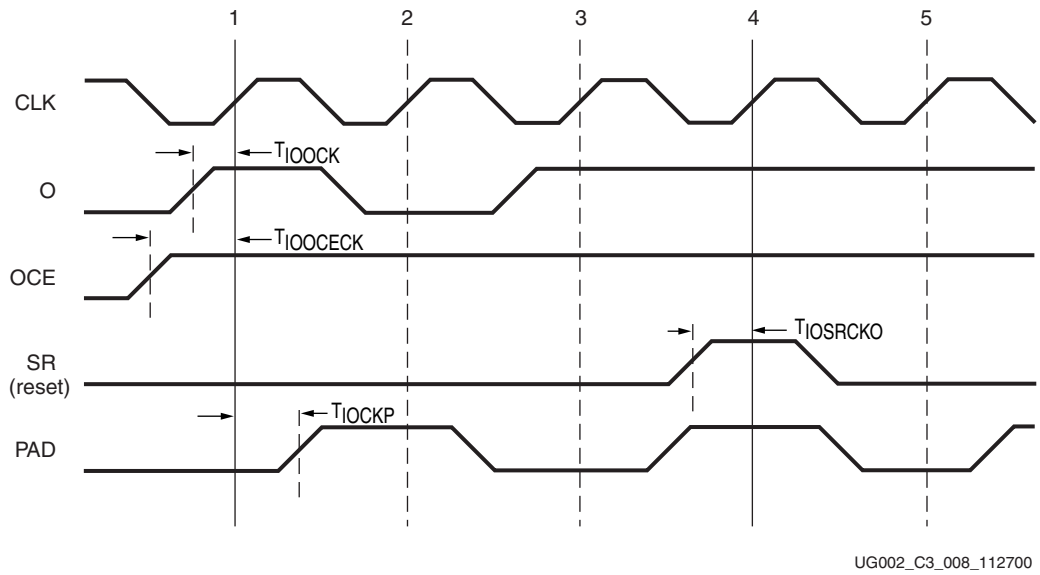


Figure 2-17: **IOB Output Register Timing Diagram**

Clock Event 1

- At time $T_{IOOCECK}$ before Clock Event 1, the output clock enable signal becomes valid-high at the OCE input of the output register, enabling the output register for incoming data.
- At time T_{IOOCK} before Clock Event 1, the output signal becomes valid-high at the O input of the output register and is reflected on the pad at time T_{IOCKP} after Clock Event 1.

Clock Event 4

At time $T_{IOSRCKO}$ before Clock Event 4, the SR signal (configured as synchronous reset in this case) becomes valid-high, resetting the output register and reflected on the pad at time T_{IOCKP} after Clock Event 4.

Timing Characteristics, DDR

Figure 2-18 illustrates IOB DDR output register timing.

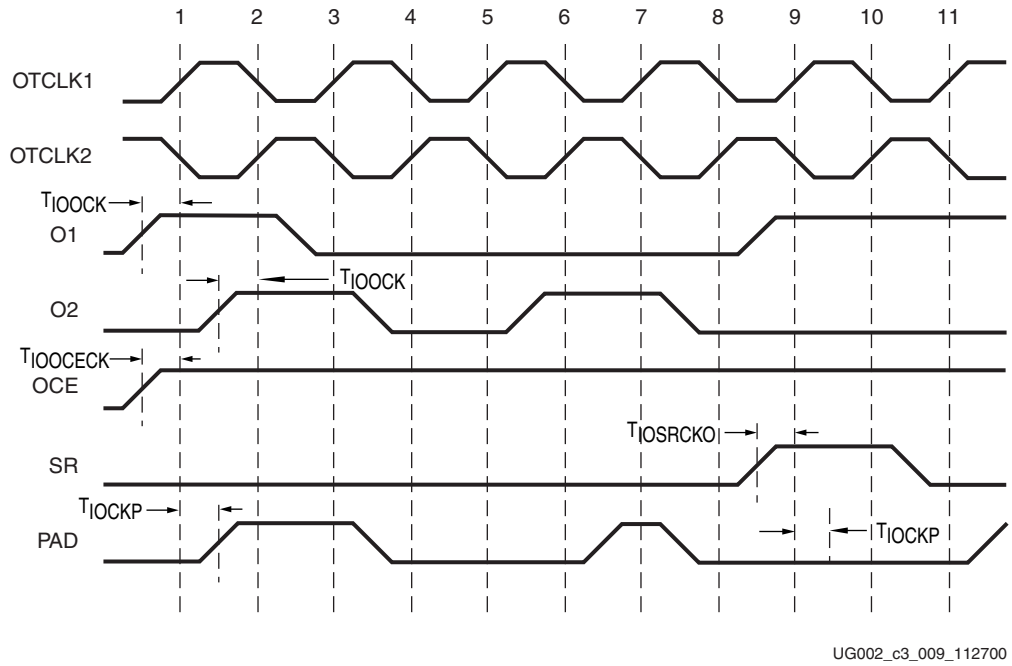


Figure 2-18: **IOB DDR Output Register Timing Diagram**

Clock Event 1

- At time $T_{IOOCECK}$ before Clock Event 1, the output clock enable signal becomes valid-high at the OCE input of both of the DDR output registers, enabling them for incoming data. Since the OCE signal is common to both DDR registers, care must be taken to toggle this signal between the rising edges of OTCLK1 and OTCLK2 as well as meeting the register setup-time relative to both clocks.
- At time T_{IOOCK} before Clock Event 1 (rising edge of OTCLK1), the output signal O1 becomes valid-high at the O1 input of output register 1 and is reflected on the pad at time T_{IOCKP} after Clock Event 1.

Clock Event 2*

At time T_{IOOCK} before Clock Event 2 (rising edge of OTCLK2), the output signal O2 becomes valid-high at the O2 input of output register 2 and is reflected on the pad at time T_{IOCKP} after Clock Event 2 (no change on the pad in this case).

Clock Event 9

At time $T_{IOSRCKO}$ before Clock Event 9, the SR signal (configured as synchronous reset in this case) becomes valid-high, resetting output-register 1 (reflected on the pad at time T_{IOCKP} after Clock Event 9) (no change in this case) and output-register 2 (reflected on the pad at time T_{IOCKP} after Clock Event 10) (no change in this case).

IOB 3-State Timing Model and Parameters

Figure 2-19 illustrates IOB 3-state timing

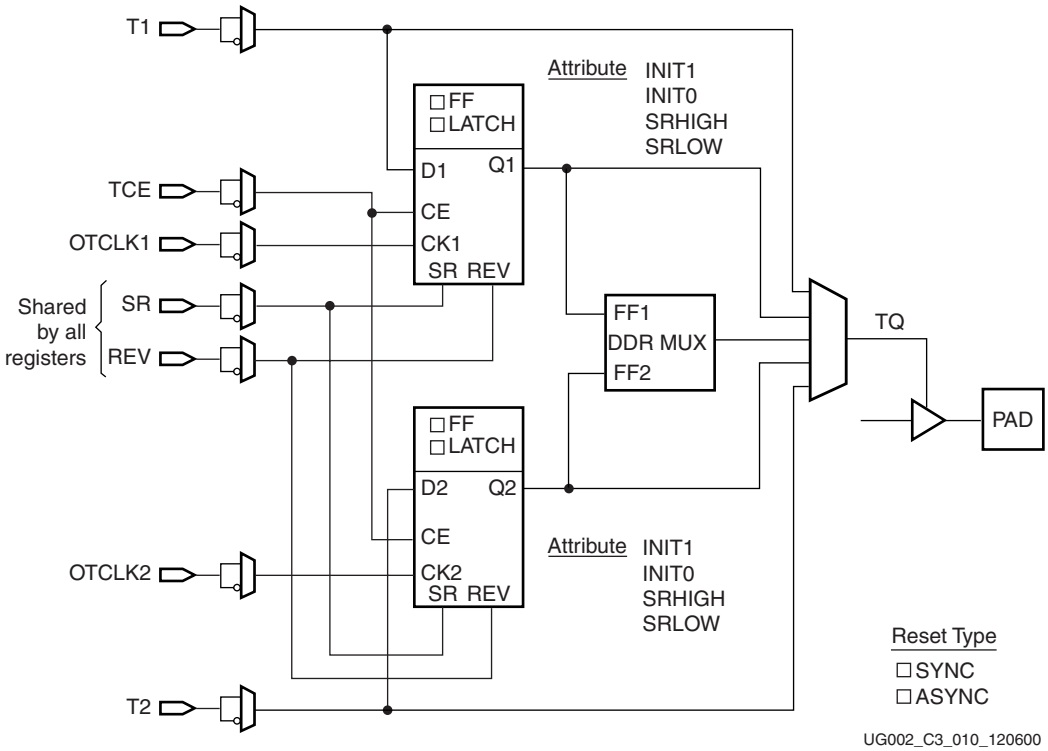


Figure 2-19: **Virtex-II IOB 3-State Diagram**

Timing Parameters

Table 2-7: **IOB 3-State Timing Parameters**

Parameter	Function	Control Signal	Description
Propagation Delays			
T_{IOTHZ}			Time after T input of the IOB is toggled that the pad goes to high-impedance.
T_{IOTON}			Time after the T input of the IOB is toggled that the pad goes from high-impedance to valid data.
$T_{IOTLPHZ}$			Time after the T input of the IOB via transparent latch is toggled that the pad goes to high-impedance.
$T_{IOTLPON}$			Time after the T input of the IOB via transparent latch is toggled that the pad goes from high-impedance to valid data.
T_{GTS}			Time after the Global 3-state signal is asserted that the pad goes to high-impedance.

Table 2-7: **IOB 3-State Timing Parameters *(Continued)***

Parameter	Function	Control Signal	Description
Setup and Hold With Respect to Clock at IOB 3-State Register			
T_{xxCK} = Setup time (before clock edge) T_{xxCKxx} = Hold time (after clock edge)			The following descriptions are for setup times only.
T_{IOTCK}/T_{IOCKT}	T input		Time before the clock that the signal must be stable at the T input of the IOB 3-state Register.
$T_{IOTCECK}/T_{IOCKTCE}$	TCE input		Time before the clock that the clock enable signal must be stable at the TCE input of the IOB 3-state Register.
$T_{IOSRCKT}/T_{IOCKTSR}$	SR input (TFF)		Time before the clock that the set/reset signal.
Clock to Out			
T_{IOCKHZ}	Clock (CLK) to pad High-Z		Time after clock that the pad goes to high-impedance.
T_{IOCKON}	Clock (CLK) to valid data on pad		Time after clock that the pad goes from high-impedance to valid data.
Set/Reset Delays			
T_{IOSRHZ}	SR Input to pad High-Z (asynchronous)		Time after the SR signal is toggled that the pad goes to high-impedance.
T_{IOSRON}	SR Input to valid data on pad (asynchronous)		Time after the SR signal is toggled that the pad goes from high-impedance to valid data.

Timing Characteristics

Figure 2-20 illustrates IOB 3-state register timing.

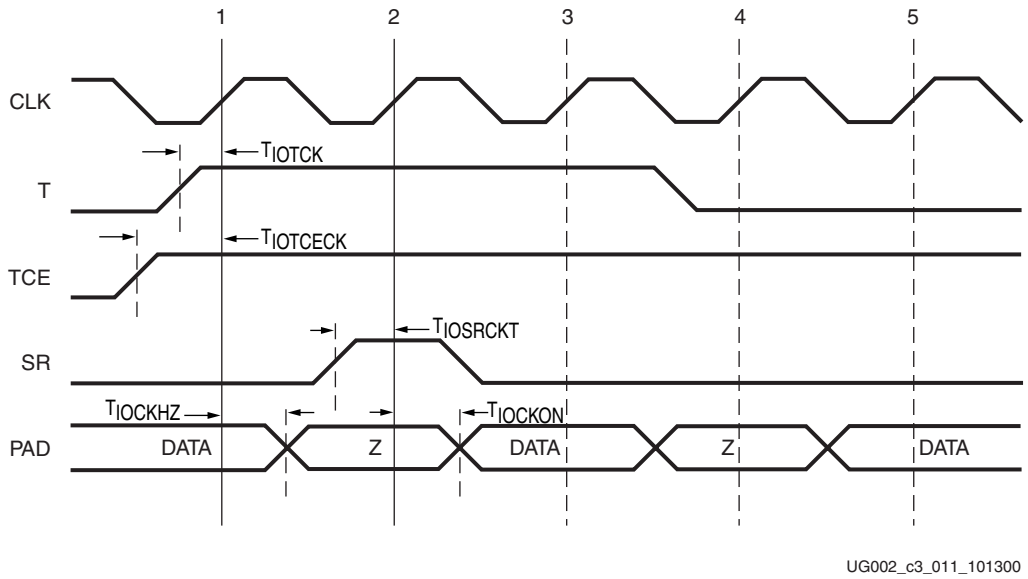


Figure 2-20: **IOB 3-State Register Timing Diagram**

Clock Event 1

- At time $T_{IOTCECK}$ before Clock Event 1, the 3-state clock enable signal becomes valid-high at the TCE input of the 3-state register, enabling the 3-state register for incoming data.
- At time T_{IOTCK} before Clock Event 1 the 3-state signal becomes valid-high at the T input of the 3-state register, returning the pad to high-impedance at time T_{IOCKHZ} after Clock Event 1.

Clock Event 2

At time $T_{IOSRCKT}$ before Clock Event 2, the SR signal (configured as synchronous reset in this case) becomes valid-high, resetting the 3-state register and returning the pad to valid data at time T_{IOSRON} after Clock Event 2.

Timing Characteristics, DDR

Figure 2-21 illustrates IOB DDR 3-state register timing.

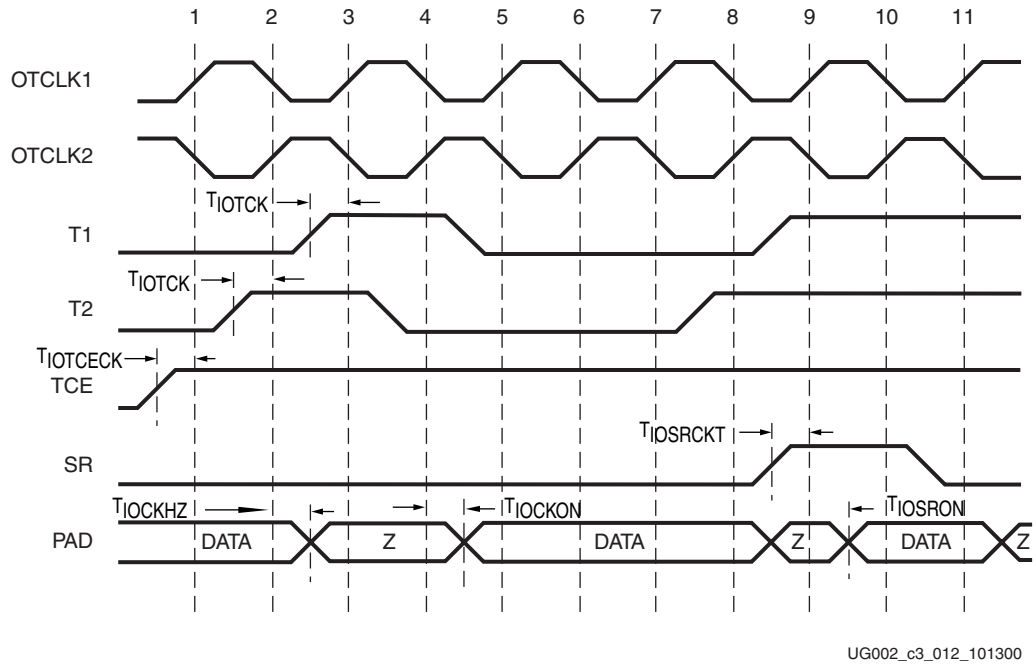


Figure 2-21: **IOB DDR 3-State Register Timing Diagram**

Clock Event 1

At time $T_{IOTCECK}$ before Clock Event 1, the 3-state clock enable signal becomes valid-high at the TCE input of both of the DDR 3-state registers, enabling them for incoming data. Since the TCE signal is common to both DDR registers, care must be taken to toggle this signal between the rising edges of OTCLK1 and OTCLK2 as well as meeting the register setup-time relative to both clocks.

Clock Event 2

At time T_{IOTCK} before Clock Event 2 (rising edge of OTCLK2), the 3-state signal T2 becomes valid-high at the T2 input of 3-state register 2, switching the pad to high-impedance at time T_{IOCKHZ} after Clock Event 2.

Clock Event 3

At time T_{IOTCK} before Clock Event 3 (rising edge of OTCLK1), the 3-state signal T1 becomes valid-high at the T1 input of 3-state register 1, keeping the pad at high-impedance for another half clock cycle (half the period of OTCLK1 or 2).

Clock Event 4

At time T_{IOTCK} before Clock Event 4 (rising edge of OTCLK2), the 3-state signal T2 becomes valid-low at the T2 input of 3-state register 2, switching the pad to valid data at time T_{IOCKON} after Clock Event 4. This is repeated for 3-state signal T1 at the following clock event (5) maintaining valid data on the pad until Clock Event 8.

Clock Event 8

At time T_{IOTCK} before Clock Event 8 (rising edge of OTCLK2), the 3-state signal T2 becomes valid-high at the T2 input of 3-state register 2, switching the pad to high-impedance at time T_{IOCKHZ} after Clock Event 8.

Clock Event 9

At time $T_{IOSRCKT}$ before Clock Event 9 (rising edge of OTCLK1), the SR signal (configured as synchronous reset in this case) becomes valid-high at the SR input of 3-state Register 1, returning the pad to valid data at time T_{IOSRON} after Clock Event 9.

Pin-to-Pin Timing Model

Introduction

This section explains the delays and timing parameters associated with the use of the Global Clock network and the DCM. These delays are true pin-to-pin delays relative to the Global Clock pin and an output or input pin with or without the DCM.

This section consists of two parts:

- Global Clock Input to Output
- Global Clock Setup and Hold

The former describes the delay from the Global Clock pin (with and without the DCM) to an output pin via an Output flip-flop. The latter describes the set-up time for an Input flip-flop from an input pin relative to the Global Clock pin (with and without the DCM).

The values reported in the switching characteristics section of the **Virtex-II Data Sheet (DS031)** are for LVTTL I/O standards. For different I/O standards, adjust these values with those shown in the "IOB Switching Characteristics Standard Adjustments" tables.

This section is intended to be used in conjunction with the section on switching characteristics in the *Virtex-II Data Sheet* and the Timing Analyzer (TRCE) report from Xilinx software. For specific timing parameter values, refer to the *Virtex-II Data Sheet*.

Global Clock Input to Output

Figure 2-22 illustrates the paths associated with the timing parameters defined in this section. Note that they differ only in their use of the DCM.

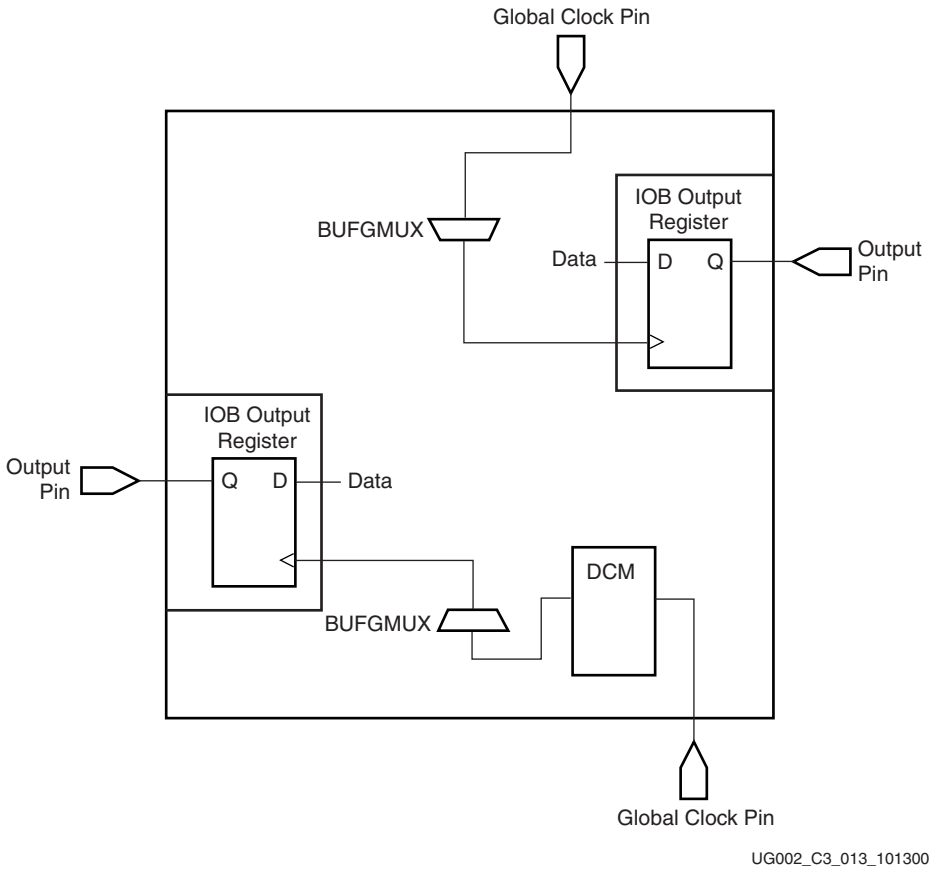


Figure 2-22: **Global Clock Input to Output Model**

Timing Parameters

Table 2-8: **Global Clock Input to Output Timing Parameters**

Parameter	Description
$T_{ICKOFDLL}$	Time after the Global Clock (pin), using the DCM, that the output data from an IOB Output flip-flop is stable at the output pin.
T_{ICKOF}	Time after the Global Clock (pin), without the DCM, that the output data from an IOB Output flip-flop is stable at the output pin.

Timing Characteristics

The waveforms depicted in Figure 2-23 demonstrate the relation of the Global Clock pin, the output data, and the use of the timing parameters.

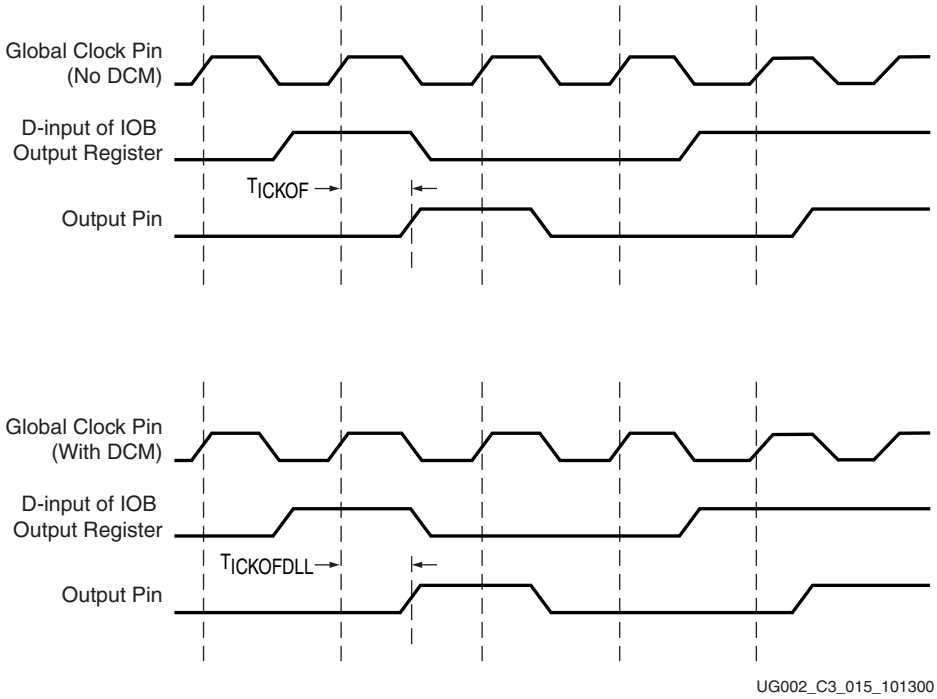


Figure 2-23: **Global Clock Input to Output Timing Diagram**

Global Clock Setup and Hold

Figure 2-24 illustrates the paths associated with the timing parameters defined in this section. Note, they differ only in their use of the DCM.

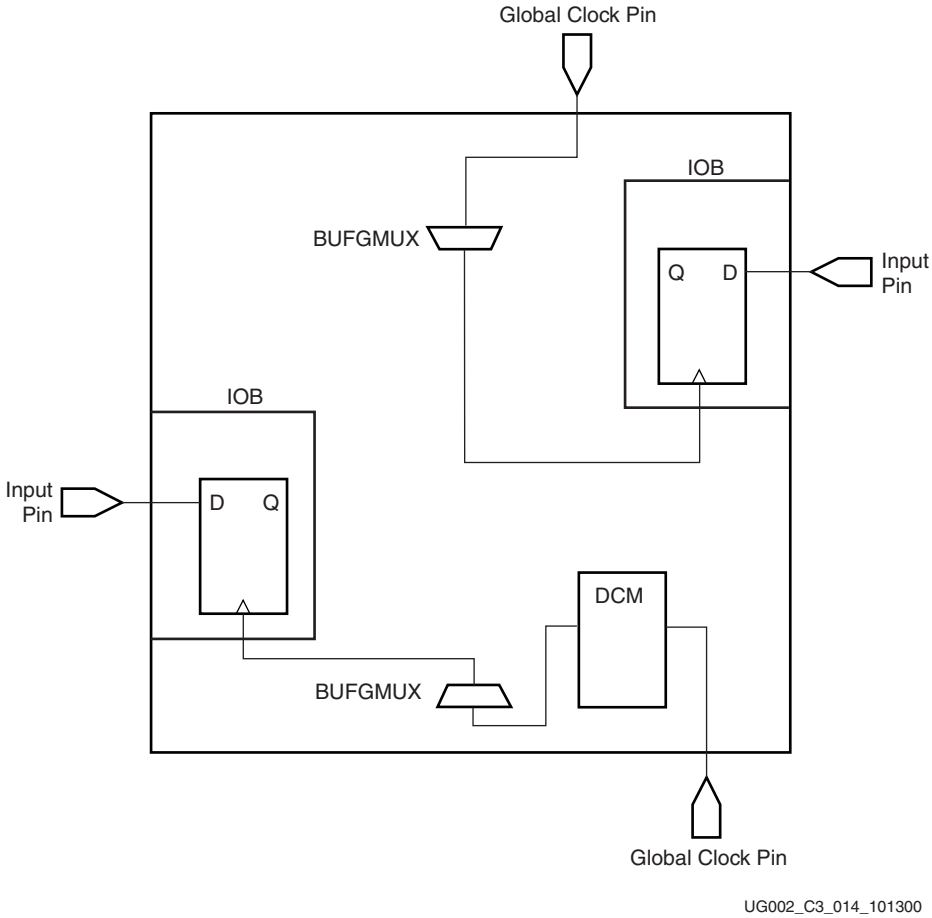


Figure 2-24: **Global Clock Setup and Hold Model**

Timing Parameters

Setup and Hold for Input Registers Relative to the Global Clock (pin):

- T_{PSDLL} / T_{PHDLL} - Time before the Global Clock (pin), with DCM, that the input signal must be stable at the D-input of the IOB input register.
- T_{PSFD} / T_{PHFD} - Time before the Global Clock (pin), without DCM, that the input signal must be stable at the D-input of the IOB input register.

  Note: T_{PSFD} = Setup time (before clock edge) and T_{PHFD} = Hold time (after clock edge). The previous descriptions are for setup times only.

Timing Characteristics

The waveforms depicted in Figure 2-25 demonstrate the relation of the Global Clock pin, the input data, and the use of the timing parameters.

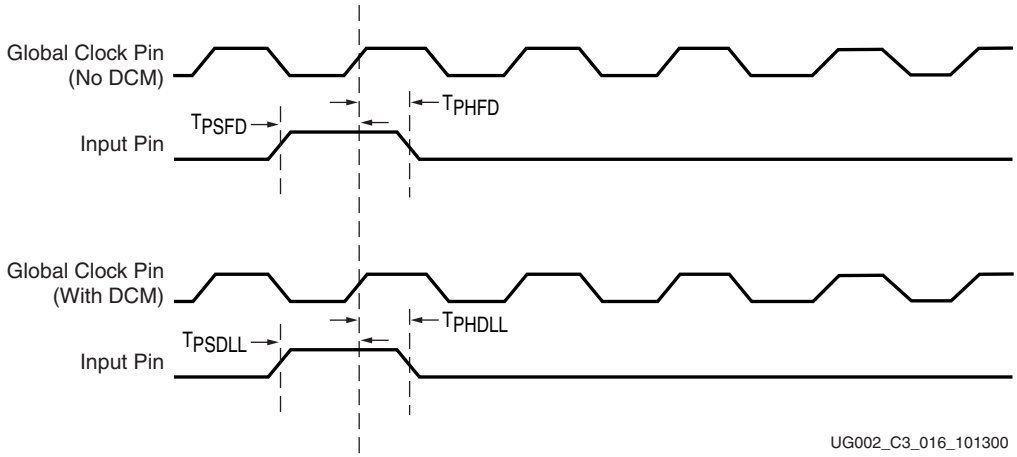


Figure 2-25: **Global Clock Setup and Hold Timing Diagram**

Digital Clock Manager Timing Model

This section describes the timing parameters associated with the Digital Clock Manager (DCM), which are reported in the **Virtex-II Data Sheet (DS031)**. Note that these parameters are not used by the Timing Analyzer software in the production of timing reports; they are all measured values and are fully characterized in silicon. For specific timing parameter values, refer to the *Virtex-II Data Sheet*. This section discusses the following:

- **Operating Frequency Ranges:** The minimum and maximum frequencies supported by the DCM for all clock inputs and outputs.
- **Input Clock Tolerances:** Input clock period (pulse widths), jitter, and drift requirements for proper function of the DCM for all clock inputs.
- **Output Clock Precision:** Output clock period jitter, phase offsets, and duty cycle for all clock outputs of the DCM (worst case).
- **Miscellaneous Timing Parameters:** DCM lock times, Tap delay and shifting range.

For a detailed description of input clock tolerance, jitter, and phase offset see the waveforms at the end of this section.

Operating Frequency Ranges

Figure 2-26 illustrates the DCM functional block and corresponding timing parameters for all clock inputs and outputs.

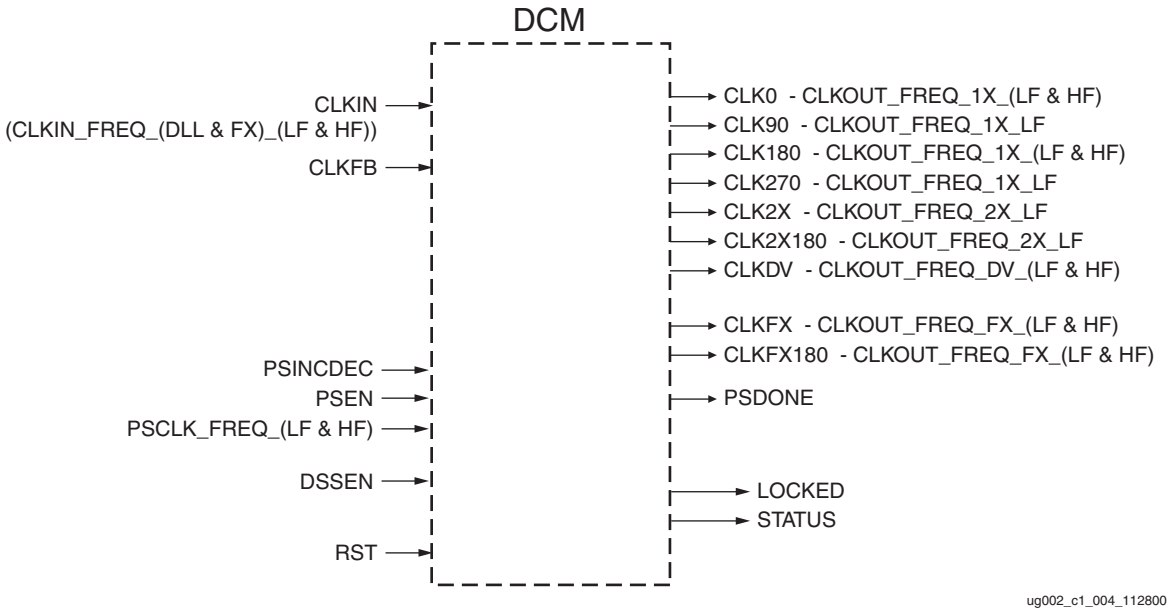


Figure 2-26: **DCM Functional Block: Operating Frequency Ranges**

Timing Parameters

Table 2-9: **Operating Frequency Range Parameters**

Parameter	Description
Low Frequency Mode	
CLKOUT_FREQ_1X_LF	The minimum and maximum frequency for the CLK0, CLK90, CLK180, CLK270 outputs of the DCM in low-frequency mode.
CLKOUT_FREQ_2X_LF	The minimum and maximum frequency for the CLK2X and CLK2X180 outputs of the DCM in low-frequency mode.

Table 2-9: **Operating Frequency Range Parameters *(Continued)***

Parameter	Description
CLKOUT_FREQ_DV_LF	The minimum and maximum frequency for the CLKDV output of the DCM in low-frequency mode.
CLKOUT_FREQ_FX_LF	The minimum and maximum frequency for the CLKFX and CLKFX180 outputs of the DCM in low-frequency mode.
CLKIN_FREQ_DLL_LF[1]	The minimum and maximum frequency for the CLKIN input to the DCM in low-frequency mode when using the delay-locked loop (DLL) outputs.
CLKIN_FREQ_FX_LF[2]	The minimum and maximum frequency for the CLKIN input to the DCM in low-frequency mode when using the FX outputs.
PSCLK_FREQ_LF	The minimum and maximum frequency for the PSCLK input to the DCM in low-frequency mode.
High Frequency Mode	
CLKOUT_FREQ_1X_HF	The minimum and maximum frequency for the CLK0, CLK180 outputs of the DCM in high-frequency mode.
CLKOUT_FREQ_DV_HF	The minimum and maximum frequency for the CLKDV output of the DCM in high-frequency mode.
CLKOUT_FREQ_FX_HF	The minimum and maximum frequency for the CLKFX and CLKFX180 outputs of the DCM in high-frequency mode.
CLKIN_FREQ_DLL_HF	The minimum and maximum frequency for the CLKIN input to the DCM in high-frequency mode when using the DLL outputs.
CLKIN_FREQ_FX_HF	The minimum and maximum frequency for the CLKIN input to the DCM in high-frequency mode when using the FX outputs.
PSCLK_FREQ_HF	The minimum and maximum frequency for the PSCLK input to the DCM in high-frequency mode.

Notes:
1. Delay-locked loop (DLL) outputs include: CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, and CLKDV.
2. FX outputs include: CLKFX and CLKFX180

Input Clock Tolerances

Timing Parameters

Table 2-10: **Input Clock Tolerance Parameters**

Parameter	Description
PSCLK_PULSE[1]	The minimum pulse width (HIGH and LOW) that the PSCLK input to the DCM can have over a range of frequencies.
CLKIN_PULSE	The minimum pulse width (HIGH and LOW) that the CLKIN input to the DCM can have over a range of frequencies. Also applies to PSCLK.
CLKFB_DELAY_VAR_EXT	The maximum allowed variation in delay (across environmental changes) of the feedback clock path when routed externally for board-level de-skew.
Low Frequency Mode	

Table 2-10: **Input Clock Tolerance Parameters *(Continued)***

Parameter	Description
CLKIN_CYC_JITT_DLL_LF	The maximum cycle-to-cycle jitter the CLKIN input to the DCM can have when using the DLL outputs in low-frequency mode.
CLKIN_CYC_JITT_FX_LF	The maximum cycle-to-cycle jitter the CLKIN input to the DCM can have when using the FX outputs in low-frequency mode.
CLKIN_PER_JITT_DLL_LF	The maximum period jitter the CLKIN input to the DCM can have when using the DLL outputs in low-frequency mode.
CLKIN_PER_JITT_FX_LF	The maximum period jitter the CLKIN input to the DCM can have when using the FX outputs in low-frequency mode.
High Frequency Mode	
CLKIN_CYC_JITT_DLL_HF	The maximum cycle-to-cycle jitter the CLKIN input to the DCM can have when using the DLL outputs in high-frequency mode.
CLKIN_CYC_JITT_FX_HF	The maximum cycle-to-cycle jitter the CLKIN input to the DCM can have when using the FX outputs in high-frequency mode.
CLKIN_PER_JITT_DLL_HF	The maximum period jitter the CLKIN input to the DCM can have when using the DLL outputs in high-frequency mode.
CLKIN_PER_JITT_FX_HF	The maximum period jitter the CLKIN input to the DCM can have when using the FX outputs in high-frequency mode.

Notes:

1. The frequencies applicable to CLKIN_PULSE range from 1 to >400 MHz. These frequencies also apply to PSCLK_PULSE. Since PSCLK can be less than 1 MHz, the pulse width under this condition is specified for PSCLK only.

Output Clock Precision

Timing Parameters

Table 2-11: **Output Clock Precision Parameters**

Parameter	Description
CLKOUT_PER_JITT_0	The maximum period jitter of the CLK0 output clock from the DCM (worst case).
CLKOUT_PER_JITT_90	The maximum period jitter of the CLK90 output clock from the DCM (worst case).
CLKOUT_PER_JITT_180	The maximum period jitter of the CLK180 output clock from the DCM (worst case).
CLKOUT_PER_JITT_270	The maximum period jitter of the CLK270 output clock from the DCM (worst case).
CLKOUT_PER_JITT_2X	The maximum period jitter of the CLK2X and CLK2X180 output clocks from the DCM (worst case).
CLKOUT_PER_JITT_DV1	The maximum period jitter of the CLKDV (integer division) output clock from the DCM (worst case).
CLKOUT_PER_JITT_DV2	The maximum period jitter of the CLKDV (non-integer division) output clock from the DCM (worst case).

Table 2-11: **Output Clock Precision Parameters *(Continued)***

Parameter	Description
CLKOUT_PER_JITT_FX	The maximum period jitter of the FX output clocks from the DCM (worst case).
CLKIN_CLKFB_PHASE	Maximum phase offset between the CLKIN and CLKFB inputs to the DCM.
CLKOUT_PHASE	Maximum phase offset between any DCM clock outputs.
CLKOUT_DUTY_CYCLE_DLL	The duty-cycle precision for all DLL outputs.
CLKOUT_DUTY_CYCLE_FX	The duty-cycle precision for the FX outputs.

Miscellaneous DCM Timing Parameters

Table 2-12: **Miscellaneous DCM Timing Parameters**

Parameter	Description
LOCK_DLL	Time required for DCM to lock over a range of clock frequencies when using the DLL outputs.
LOCK_FX	Time required for DCM to lock when using the FX outputs.
LOCK_DLL_FINE_SHIFT	Additional lock time when performing fine phase shifting.
FINE_SHIFT_RANGE	Absolute range for fine phase shifting.
DCM_TAP	Resolution of delay line.

The waveforms in Figure 2-27 demonstrate the relationship between clock tolerance, jitter, and phase.

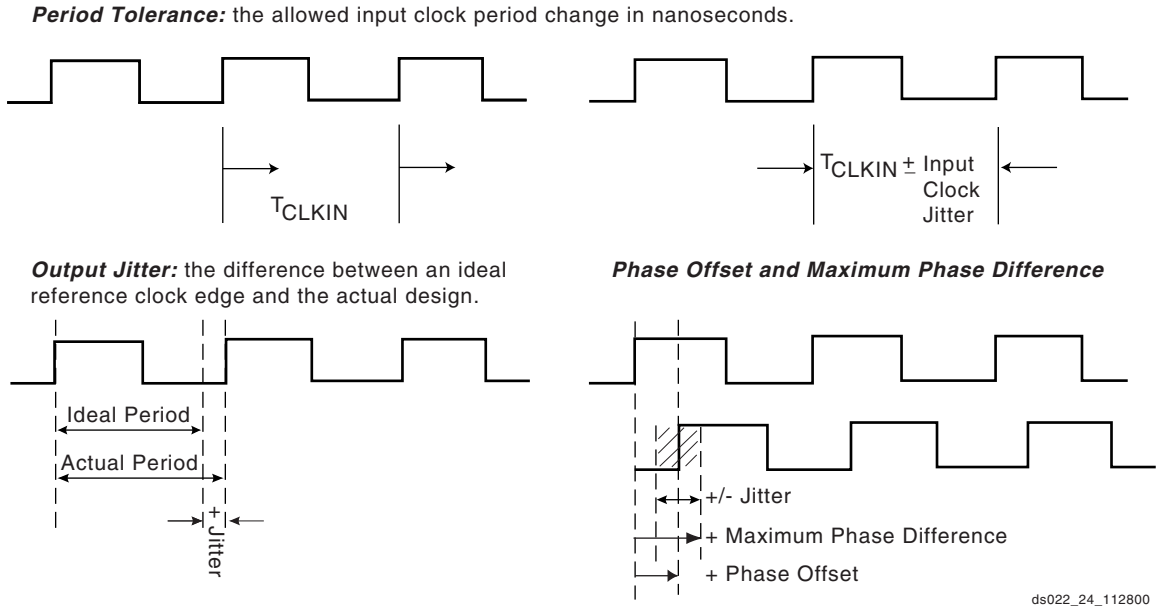


Figure 2-27: **DCM Jitter, Phase, and Tolerance Timing Waveforms**

Output jitter is period jitter measured on the DLL output clocks, excluding input clock jitter.

Phase offset between CLKIN and CLKFB is the worst-case fixed time difference between rising edges of CLKIN and CLKFB, excluding output jitter and input clock jitter.

Phase offset between clock outputs on the DLL is the worst-case fixed time difference between rising edges of any two DLL outputs, excluding output jitter and input clock jitter.

Maximum phase difference between CLKIN an CLKFB is the sum of output jitter and phase offset between CLKIN and CLKFB, or the greatest difference between CLKIN and CLKFB rising edges due to DLL alone (excluding input clock jitter).

Maximum phase difference between clock outputs on the DLL is the sum of output jitter and phase offset between any DLL clock outputs, or the greatest difference between any two DLL output rising edges due to DLL alone (excluding input clock jitter).

XILINX®

Chapter 3

Design Considerations

Summary

This chapter covers the following topics:

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

Introduction

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

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

Using Global Clock Networks

Introduction

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

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

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

Clock Distribution Resources

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

- 16 clock pads that can be used as regular user I/Os if not used as clock inputs. The 16 clock pads can be configured for any I/O standard, including differential standards (for example, LVDS, LVPECL, and so forth).
- 16 “IBUFG” elements that represent the clock inputs in a VHDL or Verilog design.
- 8 “IBUFGDS” elements (that is, attributes LVPECL_33, LVDS_25, LVDS_33, LDT_25, or ULVDS_25) that represent the differential clock input pairs in a VHDL or Verilog design. Each IBUFGDS replaces two IBUFG elements.
- 4 to 12 Digital Clock Managers (DCMs), depending on the device size, to de-skew and generate the clocks. For more information on DCMs, see "Using Digital Clock Managers (DCMs)" on page 80.
- 16 “BUFGMUX” elements that can consist of up to 16 global clock buffers (BUFG), global clock buffers with a clock enable (BUFGCE), or global clock multiplexers (BUFGMUX).

Figure 3-1 illustrates the placement of these clock resources in Virtex-II devices (the XC2V250 through the XC2V2000) that have eight DCMs.

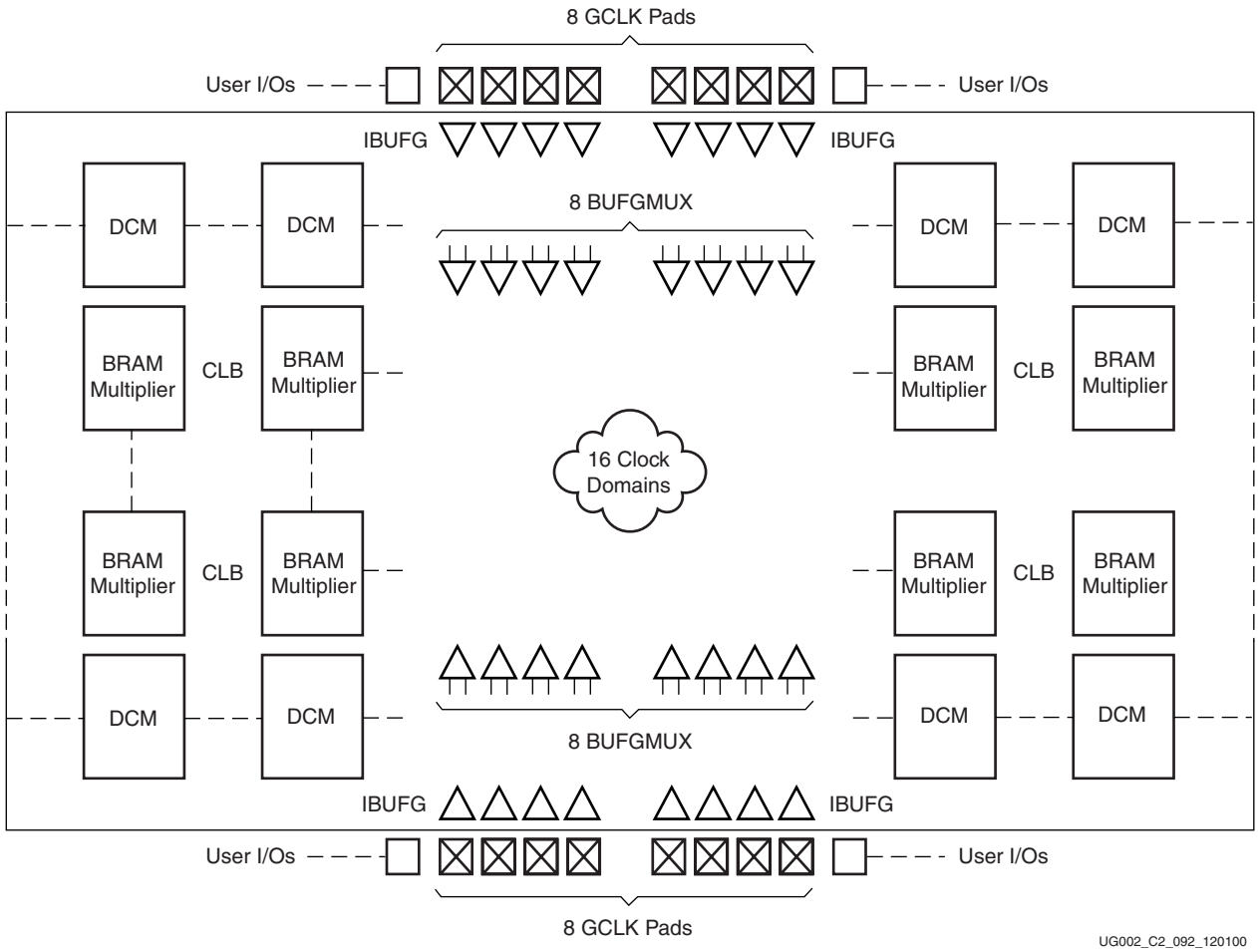


Figure 3-1: **Clock Resources in Virtex-II Devices**

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

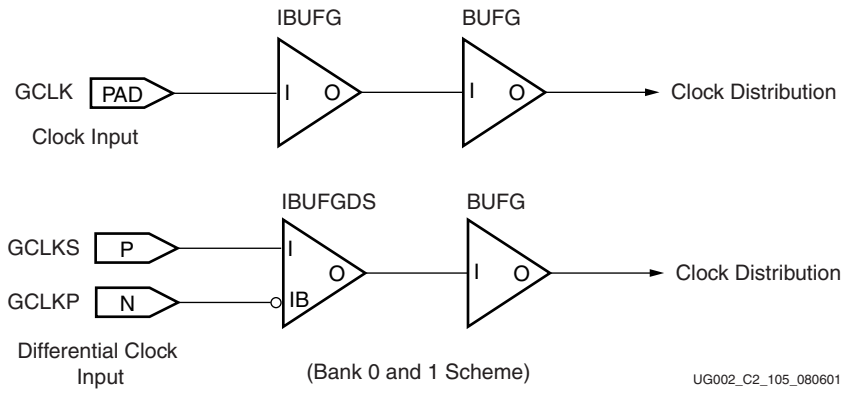


Figure 3-2: **Simple Clock Distribution (Bank 0 and 1 Scheme)**

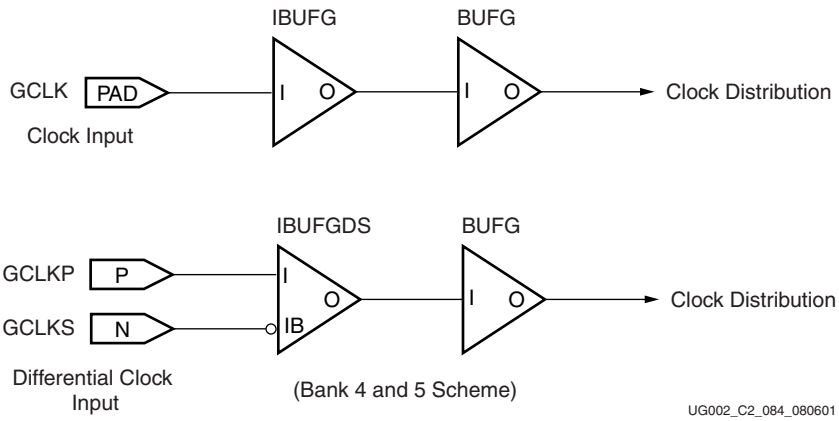


Figure 3-3: **Simple Clock Distribution (Bank 4 and 5 Scheme)**

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

A high frequency or adapted (frequency, phase, and so forth) clock distribution with low skew is implemented by using a DCM between the output of the IBUFG and the input of the BUFG, as shown in Figure 3-4. "Using Digital Clock Managers (DCMs)" on page 80 provides details about DCMs and their use.

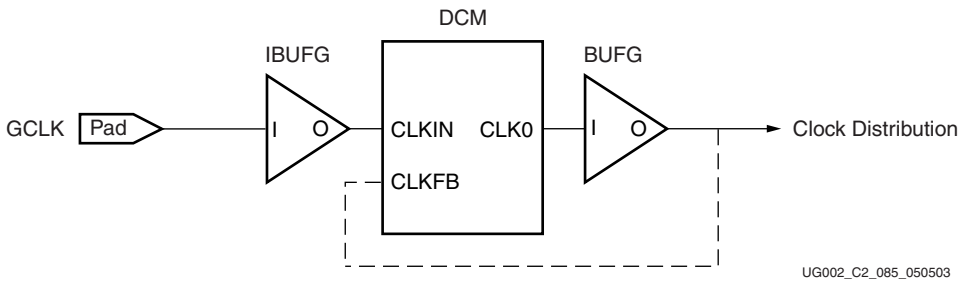


Figure 3-4: **Clock Distribution with DCM**

Clock distribution from internal sources is also possible with a BUFG only or with a DCM, as shown in Figure 3-5.

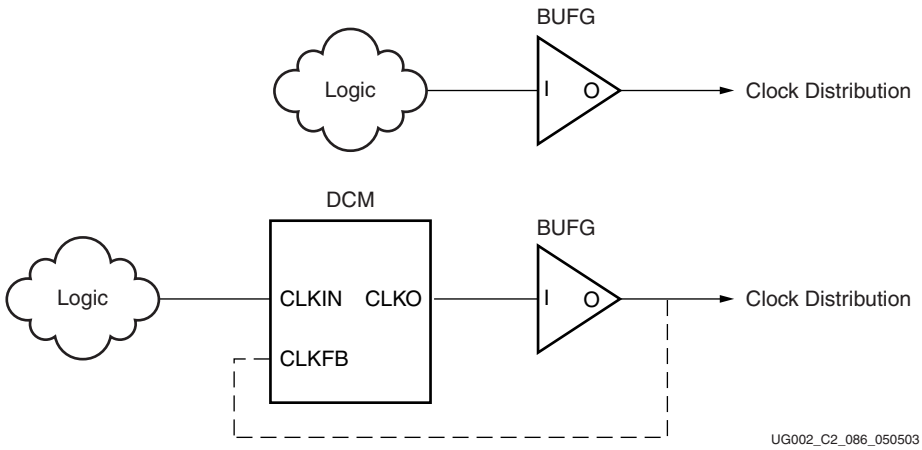


Figure 3-5: **Internal Logic Driving Clock Distribution**

Global Clock Inputs

The clock buffer inputs are fed either by one of the 16 clock pads (refer to the **Virtex-II Data Sheet (DS031)**), by the outputs of the DCM, or by local interconnect. Each clock buffer can be a synchronous "glitch-free" 2:1 multiplexer with two clock inputs and one select input. Internal logic (or alternatively a regular IOB) can feed the clock inputs. Any internal or external signal can drive the select input or clock enable input.

The possible inputs driving a global clock buffer or multiplexer are summarized in Table 3-1.

Table 3-1: **Inputs Driving Global Clock Buffers or DCMs**

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

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

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

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

Primary and Secondary Global Multiplexers

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

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

- Eight primary clock multiplexers
- Eight secondary clock multiplexers

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

Each Virtex-II device is divided into four quadrants: North-West, South-West, North-East, and South-East. Each quadrant has two primary and two secondary clock multiplexers. The clock multiplexers are indexed 0 to 7, with one primary and one secondary for each

index, alternating on the top and on the bottom (i.e., clock multiplexer "0P" at the bottom is facing clock multiplexer "0S" at the top).

In each device, the eight top/bottom clock multiplexers are divided into four primary and four secondary, indexed 0 to 7, as shown in Figure 3-6.

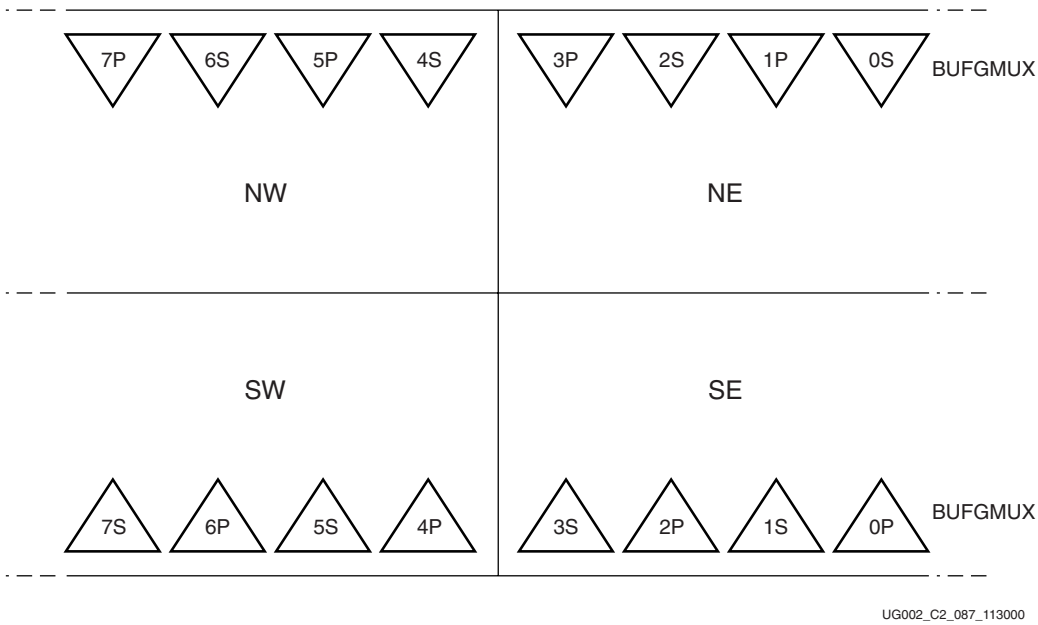


Figure 3-6: **Primary and Secondary Clock Multiplexer Locations**

Primary/Secondary: Rule 1

Considering two "facing" clock multiplexers (BUFG#P and BUFG#S), one or the other of these clock outputs can enter any quadrant of the chip to drive a clock within that quadrant, as shown in Figure 3-7. Note that the clock multiplexers "xP" and "xS" compete for

quadrant access. For example, BUFG0P output cannot be used in the same quadrant as BUFG0S.

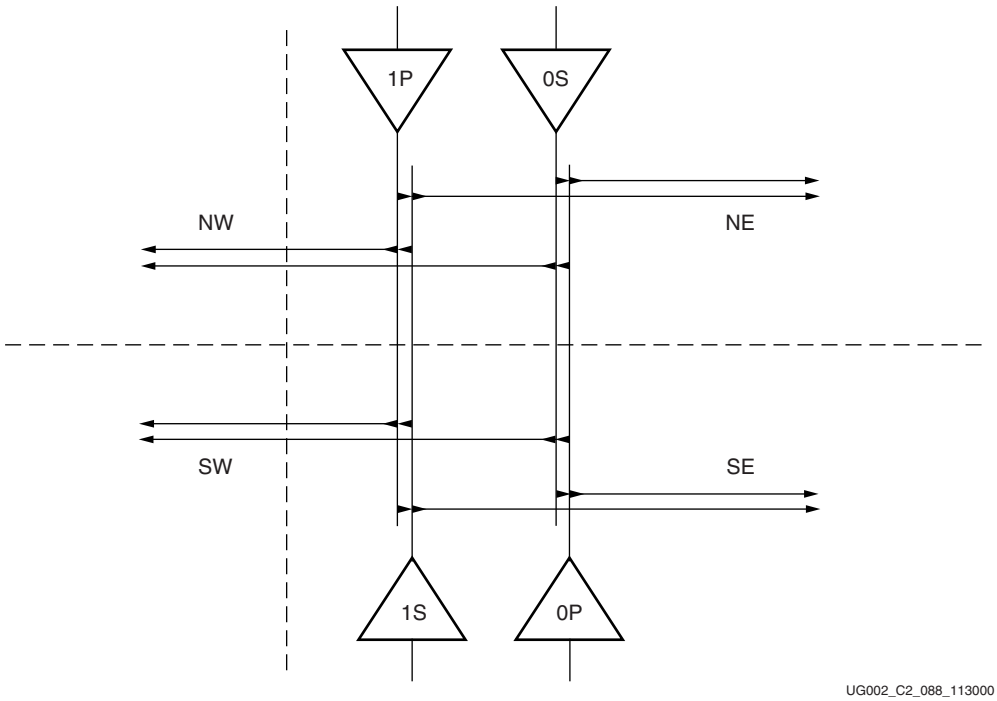


Figure 3-7: **Facing BUFG#P and BUFG#S Connections**

Primary/Secondary: Rule 2

In a BUFGCE or BUFGMUX configuration, shared inputs have to be considered. Any two adjacent clock multiplexers share two inputs, as shown in Figure 3-8. The clock multiplexer "1P" and "0S" have common I0/I1 and I1/I0 inputs.

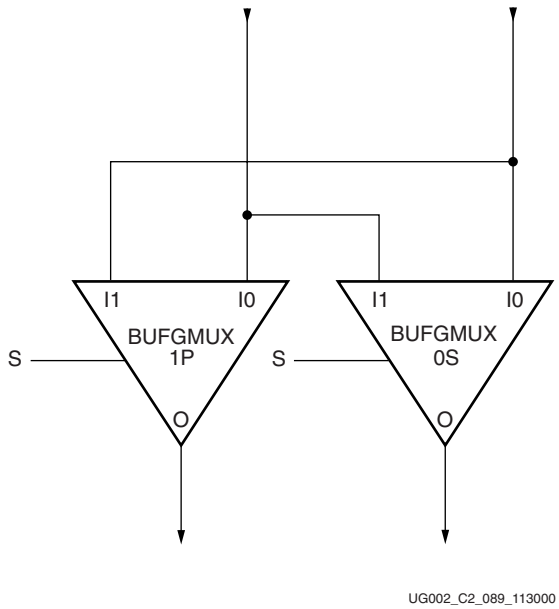


Figure 3-8: **Clock Multiplexer Pair Sharing Clock Multiplexer Inputs**

Table 3-2 lists the clock multiplexer pairs in any Virtex-II device. The primary multiplexer inputs I1/I0 are common with the corresponding secondary multiplexer inputs I0/I1 (i.e.,

Primary I1 input is common with secondary I0 input, and primary I0 input is common with secondary I1 input).

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

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

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

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

Primary/Secondary Usage

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

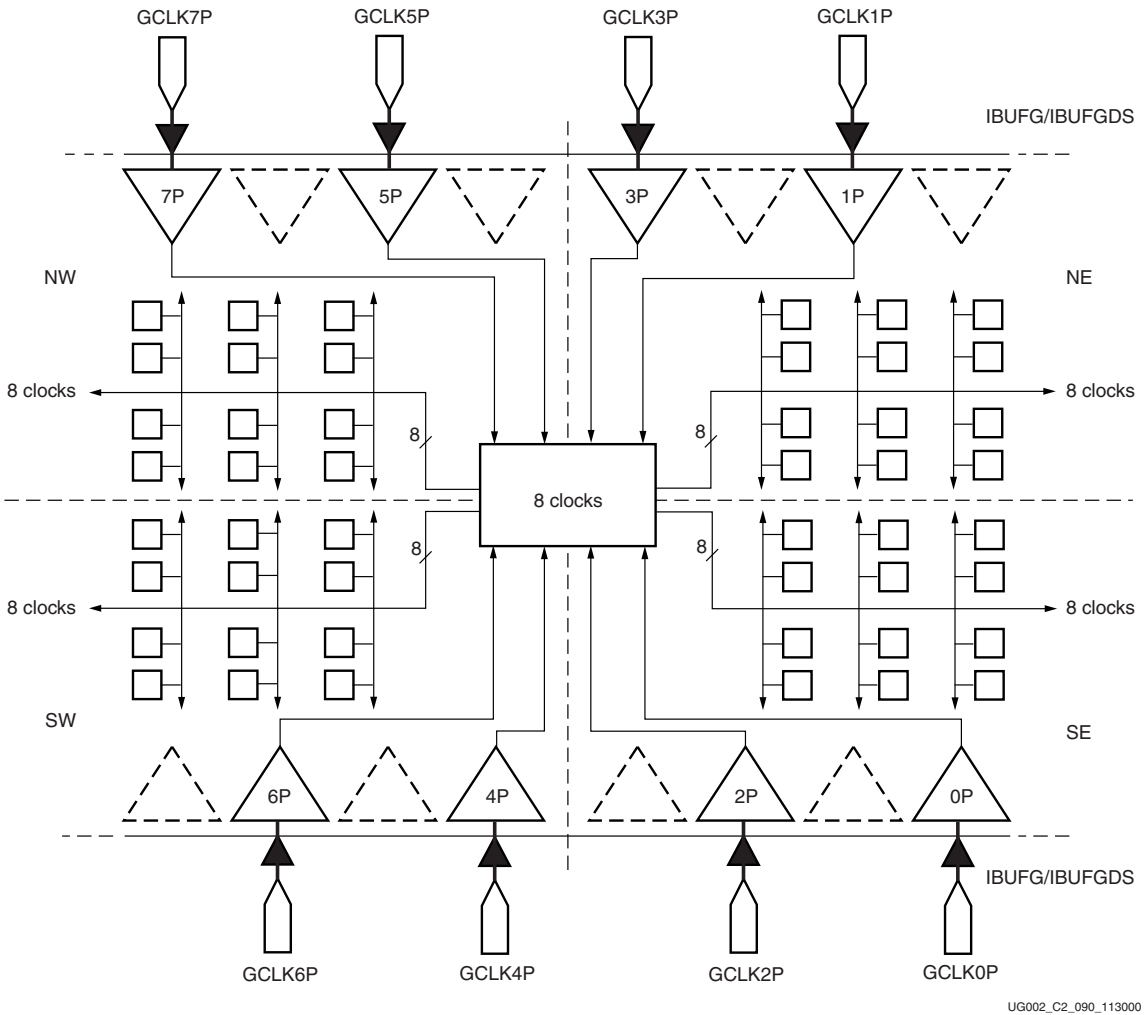


Figure 3-9: **Eight Global Clocks Design**

DCM Clocks

The four clock pins (IBUFG) in a quadrant can feed all DCMs in the same edge of the device. The clock-to-out and setup times are identical for all DCMs. Up to four clock outputs per DCM can be used to drive any clock multiplexer on the same edge (top or bottom), as shown in Figure 3-10.

BUFG Exclusivity

Each DCM has a restriction on the number of BUFGs it can drive on its (top or bottom) edge. Pairs of buffers with shared dedicated routing resources exist such that only one buffer from each dedicated pair can be driven by a single DCM. The exclusive pairs for each edge are: 0:4, 1:5, 2:6, and 3:7.

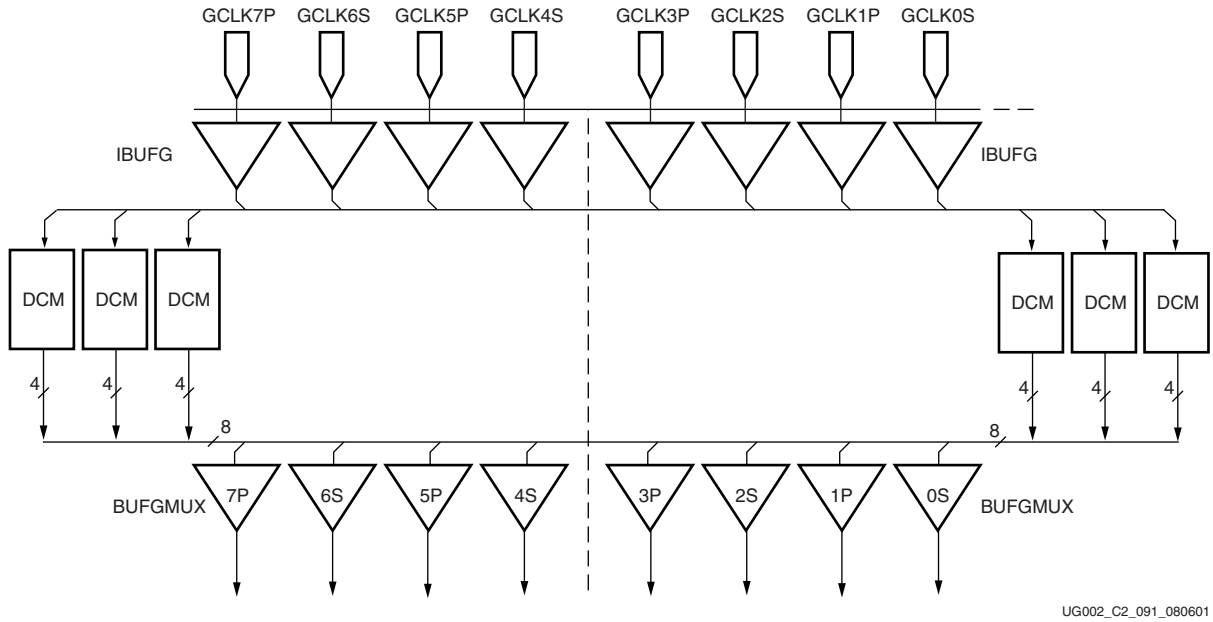


Figure 3-10: **DCM Clocks**

Clock Output

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

Eight clock buffers are in the middle of the top edge and eight are in the middle of the bottom edge. Any of these 16 clock buffer outputs can be used in any quadrant, up to a maxi-

mum of eight clocks per quadrant, as illustrated in Figure 3-11, provided there is not a primary vs. secondary conflict.

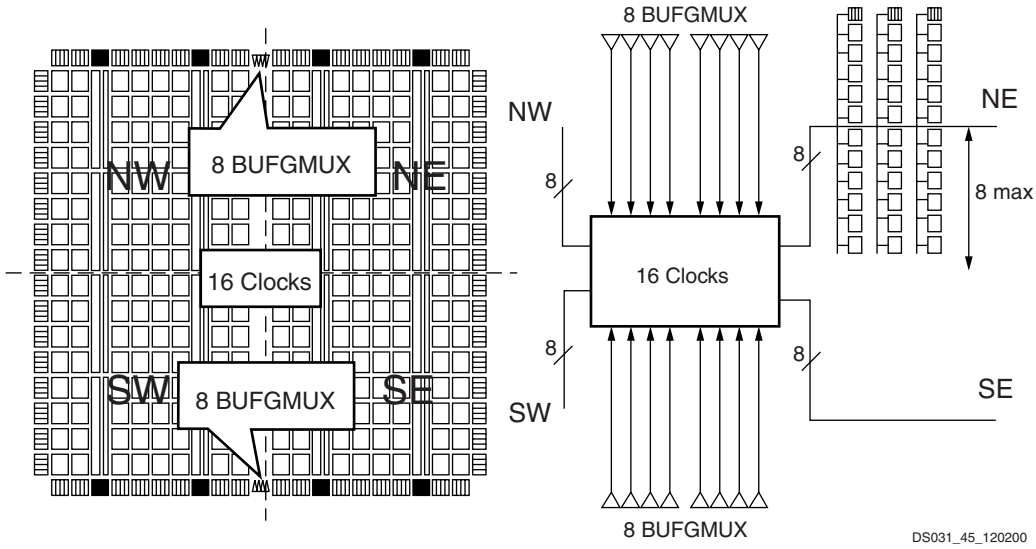


Figure 3-11: **Clock Buffer Outputs per Quadrant**

Designs with more than eight clocks must be floorplanned manually or automatically, distributing the clocks in each quadrant. As an example, a design with 16 clocks can be floorplanned as shown in Figure 3-12.

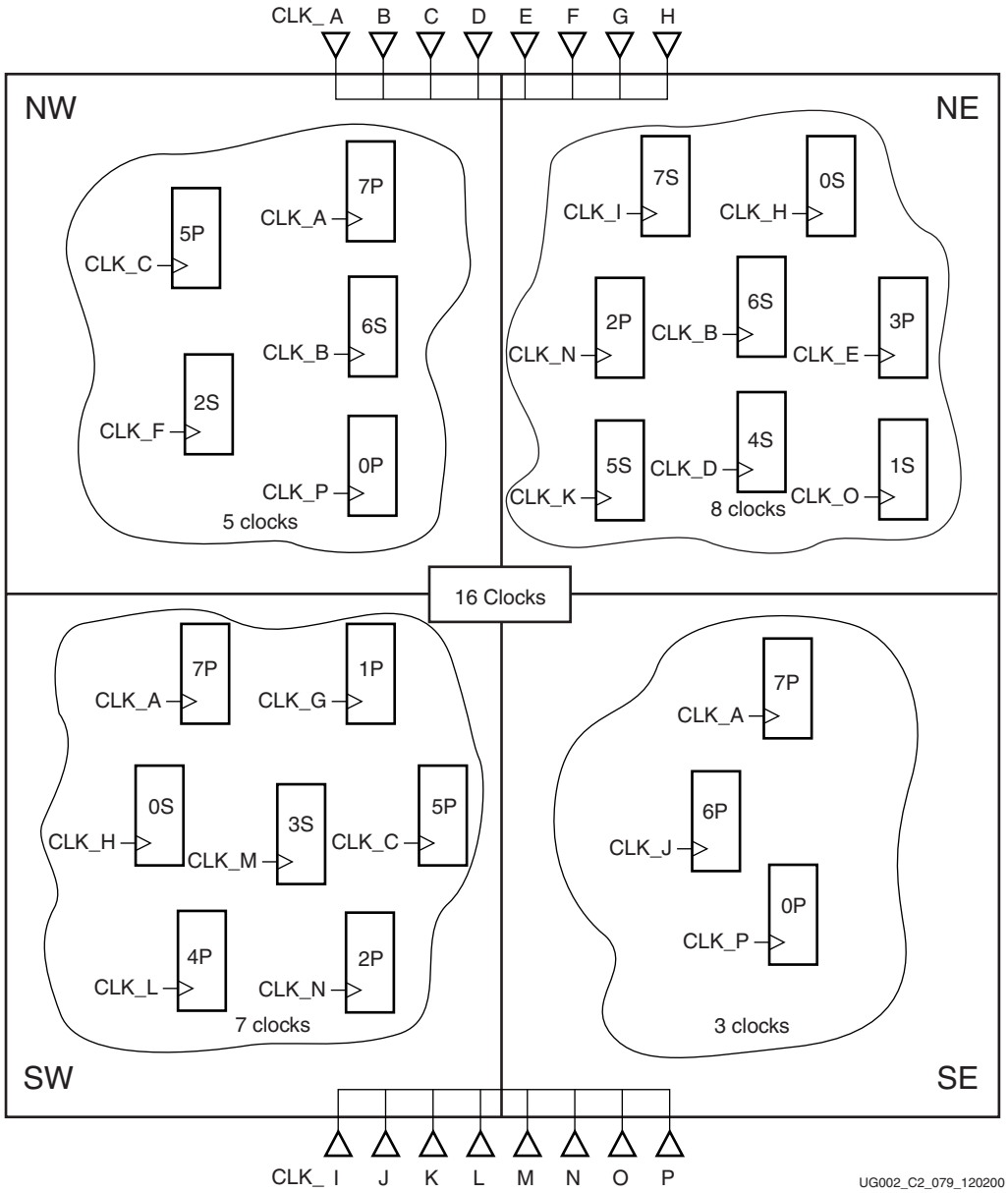


Figure 3-12: **16-Clock Floorplan**

The clock nets and clock buffers in this example are associated as shown in Table 3-4.

Table 3-4: **Clock Net Association With Clock Buffers**

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

CLK_A is used in three quadrants, and the other clocks are used in one or two quadrants, regardless of the position of the clock buffers (multiplexers), as long as they are not competing to access the same quadrant. (That is, CLK_A (BUFG7P) cannot be used in the same quadrant with CLK_I (BUFG7S). Refer to "Primary/Secondary: Rule 1" on page 64.) In other words, two buffers with the same index (0 to 7) cannot be used in the same quadrant.

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

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

Power Consumption

Clock trees have been designed for low skew and low-power operation. Any unused branch is disconnected, as shown in Figure 3-13.

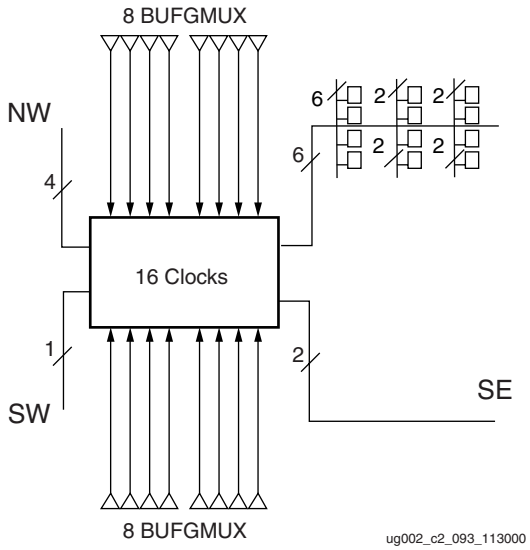


Figure 3-13: **Low-Power Clock Network**

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

The frequency synthesizer capability of the DCM can generate the low (or high) frequency clock from a single source clock, as illustrated in Figure 3-14. (See "Using Digital Clock Managers (DCMs)" on page 80).

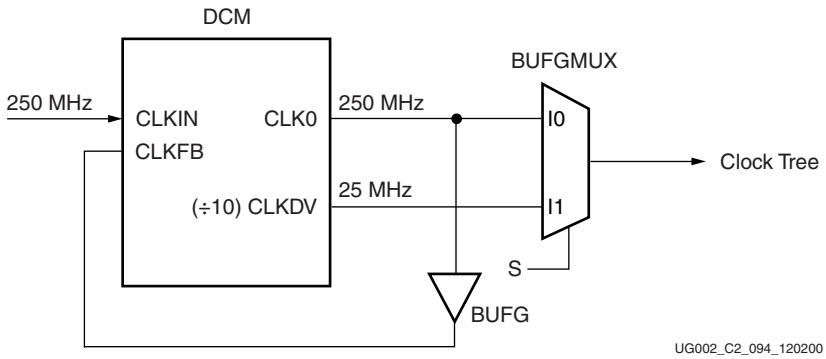


Figure 3-14: **Dynamic Power Reduction Scheme**

Library Primitives and Submodules

The primitives in Table 3-5 are available with the input, output, and control pins listed.

Table 3-5: **Clock Primitives**

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

Refer to "Using Single-Ended SelectI/O-Ultra Resources" on page 174 for a list of the attributes available for IBUFG and Refer to "Using LVDS I/O" on page 241 for a list of the attributes available for IBUFGDS.

The submodules in Table 3-6 are available with the input, output, and control pins listed.

Table 3-6: **Clock Submodules**

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

Primitive Functions

IBUFG

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

IBUFGDS

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

BUFG

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

BUFG is a global clock buffer with one clock input and one clock output, driving a low-skew clock distribution network. The output follows the input, as shown in Figure 3-15.

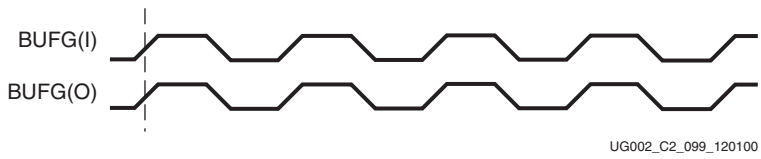


Figure 3-15: **BUFG Waveforms**

BUFGMUX and BUFGMUX_1

BUFGMUX (see Figure 3-16) can switch between two unrelated, even asynchronous clocks. Basically, a Low on S selects the I0 input, a High on S selects the I1 input. Switching from one clock to the other is done in such a way that the output High and Low time is never shorter than the shortest High or Low time of either input clock. As long as the presently selected

clock is High, any level change of S has no effect for BUFGMUX. For BUFGMUX_1, as long as the presently selected clock is Low, any level change of S has no effect.

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

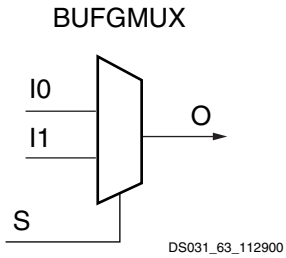


Figure 3-16: **Virtex-II BUFGMUX or BUFGMUX_1 Function**

Operation of the BUFGMUX Circuit

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

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

Figure 3-17 shows a switchover from CLK0 to CLK1.

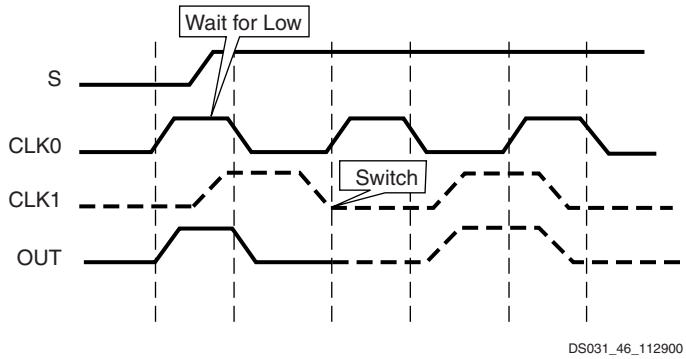


Figure 3-17: **BUFGMUX Waveform Diagram**

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

Operation of the BUFGMUX_1 Circuit

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

The two clock inputs can be asynchronous with regard to each other, and the S input can change at any time, except for a short setup time prior to the falling edge of the presently

selected clock; that is, prior to the falling edge of the BUFGMUX output O. Violating this setup time requirement can result in an undefined runt pulse output.

Figure 3-18 shows a switchover from CLK0 to CLK1.

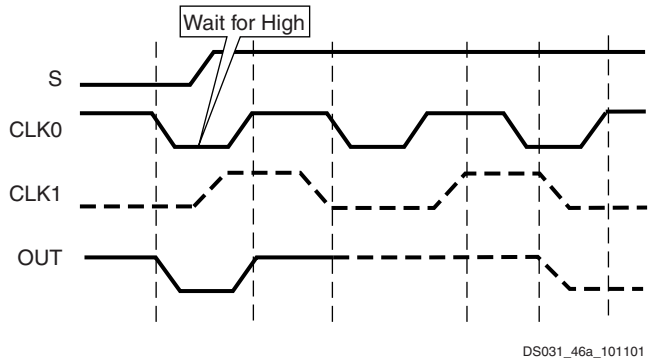


Figure 3-18: **BUFGMUX_1 Waveform Diagram**

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

Submodules

BUFGCE and BUFGCE_1

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

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

Operation of the BUFGCE Circuit

If the CE input (see Figure 3-19) is active (High) prior to the incoming rising clock edge, this Low-to-High-to-Low clock pulse passes through the clock buffer. Any level change of CE during the incoming clock High time has no effect.

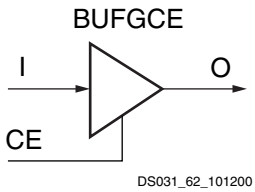


Figure 3-19: **Virtex-II BUFGCE or BUFGCE_1 Function**

If the CE input is inactive (Low) prior to the incoming rising clock edge, the following clock pulse does not pass through the clock buffer, and the output stays Low. Any level change of CE during the incoming clock High time has no effect. CE must not change during a short setup window just prior to the rising clock edge on the BUFGCE_1 input I. Violating this setup time requirement can result in an undefined runt pulse output.

This means the output stays Low when the clock is disabled, but it completes the clock-High pulse when the clock is being disabled, as shown in Figure 3-20.

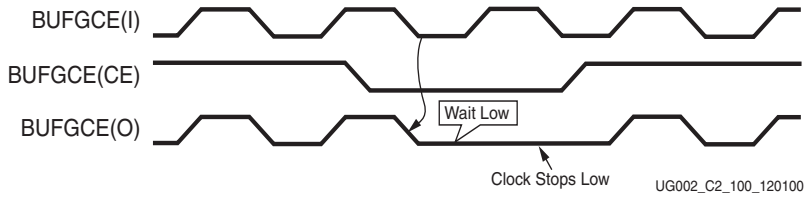


Figure 3-20: **BUFGCE Waveforms**

Operation of the BUFGCE_1 circuit

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

If the CE input is inactive (Low) prior to the incoming falling clock edge, the following clock pulse does not pass through the clock buffer, and the output stays High. Any level change of CE during the incoming clock Low time has no effect. CE must not change during a short setup window just prior to the falling clock edge on the BUFGCE input I. Violating this setup time requirement can result in an undefined runt pulse output.

This means the output stays High when the clock is disabled, but it completes the clock-Low pulse when the clock is being disabled, as shown in Figure 3-21.

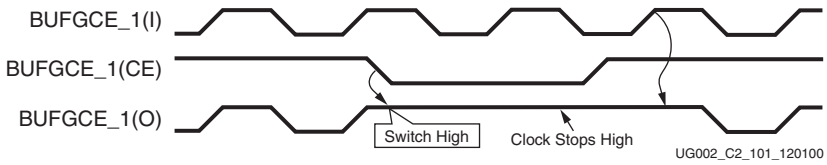


Figure 3-21: **BUFGCE_1 Waveforms**

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

Summary

Table 3-7 shows the maximum resources available per Virtex-II device.

Table 3-7: **Resources per Virtex-II Device (from XC2V40 to XC2V8000)**

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

Characteristics

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

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

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

Location Constraints

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

```
INST  "clock_buffer_instance_name" LOC="BUFGMUX#P/S";
```

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

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

Secondary Clock Network

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

VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available as examples (see "VHDL and Verilog Templates" on page 75) for all primitives and submodules.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

VHDL and Verilog Templates

The following are templates for primitives:

- BUFGMUX_INST
- BUFGMUX_1_INST

The following are templates for submodules:

- BUFGCE_SUBM
- BUFGCE_1_SUBM

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

VHDL Template

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

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

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

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

Verilog Template

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

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

output  O;

wire GND;

assign GND = 1'b0;

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

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

output  O;

wire VCC;

assign VCC = 1'b1;

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

Using Digital Clock Managers (DCMs)

Overview

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

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

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

- **Frequency Synthesis**: Separate outputs provide a doubled frequency (CLK2X and CLK2X180). Another output (CLKDV) provides a frequency that is a specified fraction of the input frequency (÷1.5, ÷2, ÷2.5, and so forth, up to ÷15 and ÷16.)

  Two other outputs (CLKFX and CLKFX180) provide an output frequency that is derived from the input clock by simultaneous frequency division and multiplication. The user can specify any integer multiplier (M) and divisor (D) within the range specified in the DCM Timing Parameters section of the **Virtex-II Data Sheet (DS031)**. An internal calculator figures out the appropriate tap selection, so that the output edge coincides with the input clock whenever that is mathematically possible. For example, M=9 and D=5, multiply the frequency by 1.8, and the output rising edge is coincident with the input rising edge every 5 input periods = every 9 output periods.

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

  The user can also dynamically and repetitively move the phase forwards or backwards by one unit of the clock period divided by 256. Note that any such phase shift is always invoked as a specific fraction of the clock period, but is always implemented by moving delay taps with a resolution of DCM_TAP (see the DCM Timing Parameters section in the **Virtex-II Data Sheet (DS031)**).

- **General Control Signals**: The input is an asynchronous reset; when High, it resets the entire DCM (all clock outputs, LOCKED, and STATUS signals are brought LOW). The LOCKED output is High when all enabled DCM circuits have locked. The active High STATUS outputs indicate the following:
  - Phase Shift Overflow (STATUS[0])
  - CLKIN Stopped (STATUS[1])
  - CLKFX Stopped (STATUS[2])

When using DCMs it is important to observe the V_{CCAUX} recommended operating noise specification in the **Virtex-II Data Sheet (DS031)**. Power distribution design details are available in Xilinx Application Note **XAPP623**.

Clock De-Skew

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

Any four of the nine outputs of the DCM can be used to drive a global clock network. All DCM outputs can drive general interconnect at the same time; for example, DCM output can be used to generate board-level clocks. The well-buffered global clock distribution network minimizes clock skew caused by loading differences. By monitoring a sample of the output clock (CLK0 or CLK2X), the de-skew circuit compensates for the delay on the routing network, effectively eliminating the delay from the external input port to the individual clock loads within the device.

Figure 3-22 shows all of the inputs and outputs relevant to the DCM de-skew feature.

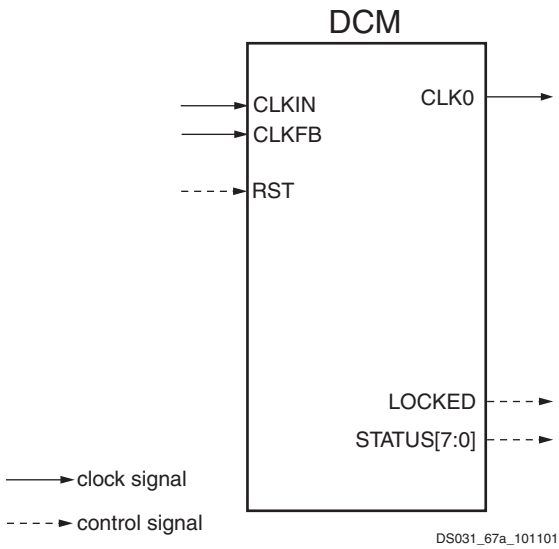


Figure 3-22: **Clock De-Skew Outputs**

The de-skew feature can also act as a clock mirror. By driving the CLK0 or CLK2X output off-chip and then back in again, the de-skew feature can be used to de-skew a board-level clock serving multiple devices. Figure 3-23 shows an example.

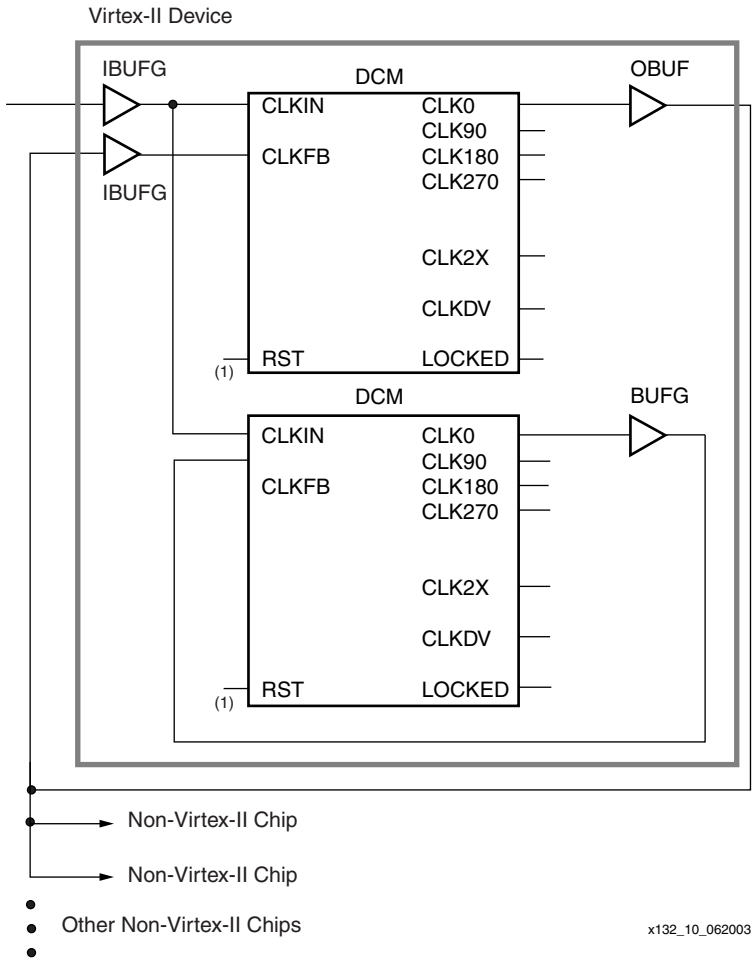


Figure 3-23: **Using De-Skew as a Clock Mirror**

Notes:

1. For a discussion of when to reset the DCM, see section "External Feedback" on page 84.

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

Operation

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

For optimum performance, the Virtex-II DCM uses a discrete digital delay line, which is a series of buffer elements each with an intrinsic delay of less than DCM_TAP (see AC characteristics in the **Virtex-II Data Sheet (DS031)**).

A de-skew circuit works by inserting delay between the input clock and the feedback clock until the two rising edges align, putting the two clocks 360 degrees out of phase, which means they are in phase. When the edges from the input clock line up with the edges from the feedback clock, the DCM achieves "lock." The two clocks have no discernible difference. Thus, the DCM output clock compensates for the delay in the clock distribution network, effectively removing the delay between the source clock and its loads.

Input Clock Requirements

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

The output clock signal of a DCM, essentially a delayed version of the input clock signal, reflects any instability on the input clock in the output waveform. A DCM cannot improve the input jitter. The DCM input clock requirements are specified in the **Virtex-II Data Sheet (DS031)**.

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

Input Clock Changes

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

While the DCM is in the locking process, no input clock edge can be missing. Once locked, it is possible to temporarily stop the input clock with little impact to the de-skew circuit, as long as CLKFX or CLKFX180 is not used.

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

CLKFX or CLKFX180 stops within D cycles of CLKIN or when CLKFX is concurrent to CLKIN. STATUS[2] is asserted within 1 to D CLKIN + 1 CLKFX cycles of CLKFX or when CLKFX180 output stops. (D is the CLKFX_DIVIDE value.)

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

When the input clock is being stopped (CLKIN remains High or Low for more than 1 clock cycle), one to eight more output clock cycles are still generated as the delay line is flushed. When the output clock stops, the CLKIN stopped (STATUS(1)) signal is asserted. When the clock is restarted, the output clock cycles are not generated for one to eight clocks while the delay line is filled. Similarly, the STATUS(1) signal is deasserted once the output clock is generated. The most common case is two or three clocks. In a similar manner, a phase shift of the input clock is possible. For example, the input clock can be stopped for 12 ns to achieve a 12 ns phase shift at the output. The phase shift propagates one to eight clocks to the output after the original shift, with no disruption to the DCM control. The STATUS(1) is forced Low whenever LOCKED is Low.

Output Clocks

Some restrictions apply regarding the connectivity of the output pins. The DCM clock outputs can each drive an OBUF, a global clock buffer BUFGMUX, or they can route directly to the clock input of a synchronous element. To use the dedicated routing, the DCM clock outputs should drive BUFGMUXs that are on the same edge (top or bottom) of the device. If the DCM and BUFGMUX are not on the same edge, local routing is used and the DCM might not deskew properly.

Do not use the DCM output clock signals until after activation of the LOCKED signal. Prior to the activation of the LOCKED signal, the DCM output clocks are not valid and can exhibit glitches, spikes, or other spurious movement.

External Feedback

To ensure consistent locking, if a DCM is configured with external feedback, applying a reset after configuration is strongly recommended.

For an optimum locking process, a DCM configured with external feedback requires both CLKIN and CLK0 to be present and stable when the DCM begins to lock. During configuration with external feedback, it is not possible to provide CLKFB in the beginning of the locking process. At the end of configuration, the DCM begins to lock once the device enters the startup sequence. Because a global 3-state signal (GTS) is still asserted during this time, the output I/O pins remain in a 3-state condition, effectively putting the CLKFB signal into a 3-state condition.

When CLKFB eventually appears (after the GTS is deasserted), the DCM proceeds with the lock. However, it might not lock at the optimal point and can introduce slightly more jitter (as well as greater clock cycle latency) through the DCM.

In addition, if CLKFB is coupling with another signal when it is put into a 3-state condition (a PCB signal integrity issue), the DCM might sense this invalid clock as CLKFB and use it to proceed with a lock. This second possibility can cause the DCM to not lock properly once the GTS deasserts and the true CLKFB signal is present.

Use of reset after configuration guarantees that the locking process starts with a valid CLKIN and CLKFB signal and ensures consistent locking.

Notes:

1. The default value is **`-g LCK_cycle:NoWait`** and **`-g GTS_cycle:5`**. When these settings are used, the startup sequence does not wait for the DCM to lock.
2. If the startup sequence is altered (for example, using the BitGen option), the **`LCK_cycle`** (wait for DCM to lock) must not be placed before the **`GTS_cycle`** (de-assert GTS); otherwise, the DCM never locks and configuration does not complete.
3. HDL example code to reset the DCM after configuration is shown below. This example requires that **`GTS_cycle`** be set before **`GWE_cycle`** in the BitGen options (the default). This ensures that the DCM is reset after the I/O pins are activated.

   Verilog Example:

```
// DCM instantiation to show the reset connection

                        DCM u_dcm1 (.CLKIN(clkin),
                        .CLKFB(clkfb),
                        .CLK0(clk0),
                        .RST(rstin),
                        .LOCKED(locked));

                        // rstin connects to RST pin of DCM
                        assign rstin = (user_reset || config_rst);

                       // This is the actual reset circuit that outputs
config_rst. It is a four-cycle shift register.

                    FDS flop1 (.D(1'b0), .C(clkin), .Q(out1), .S(1'b0));
                        FD flop2 (.D(out1), .C(clkin), .Q(out2));
                        FD flop3 (.D(out2), .C(clkin), .Q(out3));
                        FD flop4 (.D(out3), .C(clkin), .Q(out4));

                      //config_rst will be asserted for 3 clock cycles.
                        assign config_rst = (out2 | out3 | out4);
```

VHDL Example:

```
-- DCM instantiation to show the reset connection

                        u_dcm1: DCM port map (
                        CLKIN => clkin,
                        CLKFB => clkfb,
                        CLK0 => clk0,
                        RST => rstin,
                        LOCKED => locked);

                        -- rstin connects to RST pin of DCM
                        rstin <= user_reset or config_rst;

                       -- This is the actual reset circuit that outputs
config_rst. It is a four-cycle shift register.

                        flop1: FDS port map (D => '0', C => clkin, Q =>
out1, S => '0');
                  flop2: FD port map (D => out1, C => clkin, Q => out2);
                  flop3: FD port map (D => out2, C => clkin, Q => out3);
                  flop4: FD port map (D => out3, C => clkin, Q => out4);

                    -- config_rst will be asserted for 3 clock cycles.
                      config_rst <= out2 or out3 or out4;
```

Characteristics of the De-Skew Circuit

- Can eliminate clock distribution delay by effectively adding one clock period delay. Clocks are de-skewed to within CLKOUT_PHASE, specified in the **Virtex-II Data Sheet (DS031)**.
- Can be used to eliminate on-chip as well as off-chip clock delay.
- Has no restrictions on the delay in the feedback clock path.
- Requires a continuously running input clock.
- Adapts to a wide range of frequencies. However, once locked to a frequency, cannot tolerate large variations of the input frequency.
- De-skew circuit is part of the DCM, which also includes phase adjustment, frequency synthesis, and spread spectrum techniques that are described in this document.
- Does not eliminate jitter. The de-skew circuit output jitter is the sum of input jitter and some jitter value that the de-skew circuit might add.
- The completion of configuration can be delayed until after DCM locks to guarantee the system clock is established prior to initiating the device.

Port Signals

Source Clock Input — CLKIN

The CLKIN pin provides the user source clock (the clock signal on which the de-skew circuit operates) to the DCM. The CLKIN frequency must fall in the ranges specified in the **Virtex-II Data Sheet (DS031)**. The clock input signal can be provided by one of the following:

IBUF — Input buffer

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

BUFGMUX — Internal global clock buffer

Note: When IBUF is used as the clock input, the PAD to DCM input skew is not compensated. Refer to Table 3-1 for more information.

Feedback Clock Input — CLKFB

A reference or feedback signal is required to delay-compensate the output. Connect only the CLK0 or CLK2X DCM outputs to the feedback clock input (CLKFB) pin to provide the necessary feedback to the DCM. The feedback clock input signal can be driven by an internal global clock buffer (BUFGMUX), one of the global clock input buffers (IBUFG) on the same edge of the device (top or bottom), or IBUF (the input buffer.) Note that when IBUF is used, the PAD to DCM input skew is not compensated.

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

1. The CLK0 or CLK2X of the DCM must directly drive an OBUF or a BUFG-to-DDR configuration to forward the clock.
2. External to the FPGA, the same forwarded clock signal must be connected to the IBUFG (GCLK pin) that drives the CLKFB of the DCM.

Reset Input — RST

When the reset pin is activated, the LOCKED signal deactivates. The M and D values at configuration are maintained after the reset. The RST pin, active High, must be connected to a dynamic signal or tied to ground. For all designs that use the feedback feature of the DCM, applying a reset signal after configuration is strongly recommended for both production and ES devices in order to ensure consistent locking. As the DCM delay taps reset to zero, glitches can occur on the DCM clock output pins. After the DCM resets the clock, outputs have a DC behavior. Activation of the RST pin can also severely affect the duty cycle of the clock output pins. Furthermore, the DCM output clocks no longer de-skew with respect to one another and are eventually stopped Low if Status signals are deactivated (brought to Low).

For these reasons, use the reset pin when reconfiguring the device or changing the input frequency, or after reconfiguration for DCMs with external feedback. The reset input signal is asynchronous and should be held High for at least three clock cycles. The time it takes for the DCM to achieve lock is specified as LOCK_DLL (for DLL output) and LOCK_FX (for DFS output). The DCM locks faster at higher frequencies. See the LOCK_DLL timing parameter in the **Virtex-II Data Sheet (DS031)**.

Locked Output — LOCKED

In order to achieve lock, the DCM might need to sample several thousand clock cycles. After the DCM achieves lock, the LOCKED signal goes High. The DCM timing parameters section of the **Virtex-II Data Sheet (DS031)** provides estimates for locking times.

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

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

Status - STATUS

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

Attributes

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

Frequency Mode

The de-skew feature of the DCM is achieved with a delay-locked loop (DLL). This attribute specifies either the high or low-frequency mode of the DLL. The default is low-frequency mode. In high-frequency mode, the only outputs available from the DLL are the CLK0, CLK180, CLKDV, and LOCKED. (CLK90, CLK270, CLK2X, and CLK2X180 are not available in high-frequency mode.) The frequency ranges for both frequency modes are specified in the **Virtex-II Data Sheet (DS031)**. To set the DLL to high-frequency mode, attach the DLL_FREQUENCY_MODE=HIGH attribute in the source code or schematic.

Feedback Input

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

Duty Cycle Correction

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

Setting DUTY_CYCLE_CORRECTION=FALSE dectivates duty cycle correction for the DCM outputs listed above. It is strongly recommended to always set the DUTY_CYCLE_CORRECTION attribute to TRUE (default value). Note that setting this attribute to FALSE does not necessarily produce output clocks with the same duty cycle as the source clock.

Startup Delay

The default value of the STARTUP_WAIT attribute is FALSE. When STARTUP_WAIT is set to TRUE, and the LCK_cycle BitGen option is used, then the configuration startup sequence waits in the specified cycle until the DCM locks. For details, see Chapter 4: Configuration and Appendix A: BitGen and PROMGen Switches and Options.

Legacy Support

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

- CLKDLL
- CLKDLLE
- CLKDLLHF
- BUFGDLL

Library Primitive

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

Submodules

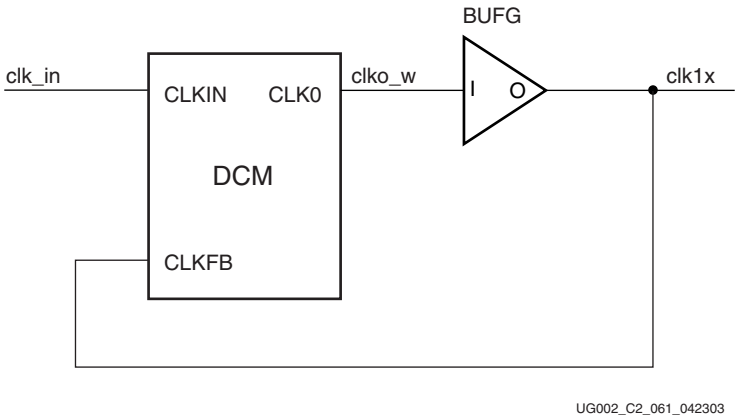


Figure 3-24: **BUFG_CLK0_SUBM**

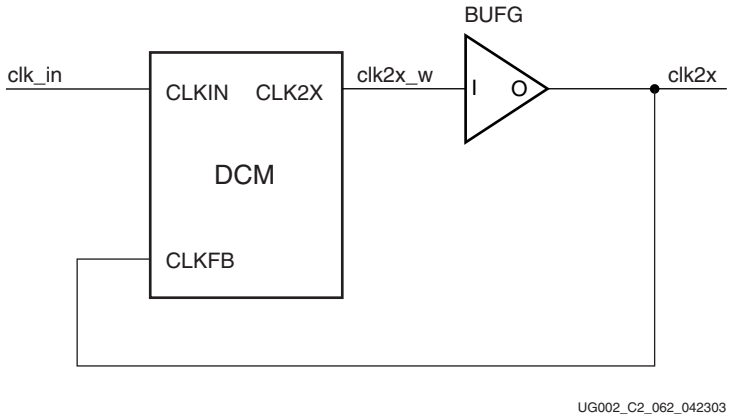


Figure 3-25: **BUFG_CLK2X_SUBM**

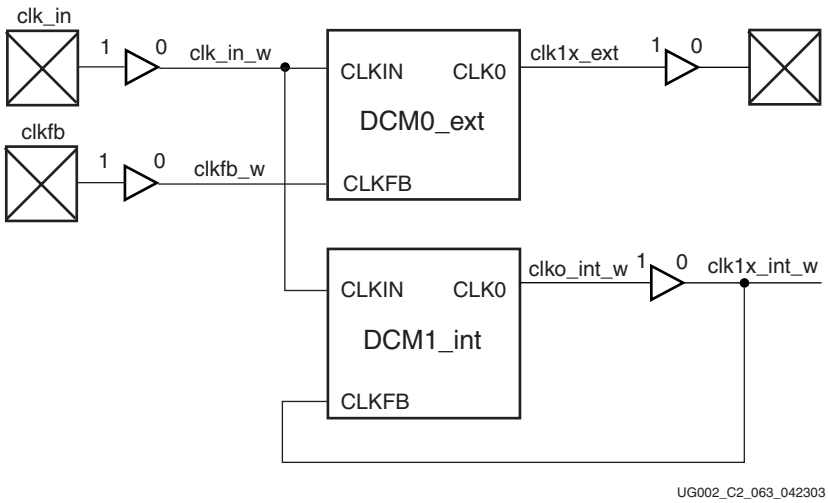


Figure 3-26: **BUFG_CLK0_FB_SUBM**

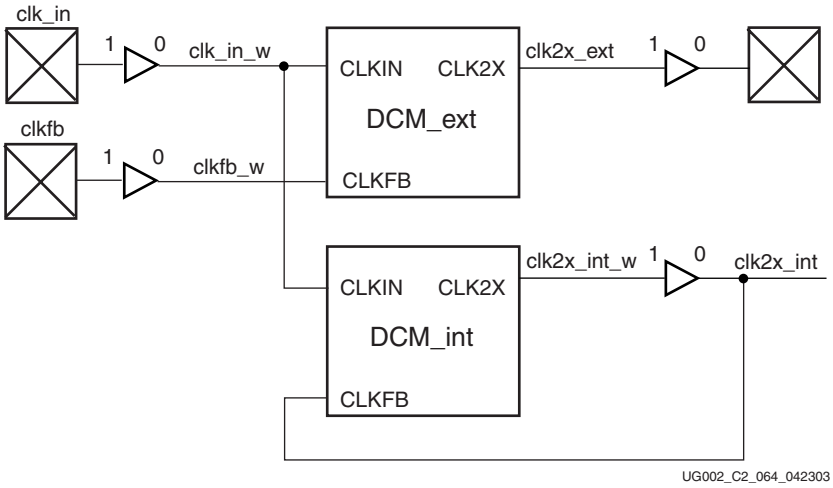


Figure 3-27: **BUFG_CLK2X_FB_SUBM**

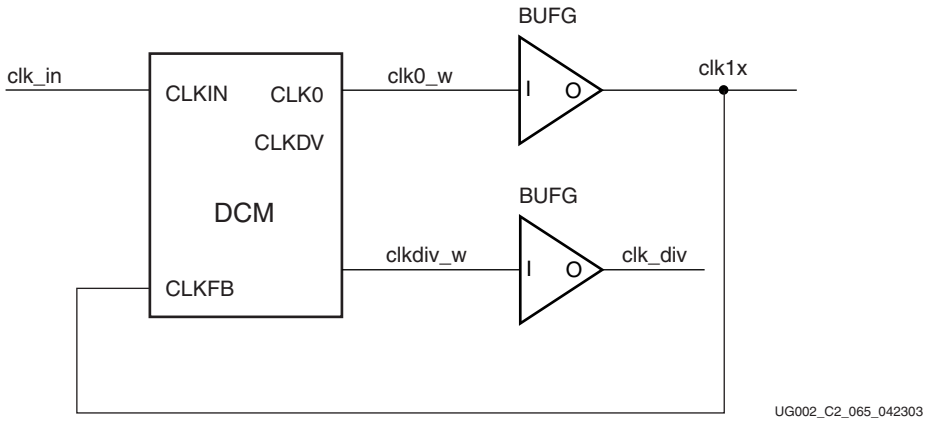


Figure 3-28: **BUFG_CLKDV_SUBM**

Frequency Synthesis

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

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

The two counter-phase frequency synthesized outputs can drive global clock routing networks within the device. The well-buffered global clock distribution network minimizes clock skew due to differences in distance or loading. See Figure 3-29.

Operation

The DCM clock output CLKFX is any M/D product of the clock input to the DCM. Specifications for M and D, as well as input and output frequency ranges for the frequency synthesizer, are provided in the **Virtex-II Data Sheet (DS031)**. The frequency synthesizer

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

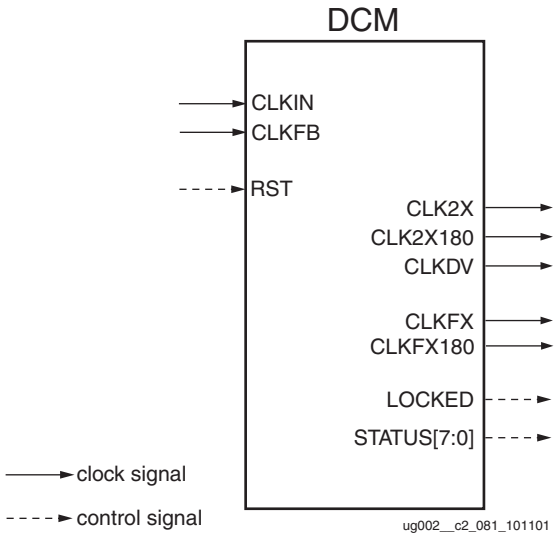


Figure 3-29: **Frequency Synthesis Outputs**

The internal operation of the frequency synthesizer is complex and beyond the scope of this document. The frequency synthesizer multiplies the incoming frequencies by the pre-calculated quotient M/D and generates the correct output frequencies as long as it is within the range specified in the **Virtex-II Data Sheet (DS031)**.

For example, assume input frequency = 50 MHz, M = 25, and D = 8 (note that M and D values have no common factors and hence cannot be reduced). The output frequency is correctly 156.25 MHz, although 25 x 50 MHz = 1.25 GHz and 50 MHz / 8 = 6.25 MHz, and both of these values are far outside the range of the input frequency.

Frequency Synthesizer Characteristics

- The frequency synthesizer provides an output frequency equal to the input frequency multiplied by M and divided by D.
- The outputs CLKFX and CLKFX180 always have a 50/50 duty-cycle.
- Smaller M and D values achieve faster lock times. The user should divide M and D by the largest common factor.
- The outputs are phase aligned with CLK0 when CLKFB is connected.

Port Signals

Source Clock Input — CLKIN

The CLKIN pin provides the user source clock to the DCM. The CLKIN frequency must fall in the ranges specified in the **Virtex-II Data Sheet (DS031)**. The clock input signal can be provided by one of the following:

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

Note: When IBUF is used as the clock input, the PAD to DCM input skew is not compensated. Refer to Table 3-1 for more information.

2x Clock Output — CLK2X

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

Until the DCM has achieved lock, the CLK2X output appears as a 1x version of the input clock with a 25/75 duty cycle. This behavior allows the DCM to lock on the correct edge with respect to source clock.

Clock Divide Output — CLKDV

The clock divide output pin CLKDV provides a lower frequency version of the source clock. The CLKDV_DIVIDE property controls CLKDV such that the source clock is divided by N where N is either 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 9, 10, 11, 12, 13, 14, 15, or 16.

This feature provides automatic duty cycle correction such that the CLKDV output pin has a 50/50 duty cycle always in low-frequency mode, as well as for all all integer values of the division factor N in high-frequency mode.

Frequency Synthesized Clock Output - CLKFX

The CLKFX output provides a frequency-synthesized clock (M/D * CLKIN) with a 50/50 duty cycle. For the CLKFX output to be phase-aligned with CLKIN, the clock feedback (CLK0) must be provided at the CLKFB input. With M and D adjusted such that they have no common factor, the alignment occurs only once every D input clock cycles. There is no CLKFX phase alignment when the CLKIN frequency is below 24 Mhz.

Frequency Synthesized Clock Output 180º Phase Shifted - CLKFX180

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

Locked Output — LOCKED

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

Reset Input — RST

When the reset pin is activated, the LOCKED signal deactivates. The M and D values at configuration are maintained after the reset. The RST pin, active High, must be connected to a dynamic signal or tied to ground. For all designs that use the feedback feature of the DCM, applying a reset signal after configuration is strongly recommended for both production and ES devices in order to ensure consistent locking. As the DCM delay taps reset to zero, glitches can occur on the DCM clock output pins. After the DCM resets the clock, outputs have a DC behavior. Activation of the RST pin can also severely affect the duty cycle of the clock output pins. Furthermore, the DCM output clocks no longer de-skew with respect to one another and are eventually stopped Low if Status signals are deactivated (brought to Low).

For these reasons, use the reset pin when reconfiguring the device or changing the input frequency, or after reconfiguration for DCMs with external feedback. The reset input signal is asynchronous and should be held High for at least three clock cycles. The time it takes for the DCM to achieve lock is specified as LOCK_DLL (for DLL output) and LOCK_FX (for DFS output). The DCM locks faster at higher frequencies. See the LOCK_DLL timing parameter in the **Virtex-II Data Sheet (DS031)**.

Status - STATUS

The STATUS output is an 8-bit output:

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

Attributes

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

Clock Divide

The CLKDV_DIVIDE attribute specifies how the signal on the CLKDV pin is frequency divided with respect to the CLK0 pin. The values allowed for this attribute are 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 9, 10, 11, 12, 13, 14, 15, or 16; the default value is 2.

Frequency Mode for Frequency Synthesis

This attribute specifies either the high or low-frequency mode of the frequency synthesizer. The default is low-frequency mode. The frequency ranges for both frequency modes are specified in the **Virtex-II Data Sheet (DS031)**.

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

Multiply/Divide Attribute

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

Startup Delay

The default value of the STARTUP_WAIT attribute is FALSE. When STARTUP_WAIT is set to TRUE, and the LCK_cycle BitGen option is used, then the configuration startup sequence waits in the specified cycle until the DCM locks. For details, see Chapter 4: Configuration and Appendix A: BitGen and PROMGen Switches and Options.

CLKIN_PERIOD

CLKIN_PERIOD specifies the period of the clock used to drive the CLKIN pin of the DCM. It must be specified for optimal frequency synthesis operation when using CLKFX or CLKFX180 outputs. It is not needed for other DCM clock outputs.

Submodules

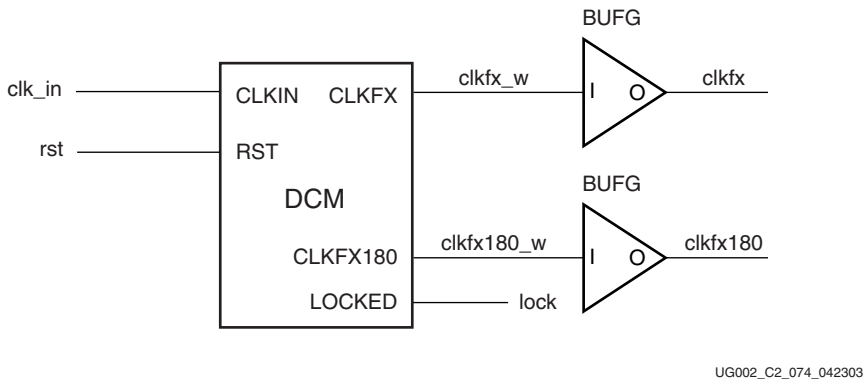


Figure 3-30: **BUFG_DFS_SUBM**

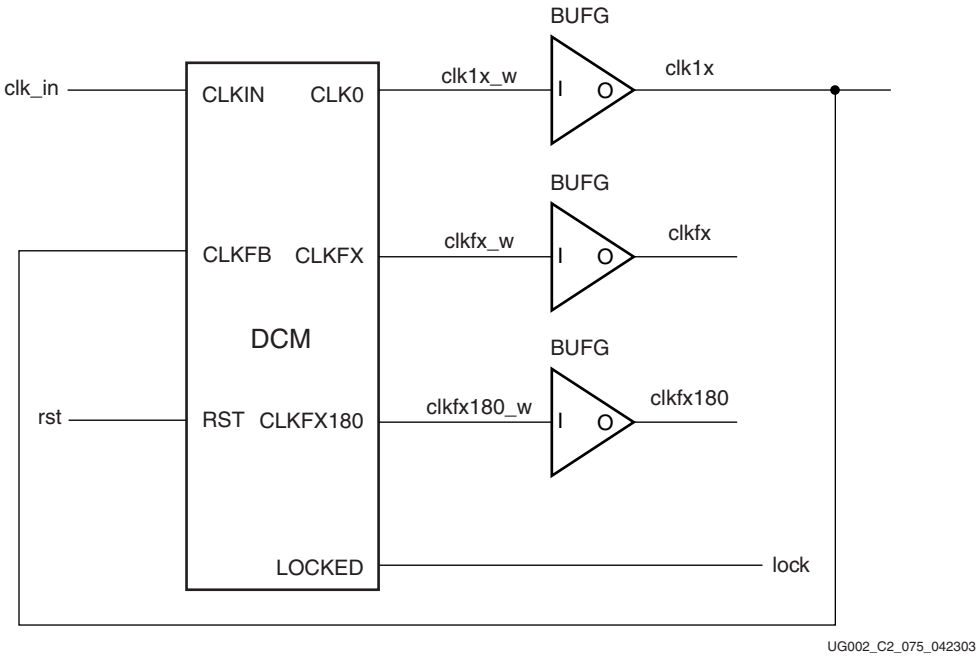


Figure 3-31: **BUFG_DFS_FB_SUBM**

Phase Shifting

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

Operation

Figure 3-32 shows a block diagram of the DCM and all of the outputs affected by the circuitry of the phase shift feature.

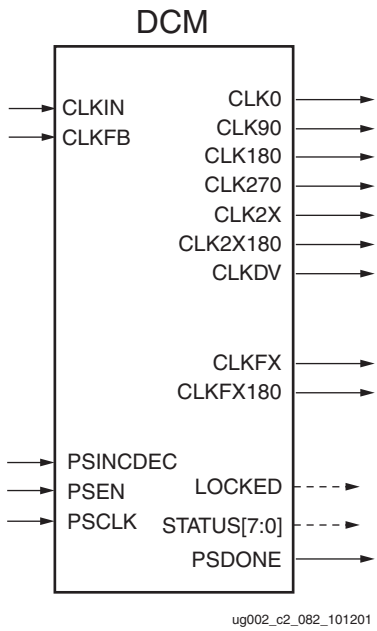


Figure 3-32: **Phase Shift Outputs**

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

In variable mode, the PHASE_SHIFT value can also be dynamically incremented or decremented as determined by PSINCDEC synchronously to PSCLK, when the PSEN input is active. Figure 3-33 illustrates the effects of fine-phase shifting.

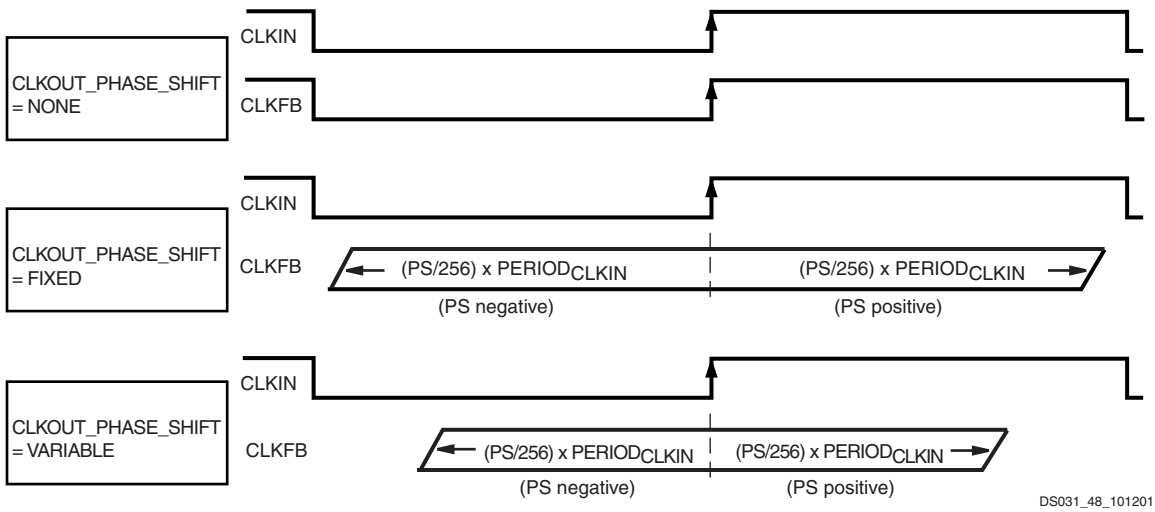


Figure 3-33: **Phase Shift Effects**

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

- PHASE_SHIFT attribute range
- FINE_SHIFT_RANGE DCM timing parameter range

The PHASE_SHIFT attribute is the numerator in the following equation:

Phase Shift (ns) = (PHASE_SHIFT/256) * $PERIOD_{CLKIN}$

The full range of this attribute is always -255 to +255, but its practical range varies with CLKIN frequency, as constrained by the FINE_SHIFT_RANGE component, which represents the total delay achievable by the phase shift delay line. Total delay is a function of the number of delay taps used in the circuit. Across process, voltage, and temperature, this absolute range is guaranteed to be as specified in the DCM Timing Parameters section of the **Virtex-II Data Sheet (DS031)**.

Absolute range (fixed mode) = ± FINE_SHIFT_RANGE

Absolute range (variable mode) = ± FINE_SHIFT_RANGE/2

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

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

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

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

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

Phase Shift Characteristics

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

The minimum granularity is the greater of two limiting factors:

1. The minimum phase shift step size = $1/256$ * CLKIN_Period.
2. The tap delay resolution (DCM_TAP). The specifications for DCM_TAP_MIN and DCM_TAP_MAX are available in the datasheet

Therefore, assuming a typical DCM_TAP of 45 ps, at ~90 Mhz with an 11.11 ns period, the $1/256$ * CLKIN_Period is ~45 ps.

- At a frequency < approx. 90 MHz, $1/256$ * CLKIN_Period > 45 ps; hence the minimum step size is limited by #1 above.
- At a frequency > approx. 90 MHz, $1/256$ * CLKIN_Period < 45 ps; hence the minimum step size is limited by #2 above.

At any condition, regardless of the CLKIN frequency, the PHASE_SHIFT (PS) value is determined by the following equation:

desired phase shift" = (PS/256) * (CLKIN period)

From this PS value, the DPS function of DCM will select the appropriate tap setting. This tap setting will vary across process, voltage, and temperature (PVT).

For example, if CLKIN = 150 Mhz (6.67 ns period), and the "desired phase shift" = 680 ps:

0.680 = PS/256 * 6.67
PS = 26

(This will be the PHASE_SHIFT value.)

The DCM_TAP of ~45 ps is only an approximation. The actual tap value can range from 30 ps to 60 ps at any given time, depending upon PVT. Hence, 10 tap steps does not always mean 10 * 45 ps = 450 ps. In addition, the DCM might not adjust to a new phase shift at every $^{1}/_{256}$ phase shift increment if $^{1}/_{256}$ * CLK_Period < DCM_TAP, because $^{1}/_{256}$ is not equal to 1 tap.

For example, when you change the PS value from "3" to "4", the DCM may not increment the phase shift if the current phase shift is closer to the ideal value when compared to the next tap at the PVT.

In this case, the design consideration is that at any time, the DCM will find the closest phase shift given the PHASE_SHIFT attribute value. The error at this PHASE_SHIFT for a simple case of using CLK0 will be CLKOUT_PER_JITT_0 + CLKIN_CLKFB_PHASE. This is specified in the datasheet.

The DCM_TAP value is included in the output jitter specs along with random jitter introduced by the DCM and by the FPGA. Using a FIXED phase shift will allow the design to be within 5-10% of the desired phase shift point. If more accuracy is needed, use the VARIABLE mode to dial it in.

Fixed-Mode Phase Shifting

The use of negative, fixed-mode phase shifting with a Virtex-II DCM requires a minor modification to the clock muxing. Specifically, CLKIN must be used to drive CLKFB while the DCM is in reset (when the RST signal is held High or during the startup sequence after configuration). Figure 3-34 and the example Verilog and VHDL code below illustrate the implementation of this simple modification. This modification is NOT required for any positive phase shifting or for any variable-mode phase shifting.

Note that this requirement applies to any DCM using fixed-mode phase shifting, regardless of whether the RST of the DCM is connected to a user signal. RST must be held High for at least three clock cycles.

Solution 1: Clock-Muxing Modification

Figure 3-34 illustrates the clock-muxing modification.

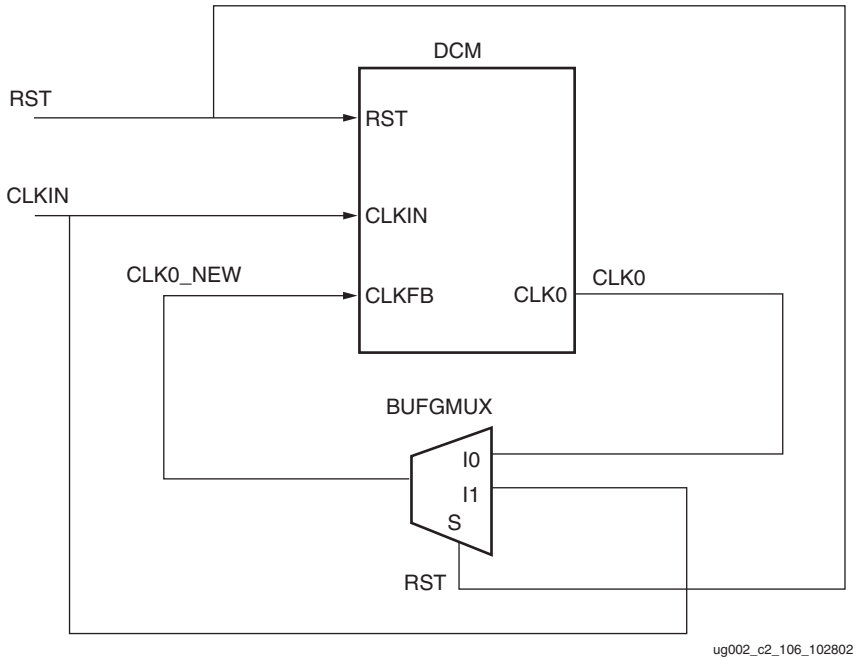


Figure 3-34: **Clock Muxing for Negative PHASE_SHIFT Value**

Note: PAR might report a warning that the IBUFG (CLKIN)-to-BUFGMUX connection is not an optimal connection and so does not use the fast connection between the two components. This warning can be safely ignored; in this case, a dedicated connection is not necessary for the IBUFG-BUFGMUX connection.

Verilog Example:

```
//DCM instantiation
DCM U_DCM ( .CLKIN(clkin), .CLKFB(clk0_new), .RST(rst), .CLK0(clk0),
.LOCKED(locked) );

//BUFGMUX instantiation
BUFGMUX U_BUFGMUX ( .O(clk0_new), .I0(clk0), .I1(clkin), .S(rst) );
```

VHDL Example:

```
-- DCM instantiation --
U_DCM: DCM port map ( CLKIN => clkin,
CLKFB => clk0_new,
DSSEN => '0',
PSCLK => '0',
PSINCDEC => '0',
RST => rst,
CLK0 => clk0,
LOCKED => locked);

-- BUFGMUX instantiation --
U_BUFGMUX: BUFGMUX( I0 => clk0
I1 => clkin,
O => clk0_new);
```

Solution 2: Alternative Workaround Using Positive Phase Shift

This problem can also be resolved by using an equivalent positive PHASE_SHIFT value. Given the current data sheet specification of FINE_SHIFT_RANGE = 10 ns, this restriction begins to have an effect for input frequencies below 100 MHz (input periods larger than 10 ns).

Given the equation:

Phase shift = (PS/256) * PeriodCLKIN

For a 10 ns input clock period:

Variable mode: -128 <= PS <= 128

Fixed mode: 0 <= PS <= 255 (full range)

For a 20 ns input clock period:

Variable mode: -64 <= PS <= 64

Fixed mode: 0 <= PS <= 128

For a 40 ns input clock period:

Variable mode: -32 <= PS <= 32

Fixed mode: 0 <= PS <= 64

In either fixed or variable mode, the range can be extended by choosing CLK90, CLK180, or CLK270, rather than CLK0, choosing CLK2X180 rather than CLK2X, or choosing CLKFX180 rather than CLKFX. Even at 25 MHz (40 ns period), the fixed mode coupled with these CLK* phases allows shifting throughout the entire input clock period range. See Figure 3-35.

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

Because CLKFB must be from CLK0 or CLK2X, the DLL output is used. Hence the minimum CLKIN frequency for the DPS function is 24 MHz.

Solution 3: Alternative Workaround Using Variable Phase Shift

Configure the DCM with variable phase shifting, as follows:

1. Set the CLK_OUT_PHASE_SHIFT attribute to VARIABLE.
2. Set the PHASE_SHIFT attribute to the required value (a negative value is acceptable).
3. Connect PSCLK to a clock signal.
4. Connect PSEN to Ground if incrementing or decrementing the phase shift is not intended.
5. Connect PSINCDEC to a signal, GND, or VCC.

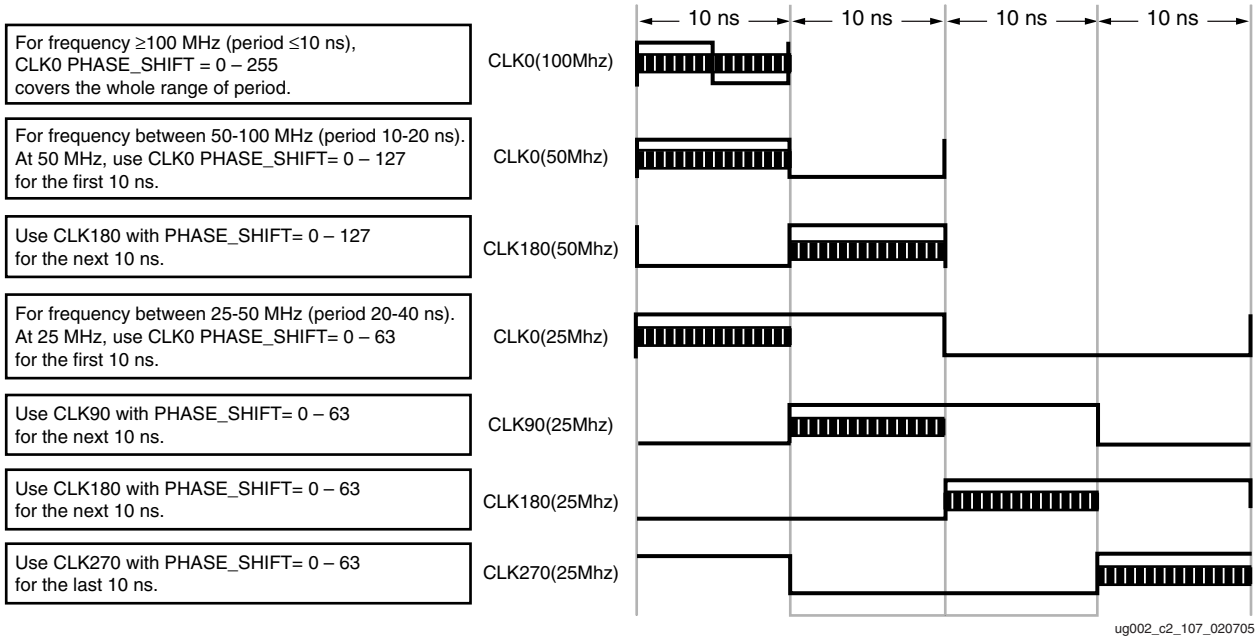


Figure 3-35: **Fixed Phase Shift Example**

Port Signals

1x Clock Outputs — CLK[0|90|180|270]

The 1x clock output pin CLK0 represents a delay-compensated version of the source clock (CLKIN) signal. In low-frequency mode, the DCM provides three phase-shifted versions of the CLK0 signal (CLK90, CLK180, and CLK270), whereas in high-frequency mode, only the 180 phase-shifted version is provided. All four (including CLK0) of the phase shifted outputs can be used simultaneously in low-frequency mode. The relationship between phase shift and the corresponding period shift appears in Table 3-8. The timing diagrams in Figure 3-36 illustrate the DLL clock output characteristics.

Table 3-8: **Relationship of Phase-Shifted Output Clock to Period Shift**

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

By default, the DCM provides a 50/50 duty cycle correction on all 1x clock outputs. The DUTY_CYCLE_CORRECTION attribute (TRUE by default), controls this feature. Attach the DUTY_CYCLE_CORRECTION=FALSE property to the DCM symbol in order to deac-

tivate the DCM duty cycle correction. With duty cycle correction deactivated, the output clocks have the same duty cycle as the source clock.

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

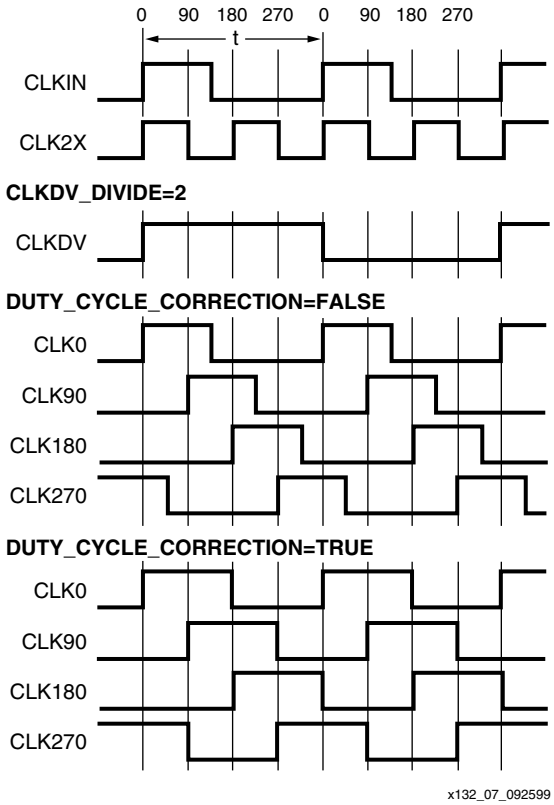


Figure 3-36: **DLL Output Characteristics**

Source Clock Input — CLKIN

The CLKIN pin provides the user source clock to the DCM. The CLKIN frequency must fall in the ranges specified in the **Virtex-II Data Sheet (DS031)**. The clock input signal can be provided by one of the following:

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

  Note: When IBUF is used as the clock input, the PAD to DCM input skew is not compensated. Refer to Table 3-1 for more information.

Feedback Clock Input — CLKFB

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

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

1. The CLK0 or CLK2X of the DCM must directly drive an OBUF or a BUFG-to-DDR configuration to forward the clock.
2. External to the FPGA, the same forwarded clock signal must be connected to the IBUFG (GCLK pin) that drives the CLKFB of the DCM.

Phase Shift Clock - PSCLK

The PSCLK input can be sourced by the CLKIN signal to the DCM, or it can be a lower or higher frequency signal provided from any clock source (external or internal). The frequency range of PSCLK is defined by PSCLK_FREQ_LF/HF (see the **Virtex-II Data Sheet (DS031)**). This input has to be tied to ground when the CLKOUT_PHASE_SHIFT attribute is set to NONE or FIXED.

Phase Shift Increment/Decrement - PSINCDEC

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

Phase Shift Enable - PSEN

To initiate a variable phase-shift operation, the PSEN input must be activated for one period of PSCLK. The phase change becomes effective after up to 100 CLKIN pulse cycles plus three PSCLK cycles, and is indicated by a High pulse on PSDONE. During the phase transition there are no sporadic changes or glitches on any output. From the time when PSEN is enabled until PSDONE is flagged, the DCM output clock will slide bit-by-bit from its original phase shift to the incremented phase shift. The increment/decrement will be ready by PSDONE. PSEN must be tied to ground when the CLKOUT_PHASE_SHIFT attribute is set to NONE or FIXED.

Reset Input — RST

When the reset pin is activated, the LOCKED signal deactivates. The M and D values at configuration are maintained after the reset. The RST pin, active High, must be connected to a dynamic signal or tied to ground. For all designs that use the feedback feature of the DCM, applying a reset signal after configuration is strongly recommended for both production and ES devices in order to ensure consistent locking. As the DCM delay taps reset to zero, glitches can occur on the DCM clock output pins. After the DCM resets the clock, outputs have a DC behavior. Activation of the RST pin can also severely affect the duty cycle of the clock output pins. Furthermore, the DCM output clocks no longer de-skew with respect to one another and are eventually stopped Low if Status signals are deactivated (brought to Low).

For these reasons, use the reset pin when reconfiguring the device or changing the input frequency, or after reconfiguration for DCMs with external feedback. The reset input signal is asynchronous and should be held High for at least three clock cycles. The time it takes for the DCM to achieve lock is specified as LOCK_DLL (for DLL output) and LOCK_FX (for DFS output). The DCM locks faster at higher frequencies. See the LOCK_DLL timing parameter in the **Virtex-II Data Sheet (DS031)**.

Locked Output — LOCKED

The LOCKED signal activates after the DCM has achieved lock. To guarantee that the system clock is established prior to the device "waking up," the DCM can delay the completion of the device configuration process until after the DCM locks. The STARTUP_WAIT attribute activates this feature. Until the LOCKED signal activates, the DCM output clocks are not valid and can exhibit glitches, spikes, or other spurious movement. For details, refer to Chapter 4: Configuration.

Phase Shift DONE - PSDONE

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

Status - STATUS

STATUS[0] indicates the overflow of the phase shift numerator (when DCM is phase shifted beyond +255 or -255) and that the absolute delay range of the phase shift delay line is exceeded.

Attributes

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

Clock Out Phase Shift

The CLKOUT_PHASE_SHIFT attribute controls the use of the PHASE_SHIFT value. It can be set to NONE, FIXED, or VARIABLE. By default, this attribute is set to NONE, indicating that the phase shift feature is not being used. When this attribute is set to NONE, the PHASE_SHIFT value has no effect on the DCM outputs. If the CLKOUT_PHASE_SHIFT attribute is set to FIXED or NONE, then the PSEN, PSINCDEC, and the PSCLK inputs must be tied to ground. The effects of the CLKOUT_PHASE_SHIFT attribute are shown in Figure 3-33.

PHASE_SHIFT

This attribute specifies the phase shift numerator as any value from -255 to 255. This attribute can be used with both fixed and variable phase shift mode. If used with variable mode, the attribute sets the starting phase shift.

Submodules

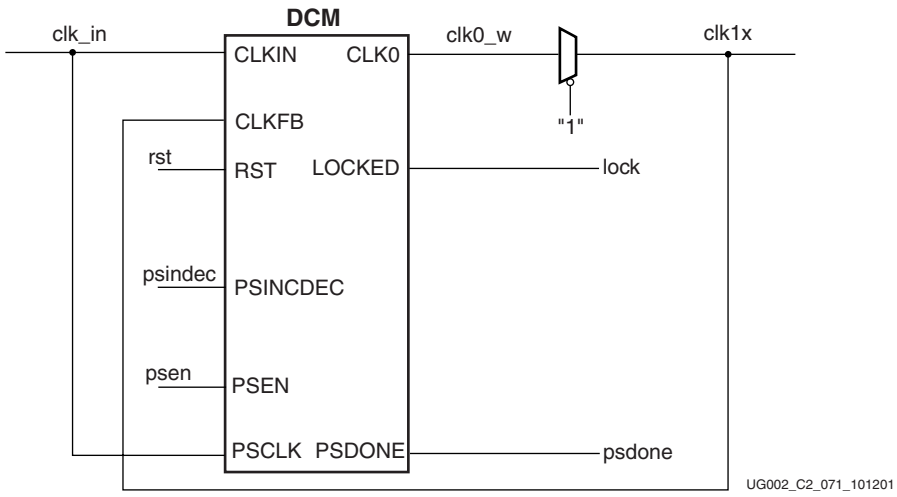


Figure 3-37: **BUFG_PHASE_CLK0_SUBM**

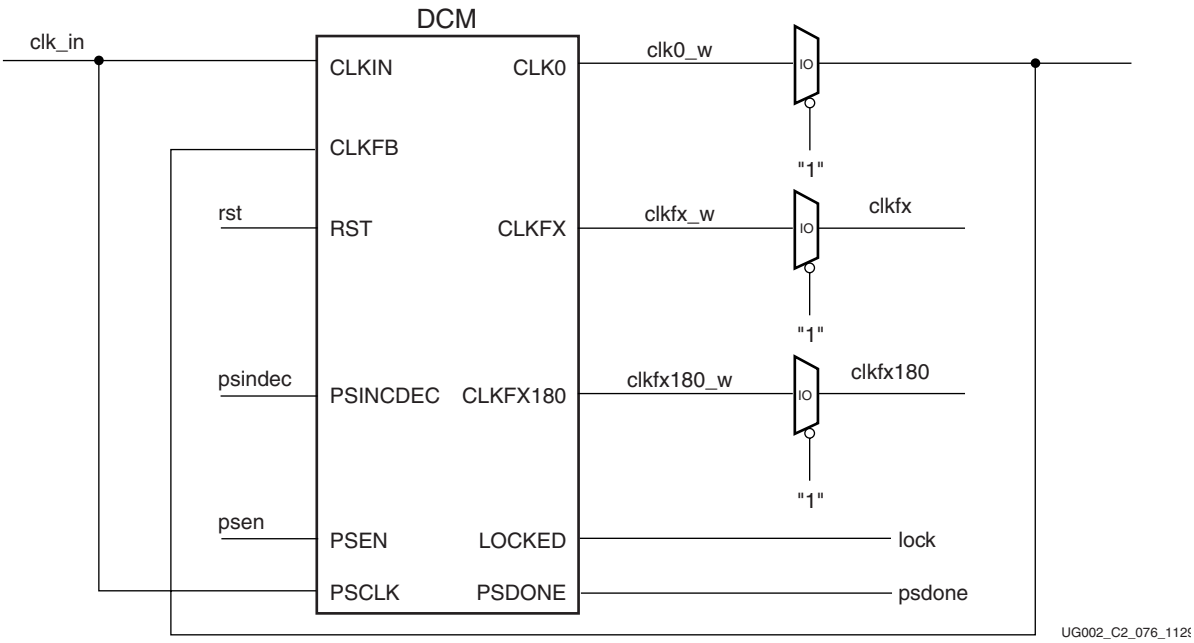


Figure 3-38: **BUFG_PHASE_CLKFX_FB_SUBM**

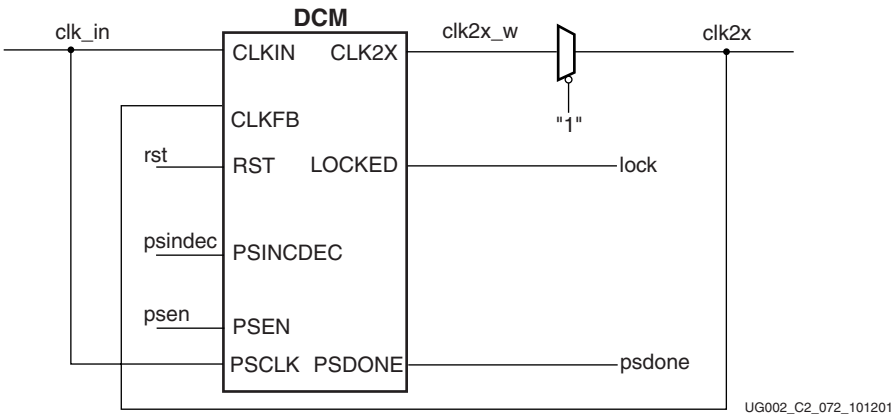


Figure 3-39: **BUFG_PHASE_CLK2X_SUBM**

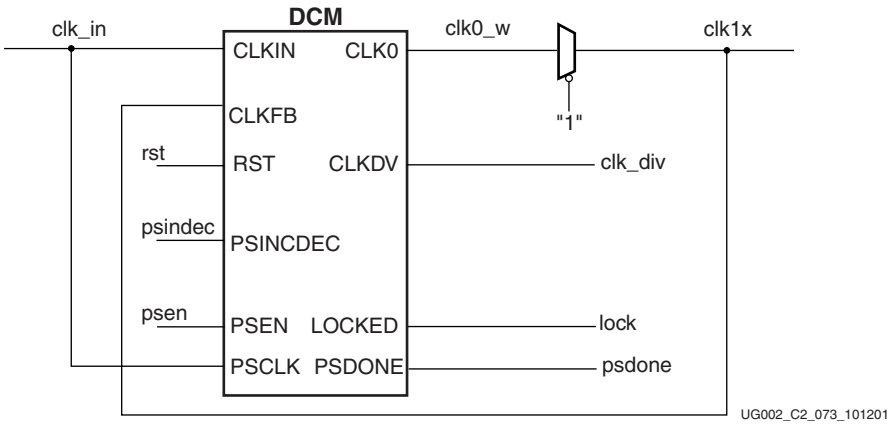


Figure 3-40: **BUFG_PHASE_CLKDV_SUBM**

Summary of All DCM Attributes

A handful of DCM attributes govern the functionality of DCM features. Below is a collection of all DCM attributes. Table 3-9 summarizes all attributes applicable to DCMs.

Table 3-9: **DCM Attributes**

DCM Attribute Name	Description	Value	Default Value
CLKDV_DIVIDE	This attribute controls CLKDV such that the source clock is divided by N. This feature provides automatic duty cycle correction such that the CLKDV output pin has a 50/50 duty cycle always in low-frequency mode, as well as for all integer values of the division factor N in high-frequency mode.	Real: 1.5 <= N+0.5 <=8, 8< N+1 < 16	2.0
CLKFX_DIVIDE		Integer: 1 to 32	1
CLKFX_MULTIPLY		Integer: 2 to 32	4
CLKIN_PERIOD	This specifies the source clock period to help DCM adjust for optimum CLKFX/CLKFX180 outputs.	Real in ns	0.0
CLKIN_DIVIDE_BY_2	This attribute allows for the input clock frequency to be divided in half when such a reduction is necessary to meet the DCM input clock frequency requirements.	Boolean: FALSE or TRUE	FALSE
CLKOUT_PHASE_SHIFT	This controls the use of the PHASE_SHIFT value.	String: “NONE” or ”FIXED” or ”VARIABLE”	NONE
CLK_FEEDBACK	This attribute specifies the feedback input to the DCM (CLK0, or CLK2X).	String: “1X” or ”2X”	1X
DESKEW_ADJUST	This affects the amount of delay in the feedback path, and should be used for source-synchronous interfaces. [See the Clock De-Skew section.]	String: “SYSTEM_SYNCHRONOUS” or “SOURCE_SYNCHRONOUS” or “0 to 15”	SYSTEM_SYNCHRONOUS
DFS_FREQUENCY_MODE	This specifies the frequency mode of the frequency synthesizer.	String: “LOW” or ”HIGH”	LOW
DLL_FREQUENCY_MODE	This specifies the DLL’s frequency mode. CLK90, CLK270, CLK2X, and CLK2X180 are not available in high-frequency mode.	String: “LOW” or ”HIGH”	LOW

Table 3-9: **DCM Attributes**

DCM Attribute Name	Description	Value	Default Value
DUTY_CYCLE_CORRECTION	This controls the DCM of 1X outputs (CLK0, CLK90, CLK180, and CLK270), such that they exhibit a 50/50 duty cycle. Leave this attribute set at the default value (TRUE).	Boolean: TRUE or FALSE	TRUE
DSS_MODE	Unsupported feature. Leave the value at default.	String	NONE
FACTORY_JF	This attribute is fixed unless recommended otherwise.	Bit_vector	XC080
PHASE_SHIFT	This specifies the phase shift numerator.	Integer: -255 to 255	0
STARTUP_WAIT	When this attribute is set to TRUE, the configuration startup sequence waits in the specified cycle until the DCM locks.	Boolean: FALSE or TRUE	FALSE

For more information on applying these attributes in UCF, VHDL, or Verilog code, refer to the *Constraints Guide* at: **http://toolbox.xilinx.com/docsan/xilinx4/manuals.htm**.

VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available as examples (see "VHDL and Verilog Templates" on page 105) for all submodules.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

VHDL and Verilog Templates

The following submodules described in this section are available:

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

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

VHDL Template

```
-- Module: BUFG_CLK0_SUBM
-- Description: VHDL submodule
-- DCM with CLK0 deskew
-- Device: Virtex-II Family
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
--
entity BUFG_CLK0_SUBM is
  port (
        CLK_IN : in std_logic;
        RST    : in std_logic;
        CLK1X  : out std_logic;
        LOCK   : out std_logic
       );
end BUFG_CLK0_SUBM;
--
architecture BUFG_CLK0_SUBM_arch of BUFG_CLK0_SUBM is
-- Components Declarations:
component BUFG
  port (
        I : in std_logic;
        O : out std_logic
      );
end component;
component DCM
-- pragma translate_off
    generic (
              DLL_FREQUENCY_MODE : string := "LOW";
              DUTY_CYCLE_CORRECTION : boolean := TRUE;
              STARTUP_WAIT : boolean := FALSE
             );
-- pragma translate_on
    port ( CLKIN     : in  std_logic;
           CLKFB     : in  std_logic;
           DSSEN     : in  std_logic;
           PSINCDEC  : in  std_logic;
           PSEN      : in  std_logic;
           PSCLK     : in  std_logic;
           RST       : in  std_logic;
           CLK0      : out std_logic;
           CLK90     : out std_logic;
           CLK180    : out std_logic;
           CLK270    : out std_logic;
           CLK2X     : out std_logic;
           CLK2X180  : out std_logic;
           CLKDV     : out std_logic;
           CLKFX     : out std_logic;
           CLKFX180  : out std_logic;
           LOCKED    : out std_logic;
           PSDONE    : out std_logic;
           STATUS    : out std_logic_vector(7 downto 0)
         );
end component;
-- Attributes
attribute DLL_FREQUENCY_MODE : string;
```

```
attribute DUTY_CYCLE_CORRECTION : string;
attribute STARTUP_WAIT : string;
attribute DLL_FREQUENCY_MODE of U_DCM: label is "LOW";
attribute DUTY_CYCLE_CORRECTION of U_DCM: label is "TRUE";
attribute STARTUP_WAIT of U_DCM: label is "FALSE";
-- Signal Declarations:
signal GND : std_logic;
signal CLK0_W: std_logic;
signal CLK1X_W: std_logic;
begin
GND <= '0';
CLK1X <= CLK1X_W;
-- DCM Instantiation
U_DCM: DCM
  port map (
            CLKIN =>    CLK_IN,
            CLKFB =>    CLK1X_W,
            DSSEN =>    GND,
            PSINCDEC => GND,
            PSEN =>     GND,
            PSCLK =>    GND,
            RST =>      RST,
            CLK0 =>     CLK0_W,
            LOCKED =>   LOCK
     );
-- BUFG Instantiation
U_BUFG: BUFG
  port map  (
       I => CLK0_W,
       O => CLK1X_W
            );
end BUFG_CLK0_SUBM_arch;
```

Verilog Template

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

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

    input CLK_IN;
    input RST;

    output CLK1X;
    output LOCK;

    wire CLK0_W;
    wire GND;

    assign GND = 1'b0;
```

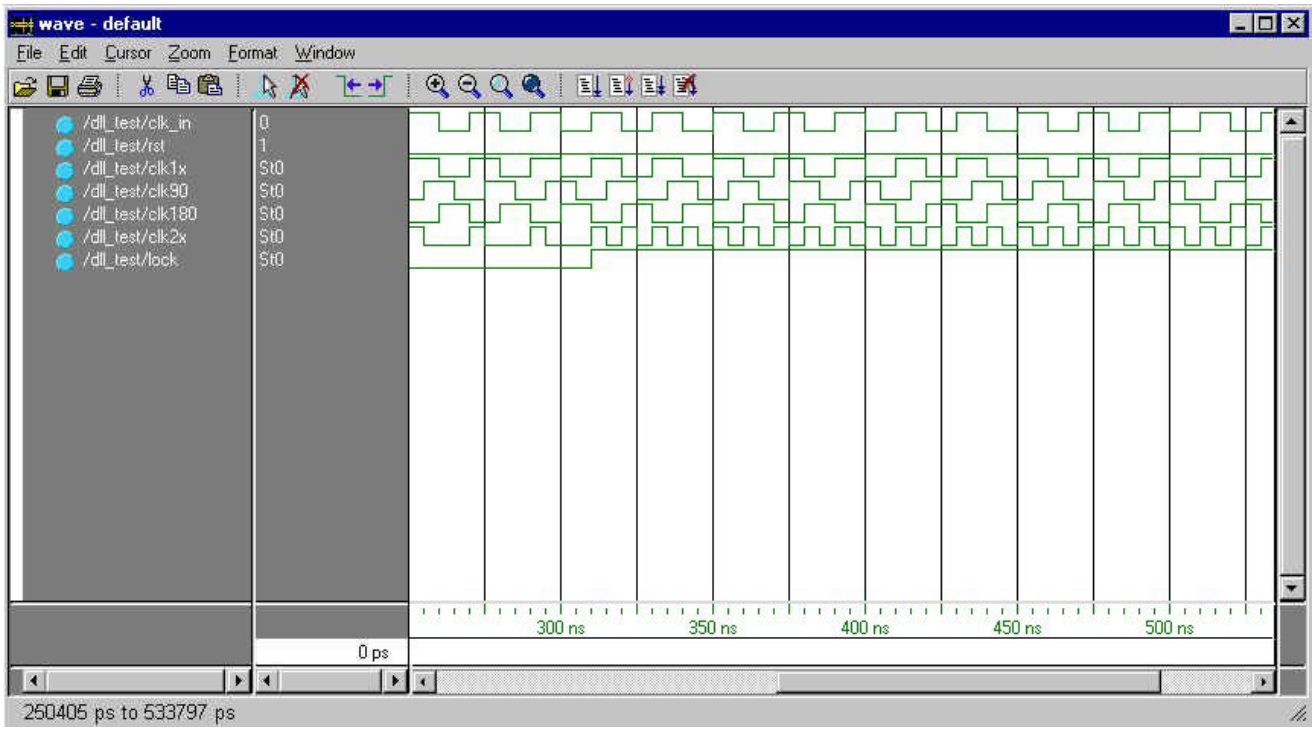
```
//BUFG Instantiation
//
BUFG  U_BUFG
            (.I(CLK0_W),
             .O(CLK1X)
            );

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

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

// synthesis attribute declarations
  /* synopsys attribute

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

DCM Waveforms

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

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

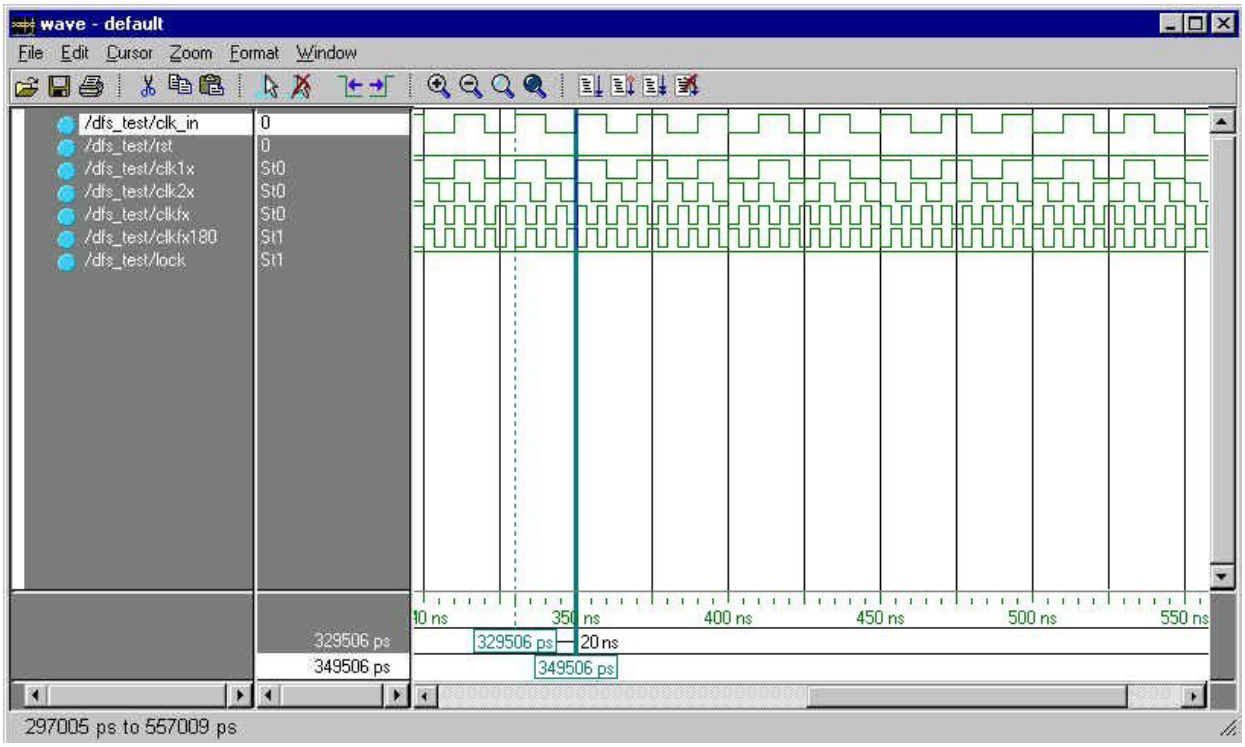
The input clock, 'clk_in' (CLKIN input of DCM) in all these waveforms is 50 MHz. The DCM_DLL waveforms in Figure 3-41 shows four DCM outputs, namely, clk1x (CLK0 output of DCM), clk2x (CLK2X output of DCM), clk90 (CLK90 output of DCM), and clk180 (CLK180 output of DCM).



ug002_c2_095_113000

Figure 3-41: **DCM_DLL Waveforms**

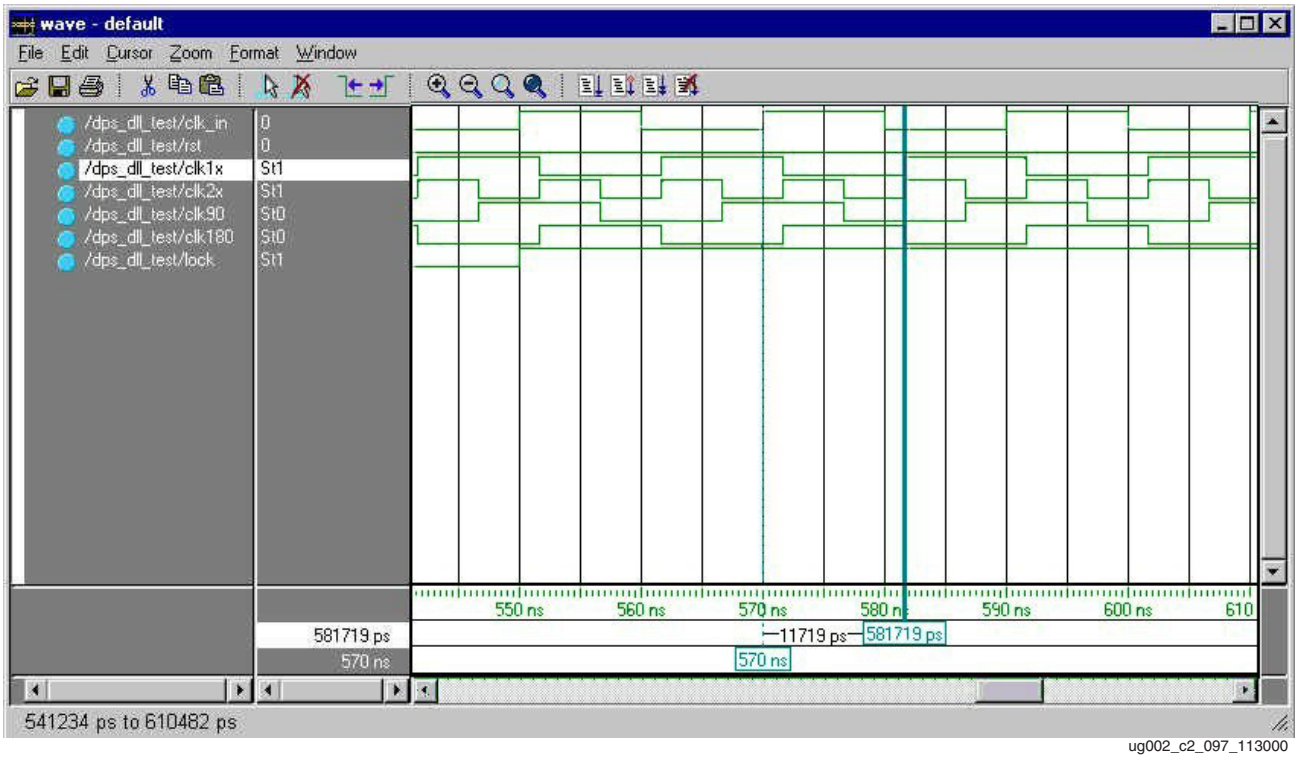
The DCM_DFS Waveforms in Figure 3-42 shows four DCM outputs namely, clk1x (CLK0 output of DCM), clk2x (CLK2X output of DCM), clkfx (CLKFX output of DCM), and clkfx180 (CLKFX180 output of DCM). In this case the attributes, CLKFX_DIVIDE = 1, and the CLKFX_MULTIPLY = 3.



ug002_c2_096_113000

Figure 3-42: **DCM_DFS Waveforms**

The DCM_DPS waveforms in Figure 3-43 shows four DCM outputs, namely, clk1x (CLK0 output of DCM), clk2x (CLK2X output of DCM), clk90 (CLK90 output of DCM), and clk180 (CLK180 output of DCM). In this case, the attribute PHASE_SHIFT = 150 which translates to a phase shift of (150 x 20 ns)/256 = 11.719 ns, where 20 ns is the clock period.



ug002_c2_097_113000

Figure 3-43: **DCM_DPS Waveforms**

The DCM_DPS_DFS waveforms in Figure 3-44 shows four DCM outputs namely, clk1x (CLK0 output of DCM), clk90 (CLK90 output of DCM), clkfx (CLKFX output of DCM), and clkfx180 (CLKFX180 output of DCM). In this case, the attributes, CLKFX_DIVIDE = 1, and the CLKFX_MULTIPLY = 4. The attribute, PHASE_SHIFT = 150 which translates to a phase shift of (150 x 20 ns)/256 = 11.719 ns, where 20 ns is the clock period.

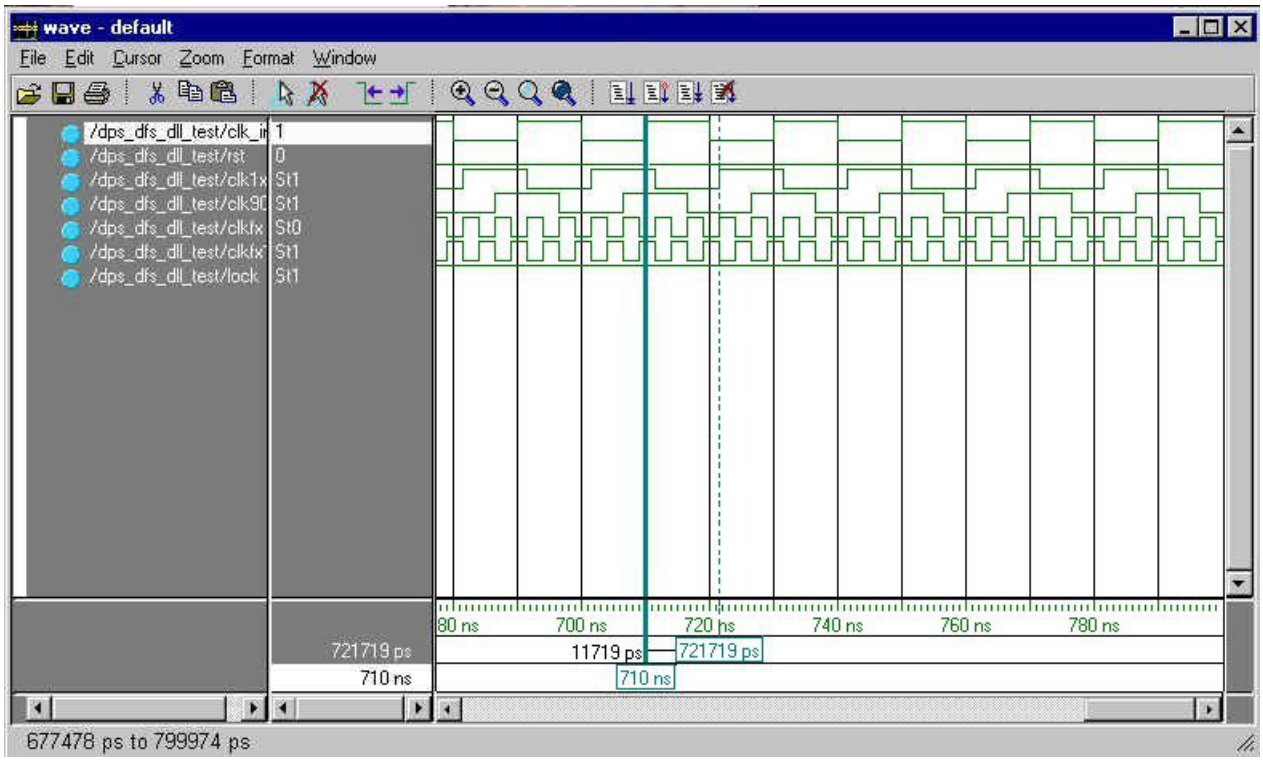


Figure 3-44: **DCM_DPS_DFS Waveforms**

Using Block SelectRAM™ Memory

Introduction

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

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

Synchronous Dual-Port and Single-Port RAM

Data Flow

The 18Kb block SelectRAM dual-port memory consists of an 18-Kb storage area and two completely independent access ports, A and B. The structure is fully symmetrical, and both ports are interchangeable.

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

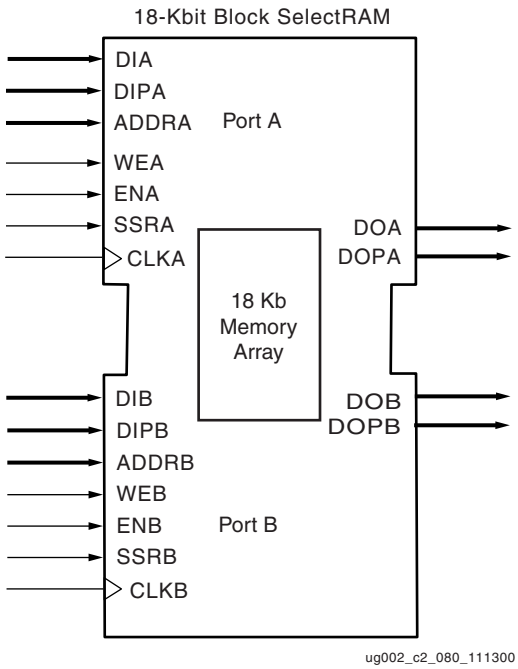


Figure 3-45: **Dual-Port Data Flows**

As described below, there are three options for the behavior of the data output during a write operation on its port. There is no dedicated monitor to arbitrate the result of identical addresses on both ports. It is up to the user to time the two clocks appropriately. However, conflicting simultaneous writes to the same location never cause any physical damage.

Operating Modes

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

Read Operation

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

Write Operations

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

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

WRITE_FIRST or Transparent Mode (Default)

In WRITE_FIRST mode, the input data is simultaneously written into memory and stored in the data output (transparent write), as shown in Figure 3-46.

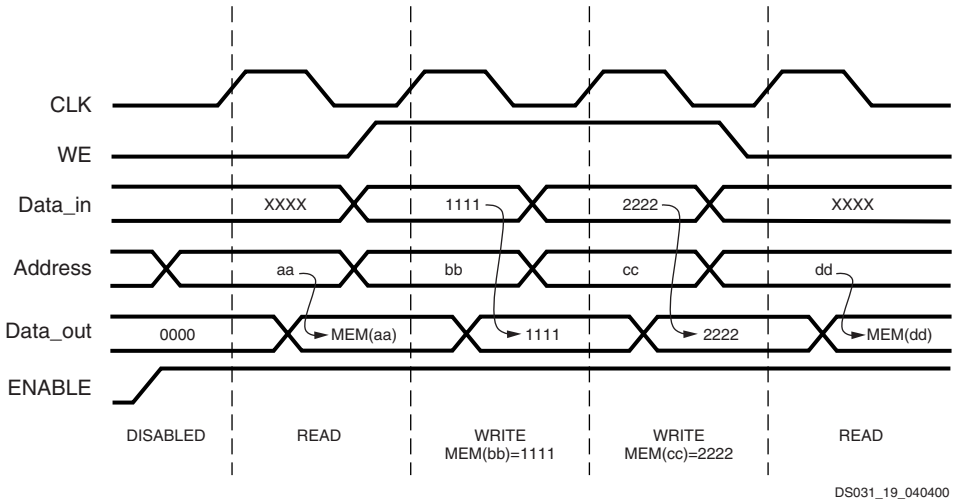


Figure 3-46: **WRITE_FIRST Mode Waveforms**

READ_FIRST or Read-Before-Write Mode

In READ_FIRST mode, data previously stored at the write address appears on the output latches, while the input data is being stored in memory (read before write). See Figure 3-47.

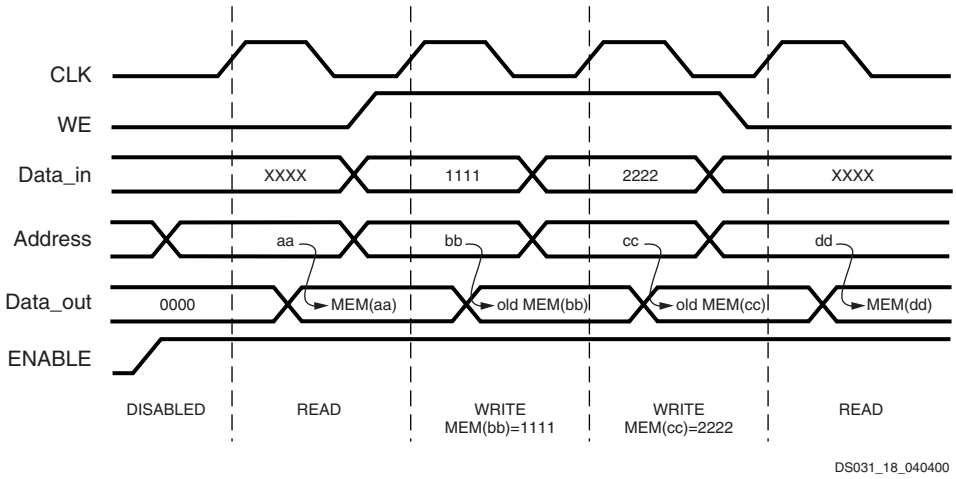


Figure 3-47: **READ_FIRST Mode Waveforms**

NO_CHANGE Mode

In NO_CHANGE mode, the output latches remain unchanged during a write operation. As shown in Figure 3-48, data output is still the last read data and is unaffected by a write operation on the same port.

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

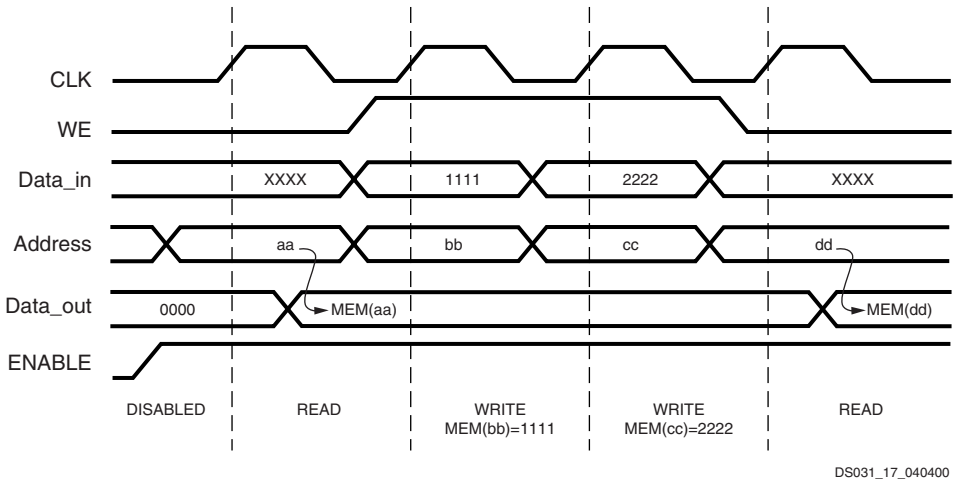


Figure 3-48: **NO_CHANGE Mode Waveforms**

Conflict Resolution

Virtex-II block SelectRAM memory is a True Dual-Port RAM that allows both ports to simultaneously access the same memory cell. When one port writes to a given memory cell, the other port must not address that memory cell (for a write or a read) within the clock-to-clock setup window. Figure 3-49 describes this asynchronous operation.

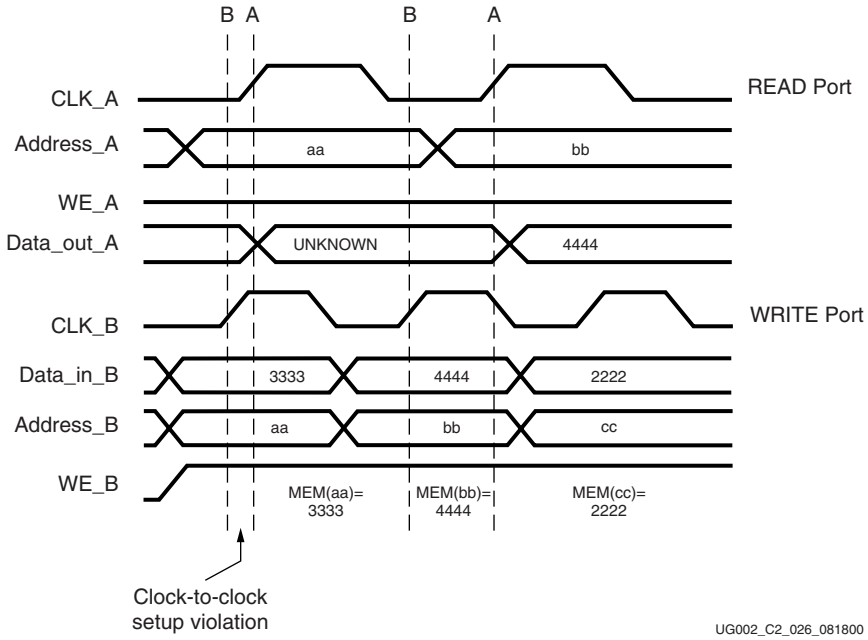


Figure 3-49: **READ-WRITE Conditions**

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

Asynchronous Clocks

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

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

The clock-to-clock setup timing parameter is specified together with other block SelectRAM switching characteristics in the **Virtex-II Data Sheet (DS031)**.

Synchronous Clocks

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

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

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

Characteristics

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

Library Primitives

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

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

Figure 3-50 shows the generic dual-port block RAM primitive. DIA, DIPA, ADDRA, DOA, DOPA, and the corresponding signals on port B are busses.

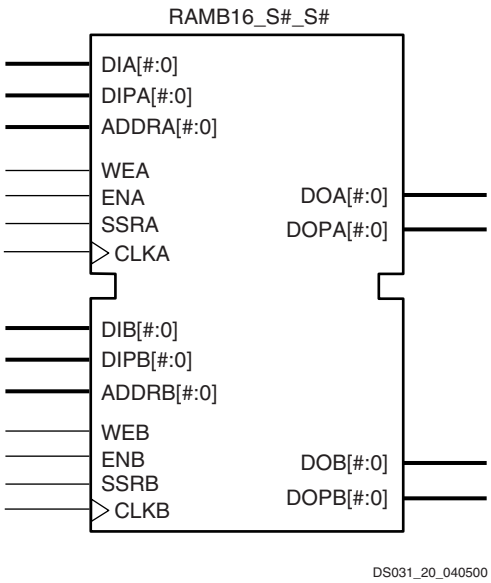


Figure 3-50: **Dual-Port Block RAM Primitive**

Table 3-10 lists the available dual-port primitives for synthesis and simulation.

Table 3-10: **Dual-Port Block RAM Primitives**

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

Figure 3-51 shows the generic single-port block RAM primitive. DI, DIP, ADDR, DO, and DOP are busses.

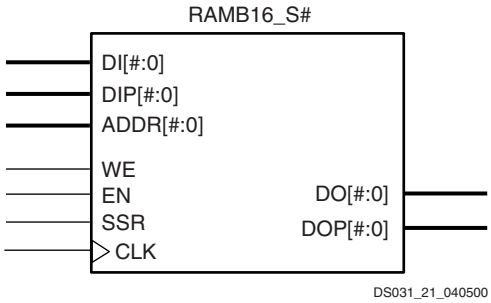


Figure 3-51: **Single-Port Block RAM Primitive**

Table 3-11 lists all of the available single-port primitives for synthesis and simulation.

Table 3-11: **Single-Port Block RAM Primitives**

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

VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available as examples (see "VHDL and Verilog Templates" on page 122).

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

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

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

Port Signals

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

Clock - CLK[A|B]

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

Enable - EN[A|B]

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

Write Enable - WE[A|B]

Both EN and WE are active when the contents of the data input bus is written to memory at the address pointed to by the address bus. The output latches are loaded or not loaded according to the write configuration (WRITE_FIRST, READ_FIRST, NO_CHANGE). When inactive, a read operation occurs, and the contents of the memory cells referenced by the address bus reflect on the data-out bus, regardless of the write mode attribute. Write enable polarity is configurable (active High by default).

Set/Reset - SSR[A|B]

The SSR pin forces the data output latches to contain the value "SRVAL" (see "Attributes" on page 120). The data output latches are synchronously asserted to 0 or 1, including the parity bit. In a 36-bit width configuration, each port has an independent SRVAL[A|B] attribute of 36 bits. This operation does not affect RAM memory cells and does not disturb write operations on the other port. Like the read and write operation, the set/reset function is active only when the enable pin of the port is active. Set/reset polarity is configurable (active High by default).

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

The address bus selects the memory cells for read or write. The width of the port determines the required address bus width, as shown in Table 3-12.

Table 3-12: **Port Aspect Ratio**

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

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

Data-in busses provide the new data value to be written into RAM. The regular data-in bus (DI) and the parity data-in bus (when available) have a total width equal to the port width. For example the 36-bit port data width is represented by DI<31:0> and DIP<3:0>, as shown in Table 3-12.

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

Data-out busses reflect the contents of memory cells referenced by the address bus at the last active clock edge during a read operation. During a write operation (WRITE_FIRST or READ_FIRST configuration), the data-out busses reflect either the data-in busses or the stored value before write. During a write operation in NO_CHANGE mode, data-out busses are not affected. The regular data-out bus (DO) and the parity data-out bus (DOP) (when available) have a total width equal to the port width, as shown in Table 3-12.

Inverting Control Pins

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

Unused Inputs

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

GSR

The global set/reset (GSR) signal of a Virtex-II device is an asynchronous global signal that is active at the end of device configuration. The GSR can also restore the initial Virtex-II state at any time. The GSR signal initializes the output latches to the INIT, or to the INIT_A and INIT_B value (see “Attributes” on page 120). A GSR signal has no impact on internal memory contents. Because it is a global signal, the GSR has no input pin at the functional level (block SelectRAM primitive).

Address Mapping

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

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

Table 3-13 shows low-order address mapping for each port width.

Table 3-13: **Port Address Mapping**

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

Attributes

Content Initialization - INIT_xx

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

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

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

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

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

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

More examples are given in Table 3-14.

Table 3-14: **Block SelectRAM Initialization Attributes**

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

Content Initialization - INITP_xx

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

Output Latches Initialization - INIT (INIT_A & INIT_B)

The INIT (single-port) or INIT_A and INIT_B (dual-port) attributes define the output latches values after configuration. The width of the INIT (INIT_A & INIT_B) attribute is the port width, as shown in Table 3-15. These attributes are hex-encoded bit vectors and the default value is 0.

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

The SRVAL (single-port) or SRVAL_A and SRVAL_B (dual-port) attributes define output latch values when the SSR input is asserted. The width of the SRVAL (SRVAL_A and SRVAL_B) attribute is the port width, as shown in Table 3-15. These attributes are hex-encoded bit vectors and the default value is 0.

Table 3-15: **Port Width Values**

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

Initialization in VHDL or Verilog Codes

Block SelectRAM memory structures can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the block SelectRAM instantiation and are copied in the EDIF output file to be compiled by Xilinx Alliance Series™ tools. The VHDL code simulation uses a `generic` parameter to pass the attributes. The Verilog code simulation uses a `defparam` parameter to pass the attributes.

The XC2V_RAMB_1_PORT block SelectRAM instantiation code examples (in VHDL and Verilog) illustrate these techniques (see "VHDL and Verilog Templates" on page 122).

Location Constraints

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

The LOC properties use the following form:

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

The RAMB16_X0Y0 is the bottom-left block SelectRAM location on the device.

Applications

Creating Larger RAM Structures

Block SelectRAM columns have specialized routing to allow cascading blocks with minimal routing delays. Wider or deeper RAM structures are achieved with a smaller timing penalty than is encountered when using normal routing resources.

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

Multiple RAM Organizations

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

- **XAPP258** "FIFOs Using Virtex-II Block RAM"
- **XAPP260** "Using Virtex-II Block RAM for High-Performance Read/Write CAMs"

VHDL and Verilog Templates

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

The following are single-port templates:

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

The following are dual-port templates:

- SelectRAM_A1_B1
- SelectRAM_A1_B2
- SelectRAM_A1_B4
- SelectRAM_A1_B9
- SelectRAM_A1_B18
- SelectRAM_A1_B36
- SelectRAM_A2_B2
- SelectRAM_A2_B4
- SelectRAM_A2_B9
- SelectRAM_A2_B18
- SelectRAM_A2_B36
- SelectRAM_A4_B4
- SelectRAM_A4_B9
- SelectRAM_A4_B18
- SelectRAM_A4_B36
- SelectRAM_A9_B9
- SelectRAM_A9_B18
- SelectRAM_A9_B36

- SelectRAM_A18_B18
- SelectRAM_A18_B36
- SelectRAM_A36_B36

VHDL Template

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

```
-- Module: XC2V_RAMB_1_PORT
-- Description: 18Kb Block SelectRAM example
-- Single Port 512 x 36 bits
-- Use template "SelectRAM_A36.vhd"
--
-- Device: Virtex-II Family
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--
-- Syntax for Synopsys FPGA Express
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
--
entity XC2V_RAMB_1_PORT is
   port (
    DATA_IN : in std_logic_vector (35 downto 0);
        ADDRESS   : in std_logic_vector (8 downto 0);
        ENABLE: in std_logic;
        WRITE_EN     : in std_logic;
        SET_RESET    : in std_logic;
        CLK    : in std_logic;
        DATA_OUT     : out std_logic_vector (35 downto 0)
       );
end XC2V_RAMB_1_PORT;
--
architecture XC2V_RAMB_1_PORT_arch of XC2V_RAMB_1_PORT is
--
-- Components Declarations:
--
component BUFG
  port (
  I : in std_logic;
  O : out std_logic
  );
end component;
--
-- Syntax for Synopsys FPGA Express
component RAMB16_S36
-- pragma translate_off
  generic (
-- "Read during Write" attribute for functional simulation
  WRITE_MODE : string := "READ_FIRST" ; -- WRITE_FIRST(default)/
READ_FIRST/ NO_CHANGE
-- Output value after configuration
  INIT : bit_vector(35 downto 0)  := X"000000000";
-- Output value if SSR active
  SRVAL : bit_vector(35 downto 0)  := X"012345678";
-- Plus bits initial content
        INITP_00 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
```

```
       INITP_01 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_02 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_03 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_04 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_05 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_06 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_07 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
-- Regular bits initial content
       INIT_00 : bit_vector(255 downto 0) :=
X"00000000000000000000000000000000000000000000000FEDCBA9876543210";
       INIT_01 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       ... (cut)
       INIT_3E : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3F : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000"
  );
-- pragma translate_on
  port (
        DI     : in std_logic_vector (31 downto 0);
        DIP    : in std_logic_vector (3 downto 0);
        ADDR   : in std_logic_vector (8 downto 0);
        EN     : in STD_LOGIC;
        WE     : in STD_LOGIC;
        SSR    : in STD_LOGIC;
        CLK    : in STD_LOGIC;
        DO     : out std_logic_vector (31 downto 0);
        DOP    : out std_logic_vector (3 downto 0)
  );
end component;
--
-- Attribute Declarations:
attribute WRITE_MODE : string;
attribute INIT: string;
attribute SRVAL: string;
--
attribute INITP_00: string;
attribute INITP_01: string;
attribute INITP_02: string;
attribute INITP_03: string;
attribute INITP_04: string;
attribute INITP_05: string;
attribute INITP_06: string;
attribute INITP_07: string;
--
attribute INIT_00: string;
attribute INIT_01: string;
attribute INIT_02: string;
… (cut)
attribute INIT_3E: string;
attribute INIT_3F: string;
--
```

```
-- Attribute "Read during Write mode" = WRITE_FIRST(default)/
READ_FIRST/ NO_CHANGE
attribute WRITE_MODE of U_RAMB16_S36: label is "READ_FIRST";
attribute INIT of U_RAMB16_S36: label is "000000000";
attribute SRVAL of U_RAMB16_S36: label is "012345678";
--
-- RAMB16 memory initialization for Alliance
-- Default value is "0" / Partial initialization strings are padded
-- with zeros to the left
attribute INITP_00 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
attribute INITP_01 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_02 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_03 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_04 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_05 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_06 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INITP_07 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
--
attribute INIT_00 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
attribute INIT_01 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_02 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
... (cut)
attribute INIT_3E of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
attribute INIT_3F of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
--
--
-- Signal Declarations:
--
-- signal VCC : std_logic;
-- signal GND : std_logic;
signal CLK_BUFG: std_logic;
signal INV_SET_RESET : std_logic;
--
begin
-- VCC <= '1';
-- GND <= '0';
--
-- Instantiate the clock Buffer
U_BUFG: BUFG
  port map (
  I => CLK,
  O => CLK_BUFG
  );
--
-- Use of the free inverter on SSR pin
INV_SET_RESET <= NOT SET_RESET;

-- Block SelectRAM Instantiation
U_RAMB16_S36: RAMB16_S36
```

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

Verilog Template

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

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


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

wire CLK_BUFG, INV_SET_RESET;

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


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

  //Plus bits initial content
  U_RAMB16_S36.INITP_00 =
256'h0123456789ABCDEF0000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_01 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_02 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
```

```
  U_RAMB16_S36.INITP_03 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_04 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_05 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_06 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INITP_07 =
256'h0000000000000000000000000000000000000000000000000000000000000000,

  //Regular bits initial content
  U_RAMB16_S36.INIT_00 =
256'h0123456789ABCDEF0000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INIT_01 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
  U_RAMB16_S36.INIT_02 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
...<cut>
  U_RAMB16_S36.INIT_3E =
256'h0000000000000000000000000000000000000000000000000000000000000000,
  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
"0123456789ABCDEF0000000000000000000000000000000000000000000000000"
  INITP_01
"0000000000000000000000000000000000000000000000000000000000000000"
  INITP_02
"0000000000000000000000000000000000000000000000000000000000000000"
  INITP_03
"0000000000000000000000000000000000000000000000000000000000000000"
  INITP_04
"0000000000000000000000000000000000000000000000000000000000000000"
  INITP_05
"0000000000000000000000000000000000000000000000000000000000000000"
  INITP_06
"0000000000000000000000000000000000000000000000000000000000000000"
  INITP_07
"0000000000000000000000000000000000000000000000000000000000000000"
```

```
  INIT_00
"0123456789ABCDEF0000000000000000000000000000000000000000000000000000"
  INIT_01
"0000000000000000000000000000000000000000000000000000000000000000000000"
  INIT_02
"0000000000000000000000000000000000000000000000000000000000000000000000"
...<cut>
  INIT_3E
"0000000000000000000000000000000000000000000000000000000000000000000000"
  INIT_3F
"0000000000000000000000000000000000000000000000000000000000000000000000"
  */

endmodule
```

Using Distributed SelectRAM Memory

Introduction

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

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

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

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

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

Single-Port and Dual-Port RAM

Data Flow

Distributed SelectRAM memory supports the following:

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

As illustrated in the Figure 3-52, the dual port has one read/write port and an independent read port.

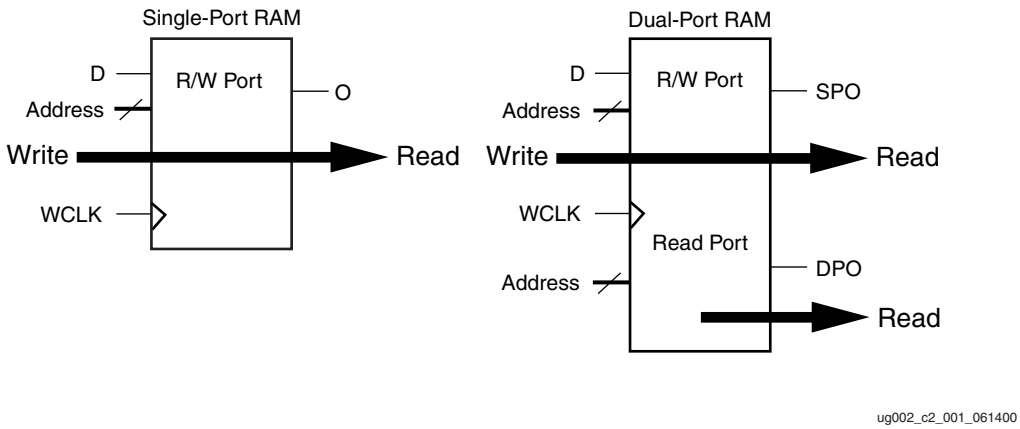


Figure 3-52: **Single-Port and Dual-Port Distributed SelectRAM**

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

Write Operations

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

Read Operation

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

Read During Write

When new data is synchronously written, the output reflects the data in the memory cell addressed (transparent mode). The timing diagram in Figure 3-53 illustrates a write operation, with the previous data read on the output port, before the clock edge and then the new data.

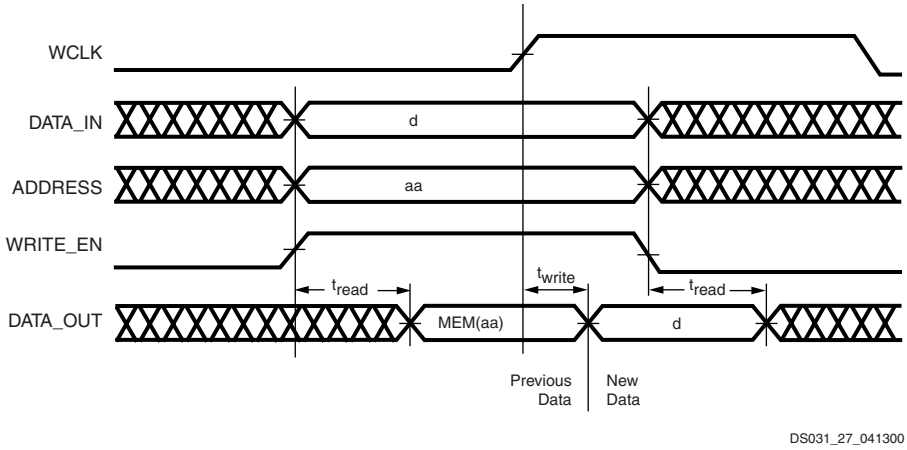


Figure 3-53: **Write Timing Diagram**

Characteristics

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

Library Primitives

Seven library primitives from 16 x 1-bit to 128 x 1-bit are available. Four primitives are single-port RAM and three primitives are True Dual-Port RAM, as shown in Table 3-16.

Table 3-16: **Single-Port and Dual-Port Distributed SelectRAM**

Primitive	RAM Size	Type	Address Inputs
RAM16X1S	16 bits	single-port	A3, A2, A1, A0
RAM32X1S	32 bits	single-port	A4, A3, A2, A1, A0
RAM64X1S	64 bits	single-port	A5, A4, A3, A2, A1, A0
RAM128X1S	128 bits	single-port	A6, A5, A4, A3, A2, A1, A0
RAM16X1D	16 bits	dual-port	A3, A2, A1, A0
RAM32X1D	32 bits	dual-port	A4, A3, A2, A1, A0
RAM64X1D	64 bits	dual-port	A5, A4, A3, A2, A1, A0

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

Figure 3-54 shows generic single-port and dual-port distributed SelectRAM primitives. The A and DPRA signals are address busses.

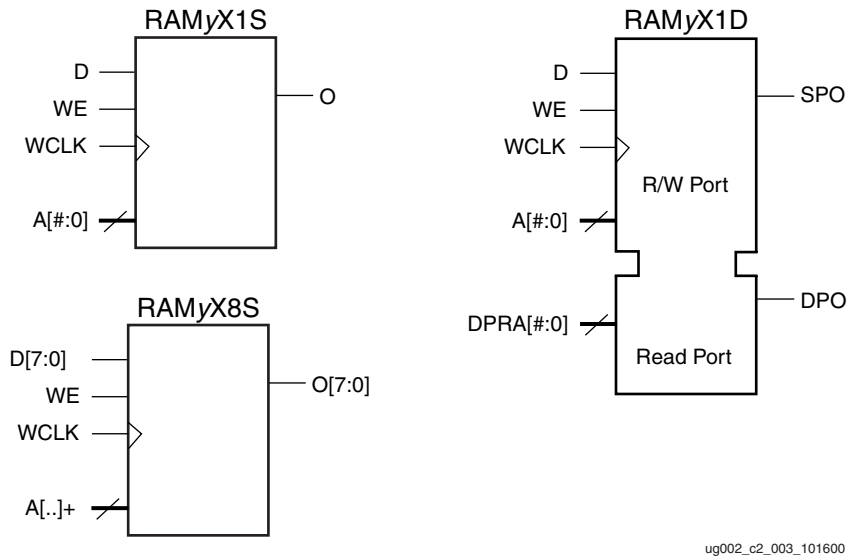


Figure 3-54: **Single-Port and Dual-Port Distributed SelectRAM Primitive**

As shown in Table 3-17, wider library primitives are available for 2-bit, 4-bit, and 8-bit RAM.

Table 3-17: **Wider Library Primitives**

Primitive	RAM Size	Data Inputs	Address Inputs	Data Outputs
RAM16x2S	16 x 2-bit	D1, D0	A3, A2, A1, A0	O1, O0
RAM32X2S	32 x 2-bit	D1, D0	A4, A3, A2, A1, A0	O1, O0
RAM64X2S	64 x 2-bit	D1, D0	A5, A4, A3, A2, A1, A0	O1, O0
RAM16X4S	16 x 4-bit	D3, D2, D1, D0	A3, A2, A1, A0	O3, O2, O1, O0
RAM32X4S	32 x 4-bit	D3, D2, D1, D0	A4,A3, A2, A1, A0	O3, O2, O1, O0
RAM16X8S	16 x 8-bit	D <7:0>	A3, A2, A1, A0	O <7:0>
RAM32X8S	32 x 8-bit	D <7:0>	A4,A3, A2, A1, A0	O <7:0>

VHDL and Verilog Instantiation

VHDL and Verilog instantiations templates are available as examples (see "VHDL and Verilog Templates" on page 136).

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

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

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

Ports Signals

Each distributed SelectRAM port operates independently of the other while reading the same set of memory cells.

Clock - WCLK

The clock is used for the synchronous write. The data and the address input pins have setup time referenced to the WCLK pin.

Enable - WE

The enable pin affects the write functionality of the port. An inactive Write Enable prevents any writing to memory cells. An active Write Enable causes the clock edge to write the data input signal to the memory location pointed to by the address inputs.

Address - A0, A1, A2, A3 (A4, A5, A6)

The address inputs select the memory cells for read or write. The width of the port determines the required address inputs. Note that the address inputs are not a bus in VHDL or Verilog instantiations.

Data In - D

The data input provides the new data value to be written into the RAM.

Data Out - O, SPO, and DPO

The data out O (Single-Port or SPO) and DPO (Dual-Port) reflects the contents of the memory cells referenced by the address inputs. Following an active write clock edge, the data out (O or SPO) reflects the newly written data.

Inverting Control Pins

The two control pins (WCLK and WE) each have an individual inversion option. Any control signal, including the clock, can be active at 0 (negative edge for the clock) or at 1 (positive edge for the clock) without requiring other logic resources.

GSR

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

Attributes

Content Initialization - INIT

With the INIT attributes, users can define the initial memory contents after configuration. By default distributed SelectRAM memory is initialized with all zeros during the device configuration sequence. The initialization attribute INIT represents the specified memory

contents. Each INIT is a hex-encoded bit vector. Table 3-18 shows the length of the INIT attribute for each primitive.

Table 3-18: **INIT Attributes Length**

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

Initialization in VHDL or Verilog Codes

Distributed SelectRAM memory structures can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the distributed SelectRAM instantiation and are copied in the EDIF output file to be compiled by Xilinx Alliance Series™ tools. The VHDL code simulation uses a `generic` parameter to pass the attributes. The Verilog code simulation uses a `defparam` parameter to pass the attributes.

The distributed SelectRAM instantiation templates (in VHDL and Verilog) illustrate these techniques (see "VHDL and Verilog Templates" on page 136).

Location Constraints

The CLB has four slices S0, S1, S2 and S3. As an example, in the bottom left CLB, the slices have the coordinates shown below: S

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

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

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

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

The RAM16X1D primitive occupies half of two slices, as shown in Figure 3-55. The first slice (output SPO) implements the read/write port with the same address A[3:0] for read

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

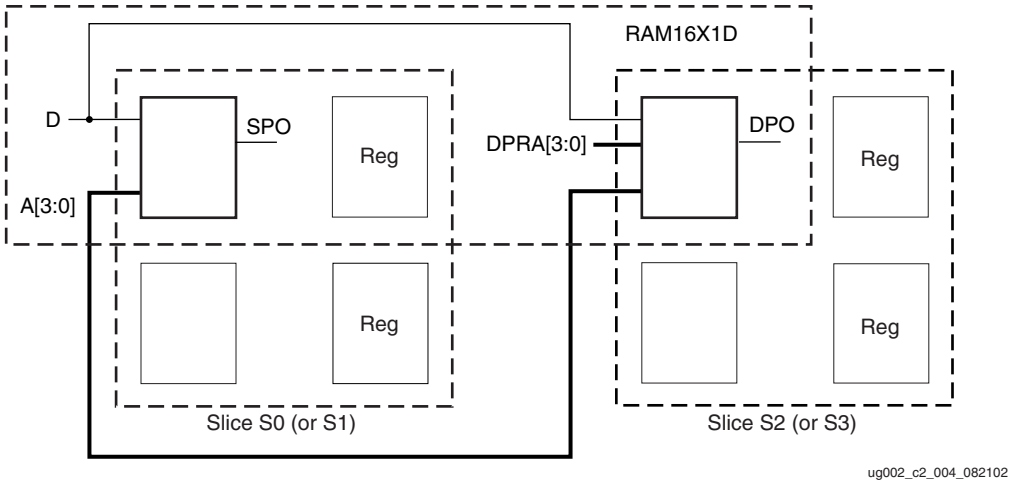


Figure 3-55: **RAM16X1D Placement**

In the same CLB module, the dual-port RAM16X1D either occupies half of slices S0 (X0Y0) and S2 (X1Y0), or half of slices S1 (X0Y1) and S3 (X1Y1).

If a dual-port 16 x 2-bit module is built, the two RAM16X1D primitives occupy two slices, as long as they share the same clock and write enable, as illustrated in Figure 3-56.

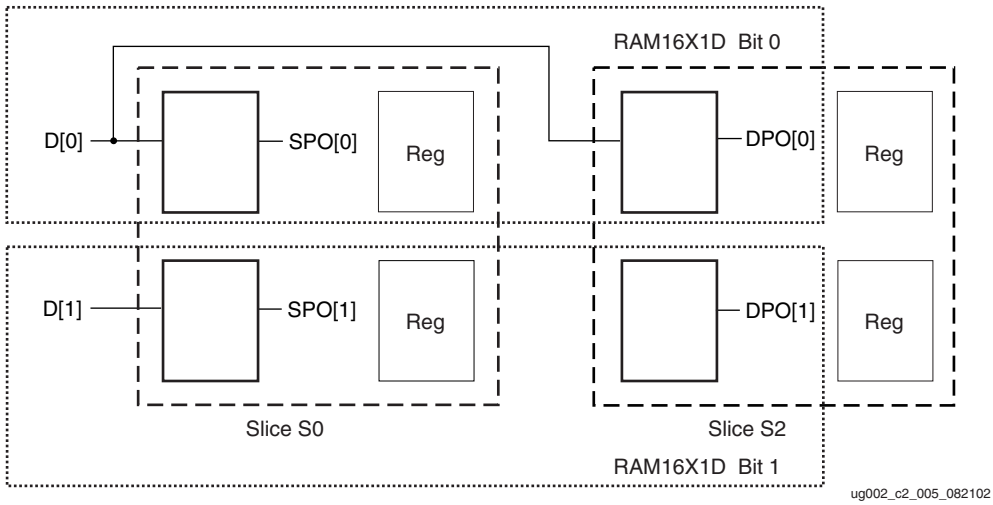


Figure 3-56: **Two RAM16X1D Placement**

A RAM32X1S primitive fits in one slice, as shown in Figure 3-57.

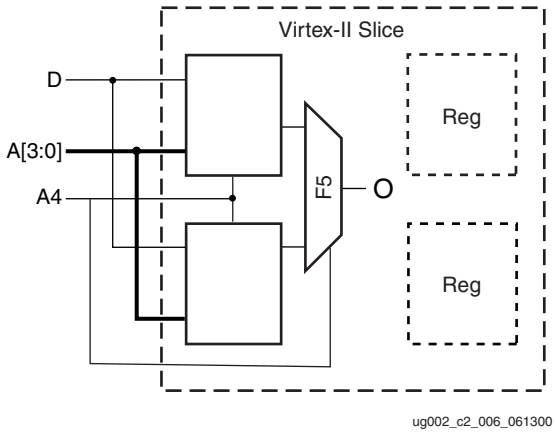


Figure 3-57: **RAM32X1S Placement**

Following the same rules, a RAM32X1D primitive fits in two slices, with one slice implementing the read/write port and the second slice implementing the second read port.

The RAM64X1S primitive occupies two slices and the RAM64X1D primitive occupies four slices (one CLB element), with two slices implementing the read/write port and two other slices implementing the second read port. The RAM64X1S read path is built on the MUXF5 and MUXF6 multiplexers.

The RAM128X1S primitive occupies four slices, equivalent to one CLB element.

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

Applications

Creating Larger RAM Structures

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

Table 3-19 shows the generic VHDL and Verilog distributed SelectRAM examples provided to implement *n*-bit-wide memories.

Table 3-19: **VHDL and Verilog Submodules**

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

By using the read/write port for the write address and the second read port for the read address, a FIFO that can read and write simultaneously is easily generated. Simultaneous access doubles the effective throughput of the memory.

VHDL and Verilog Templates

VHDL and Verilog templates are available for all single-port and dual-port primitives. The number in each template indicates the number of bits (for example, SelectRAM_16S is the template for the 16 x 1-bit RAM); S indicates single-port, and D indicates dual-port.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

The following are single-port templates:

- SelectRAM_16S
- SelectRAM_32S
- SelectRAM_64S
- SelectRAM_128S

The following are dual-port templates:

- SelectRAM_16D
- SelectRAM_32D
- SelectRAM_64D

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

VHDL Template

```
--
-- Module: SelectRAM_16S
--
-- Description: VHDL instantiation template
--              Distributed SelectRAM
--              Single Port 16 x 1
--              can be used also for RAM16X1S_1
--
-- Device: Virtex-II Family
--
-----------------------------------------------------------------------
--
-- Components Declarations:
--
component RAM16X1S
-- pragma translate_off
   generic (
-- RAM initialization ("0" by default) for functional simulation:
         INIT : bit_vector := X"0000"
         );
-- pragma translate_on
  port (
         D    : in std_logic;
         WE   : in std_logic;
         WCLK : in std_logic;
         A0   : in std_logic;
         A1   : in std_logic;
         A2   : in std_logic;
         A3   : in std_logic;
         O    : out std_logic
         );
end component;
--
-----------------------------------------------------------------------
--
-- Architecture  section:
--
-- Attributes for RAM initialization ("0" by default):
attribute INIT: string;
--
attribute INIT of U_RAM16X1S: label is "0000";
--
-- Distributed SelectRAM Instantiation
U_RAM16X1S: RAM16X1S
  port map (
    D      => , -- insert input signal
    WE     => , -- insert Write Enable signal
    WCLK   => , -- insert Write Clock signal
    A0     => , -- insert Address 0 signal
    A1     => , -- insert Address 1 signal
    A2     => , -- insert Address 2 signal
    A3     => , -- insert Address 3 signal
    O      =>   -- insert output signal
  );
--
-----------------------------------------------------------------------
```

Verilog Template

```
//
// Module: SelectRAM_16S
//
// Description: Verilog instantiation template
//              Distributed SelectRAM
//              Single Port 16 x 1
//              can be used also for RAM16X1S_1
//
// Device: Virtex-II Family
//
//-----------------------------------------------------------------------
//
//
// Syntax for Synopsys FPGA Express
// synopsys translate_off

  defparam

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


//Distributed SelectRAM Instantiation
RAM16X1S U_RAM16X1S (  .D(),      // insert input signal
                 .WE(),     // insert Write Enable signal
                .WCLK(),   // insert Write Clock signal
                 .A0(),     // insert Address 0 signal
                 .A1(),     // insert Address 1 signal
                 .A2(),     // insert Address 2 signal
                 .A3(),     // insert Address 3 signal
                 .O()       // insert output signal
                  );

// synthesis attribute declarations
  /* synopsys attribute

  INIT "0000"
  */
```

Using Look-Up Tables as Shift Registers (SRLs)

Introduction

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

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

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

Shift Register Operations

Data Flow

Each shift register (SRL16 primitive) supports:

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

In addition, cascadable shift registers (SRLC16) support synchronous shift-out output of the last (16th) bit. This output has a dedicated connection to the input of the next SRLC16 inside the CLB resource. Two primitives are illustrated in Figure 3-58.

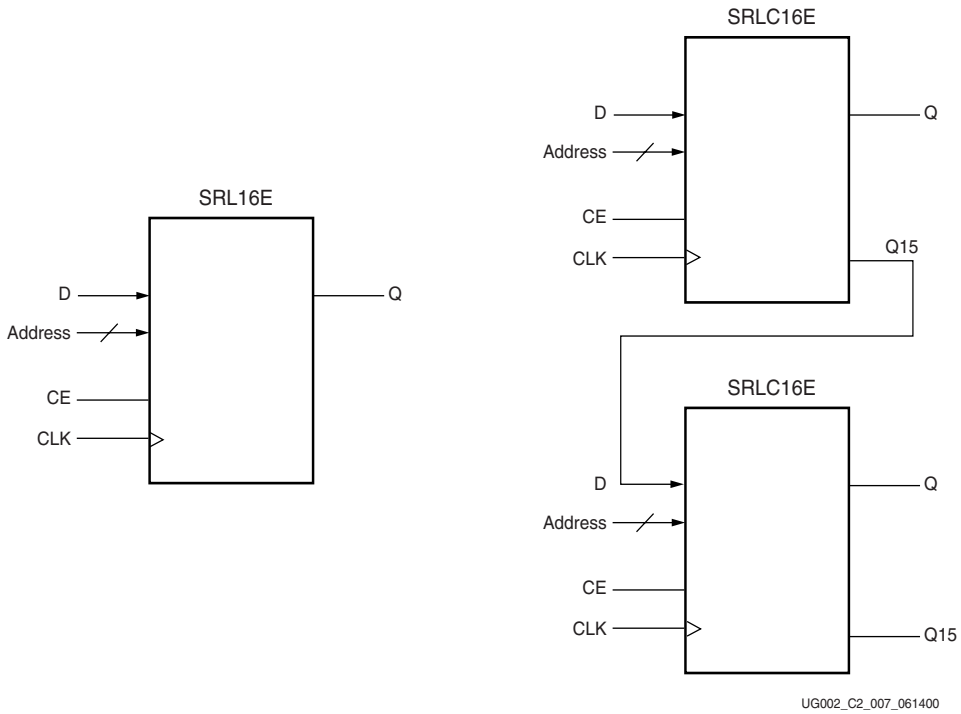


Figure 3-58: **Shift Register and Cascadable Shift Register**

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 3-59 illustrates the shift and dynamic read operations.

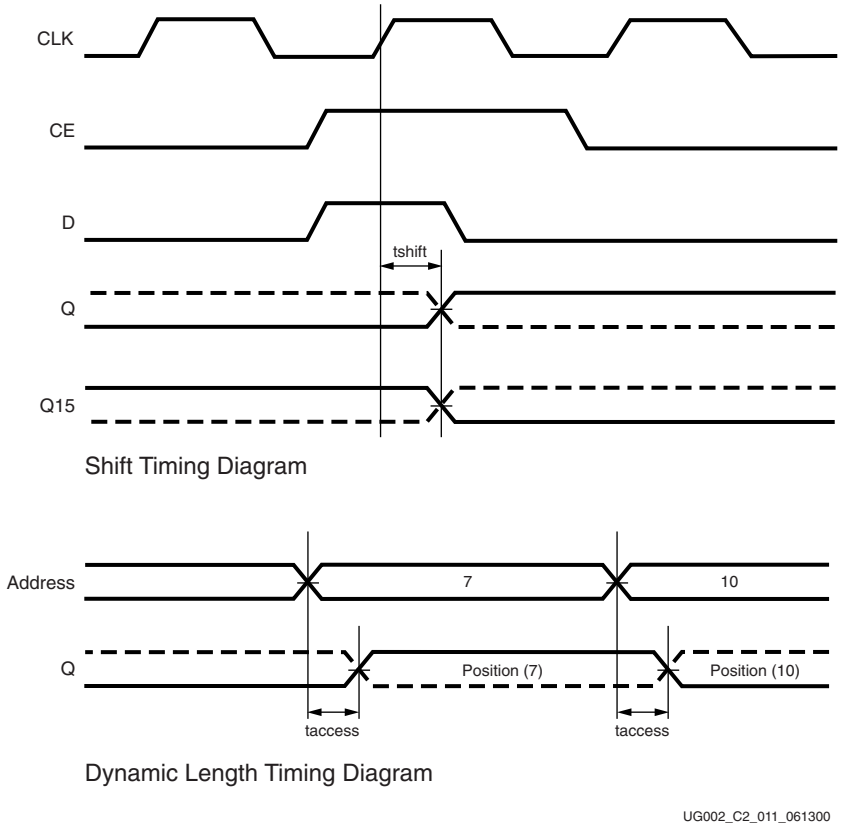


Figure 3-59: **Shift- and Dynamic-Length Timing Diagrams**

Static Read Operation

If the 4-bit address is fixed, the Q output always uses the same bit position. This mode implements any shift register length up 1 to 16 bits in one LUT. Shift register length is (N+1) where N is the input address.

The Q output changes synchronously with each shift operation. The previous bit is shifted to the next position and appears on the Q output.

Characteristics

- A shift operation requires one clock edge.
- Dynamic-length read operations are asynchronous (Q output).
- Static-length read operations are synchronous (Q output).
- The data input has a setup-to-clock timing specification.
- In a cascadable configuration, the Q15 output always contains the last bit value.
- The Q15 output changes synchronously after each shift operation.

Library Primitives and Submodules

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

Table 3-20 lists all of the available primitives for synthesis and simulation.

Table 3-20: **Shift Register Primitives**

Primitive	Length	Control	Address Inputs	Output
SRL16	16 bits	CLK	A3,A2,A1,A0	Q
SRL16E	16 bits	CLK, CE	A3,A2,A1,A0	Q
SRL16_1	16 bits	$\overline{\text{CLK}}$	A3,A2,A1,A0	Q
SRL16E_1	16 bits	$\overline{\text{CLK}}$, CE	A3,A2,A1,A0	Q
SRLC16	16 bits	CLK	A3,A2,A1,A0	Q, Q15
SRLC16E	16 bits	CLK, CE	A3,A2,A1,A0	Q, Q15
SRLC16_1	16 bits	$\overline{\text{CLK}}$	A3,A2,A1,A0	Q, Q15
SRLC16E_1	16 bits	$\overline{\text{CLK}}$, CE	A3,A2,A1,A0	Q, Q15

In addition to the 16-bit primitives, three submodules that implement 32-bit, 64-bit, and 128-bit cascadable shift registers are provided in VHDL and Verilog code. Table 3-21 lists available submodules.

Table 3-21: **Shift Register Submodules**

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

The submodules are based on SRLC16E primitives, which are associated with dedicated multiplexers (MUXF5, MUXF6, and so forth). This implementation allows a fast static- and dynamic-length mode, even for very large shift registers.

Figure 3-60 represents the cascadable shift registers (32-bit and 64-bit) implemented by the submodules in Table 3-21.

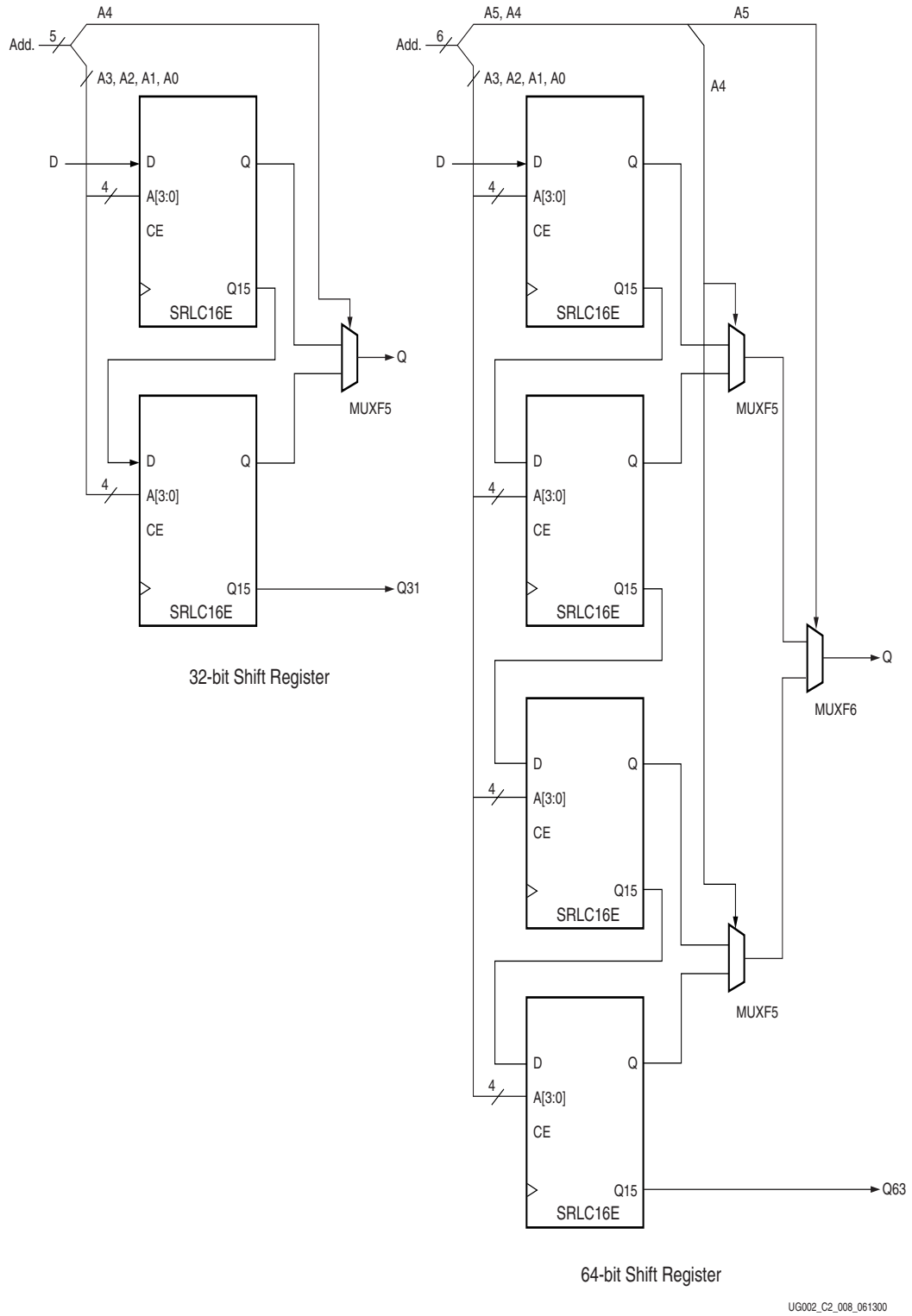


Figure 3-60: **Shift-Register Submodules (32-bit, 64-bit)**

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

All clock enable (CE) and clock (CLK) inputs are connected to one global clock enable and one clock signal per submodule. If a global static- or dynamic-length mode is not required, the SRLC16E primitive can be cascaded without multiplexers.

Initialization in VHDL and Verilog Code

A shift register can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attribute is attached to the 16-bit shift register instantiation and is copied in the EDIF output file to be compiled by Xilinx Alliance Series tools. The VHDL code simulation uses a `generic` parameter to pass the attributes. The Verilog code simulation uses a `defparam` parameter to pass the attributes.

The V2_SRL16E shift register instantiation code examples (in VHDL and Verilog) illustrate these techniques (see "VHDL and Verilog Templates" on page 148). V2_SRL16E.vhd and .v files are not a part of the documentation.

Port Signals

Clock - CLK

Either the rising edge or the falling edge of the clock is used for the synchronous shift-in. The data and clock enable input pins have set-up times referenced to the chosen edge of CLK.

Data In - D

The data input provides new data (one bit) to be shifted into the shift register.

Clock Enable - CE (optional)

The clock enable pin affects shift functionality. An inactive clock enable pin does not shift data into the shift register and does not write new data. Activating the clock enable allows the data in (D) to be written to the first location and all data to be shifted by one location. When available, new data appears on output pins (Q) and the cascadable output pin (Q15).

Address - A0, A1, A2, A3

Address inputs select the bit (range 0 to 15) to be read. The n^{th} bit is available on the output pin (Q). Address inputs have no effect on the cascadable output pin (Q15), which is always the last bit of the shift register (bit 15).

Data Out - Q

The data output Q provides the data value (1 bit) selected by the address inputs.

Data Out - Q15 (optional)

The data output Q15 provides the last bit value of the 16-bit shift register. New data becomes available after each shift-in operation.

Inverting Control Pins

The two control pins (CLK, CE) have an individual inversion option. The default is the rising clock edge and active High clock enable.

GSR

The global set/reset (GSR) signal has no impact on shift registers.

Attributes

Content Initialization - INIT

The INIT attribute defines the initial shift register contents. The INIT attribute is a hex-encoded bit vector with four digits (0000).The left-most hexadecimal digit is the most significant bit. By default the shift register is initialized with all zeros during the device configuration sequence, but any other configuration value can be specified.

Location Constraints

Each CLB resource has four slices: S0, S1, S2, and S3. As an example, in the bottom left CLB resource, each slice has the coordinates shown in Table 3-22.

Table 3-22: **Slice Coordinates in the Bottom-Left CLB Resource**

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

To constrain placement, shift register instances can have LOC properties attached to them. Each 16-bit shift register fits in one LUT.

A 32-bit shift register in static or dynamic address mode fits in one slice (two LUTs and one MUXF5). This shift register can be placed in any slice.

A 64-bit shift register in static or dynamic address mode fits in two slices. These slices are either S0 and S1, or S2 and S3. Figure 3-61 illustrates the position of the four slices in a CLB resource.

The dedicated CLB shift chain runs from the top slice to the bottom slice. The data input pin must either be in slice S1 or in S3. The address selected as the output pin (Q) is the MUXF6 output.

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

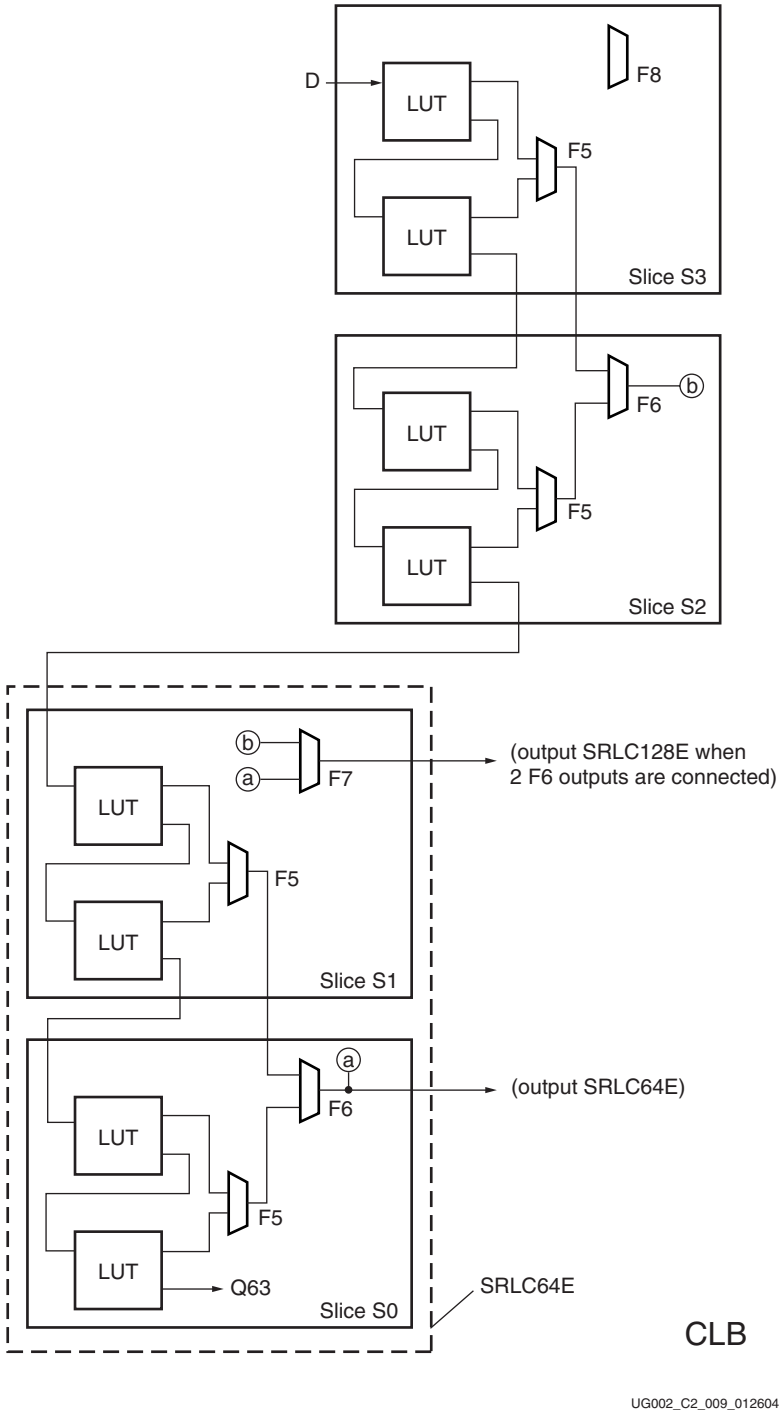


Figure 3-61: **Shift Register Placement**

Fully Synchronous Shift Registers

All shift-register primitives and submodules do not use the register(s) available in the same slice(s). To implement a fully synchronous read and write shift register, output pin Q must be connected to a flip-flop. Both the shift register and the flip-flop share the same clock, as shown in Figure 3-62.

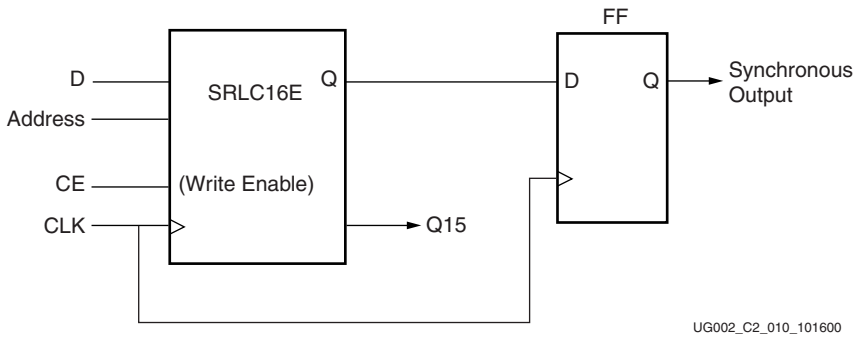


Figure 3-62: **Fully Synchronous Shift Register**

This configuration provides a better timing solution and simplifies the design. Because the flip-flop must be considered to be the last register in the shift-register chain, the static or dynamic address should point to the desired length minus one. If needed, the cascadable output can also be registered in a flip-flop.

Static-Length Shift Registers

The cascadable16-bit shift register implements any static length mode shift register without the dedicated multiplexers (MUXF5, MUXF6,…). Figure 3-63 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.)

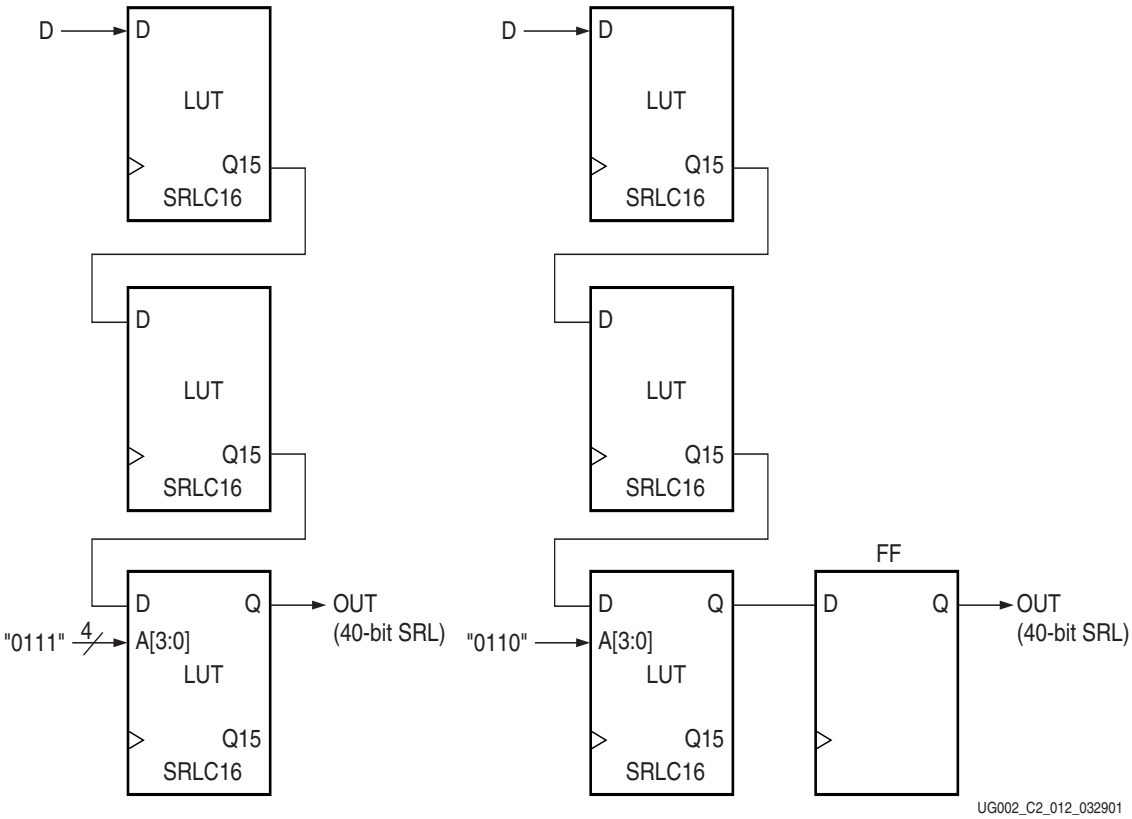


Figure 3-63: **40-bit Static-Length Shift Register**

VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available for all primitives and submodules.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

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

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

VHDL and Verilog Templates

In template names, the number indicates the number of bits (for example, SHIFT_REGISTER_16 is the template for the 16-bit shift register) and the "C" extension means the template is cascadable.

The following are templates for primitives:

- SHIFT_REGISTER_16
- SHIFT_REGISTER_16_C

The following are templates for submodules:

- SHIFT_REGISTER_32_C (submodule: SRLC32E_SUBM)
- SHIFT_REGISTER_64_C (submodule: SRLC64E_SUBM)
- SHIFT_REGISTER_128_C (submodule: SRLC128E_SUBM)

The corresponding submodules have to be synthesized with the design.

Templates for the SHIFT_REGISTER_16_C module are provided in VHDL and Verilog code as an example.

VHDL Template:

```
-- Module: SHIFT_REGISTER_C_16
-- Description: VHDL instantiation template
--  CASCADABLE 16-bit shift register with enable (SRLC16E)
-- Device: Virtex-II Family
-----------------------------------------------------------------------
-- Components Declarations:
--
component SRLC16E
-- pragma translate_off
  generic (
-- Shift Register initialization ("0" by default) for functional
simulation:
        INIT : bit_vector := X"0000"
  );
-- pragma translate_on
  port (
         D : in std_logic;
        CE   : in std_logic;
        CLK  : in std_logic;
        A0   : in std_logic;
        A1   : in std_logic;
        A2   : in std_logic;
        A3   : in std_logic;
        Q    : out std_logic;
        Q15  : out std_logic
  );
end component;
```

```
-- Architecture  Section:
--
-- Attributes for Shift Register initialization (“0” by default):
attribute INIT: string;
--
attribute INIT of U_SRLC16E: label is “0000”;
--
-- ShiftRegister Instantiation
U_SRLC16E: SRLC16E
  port map (
  D      => , -- insert input signal
  CE     => , -- insert Clock Enable signal (optional)
  CLK    => , -- insert Clock signal
  A0     => , -- insert Address 0 signal
  A1     => , -- insert Address 1 signal
  A2     => , -- insert Address 2 signal
  A3     => , -- insert Address 3 signal
  Q      => , -- insert output signal
  Q15    =>   -- insert cascadable output signal
  );
```

Verilog Template:

```
// Module: SHIFT_REGISTER_16
// Description: Verilog instantiation template
// Cascadable 16-bit Shift Register with Clock Enable (SRLC16E)
// Device: Virtex-II Family
//-----------------------------------------------------------------------
// Syntax for Synopsys FPGA Express
// synopsys translate_off

  defparam

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

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

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

Designing Large Multiplexers

Introduction

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

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

This section provides generic VHDL and Verilog reference code implementing multiplexers. These submodules are built from LUTs and the dedicated MUXF5, MUXF6, MUXF7, and MUXF8 multiplexers. To automatically generate large multiplexers using these dedicated elements, use the CORE Generator Bit Multiplexer and Bus Multiplexer modules.

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

Virtex-II CLB Resources

Slice Multiplexers

Each Virtex-II slice has a MUXF5 to combine the outputs of the 2 LUTs and an extra MUXFX. Figure 3-64 illustrates any single combinatorial function with up to 5 inputs, or in some functions up to 9 inputs, within one slice.

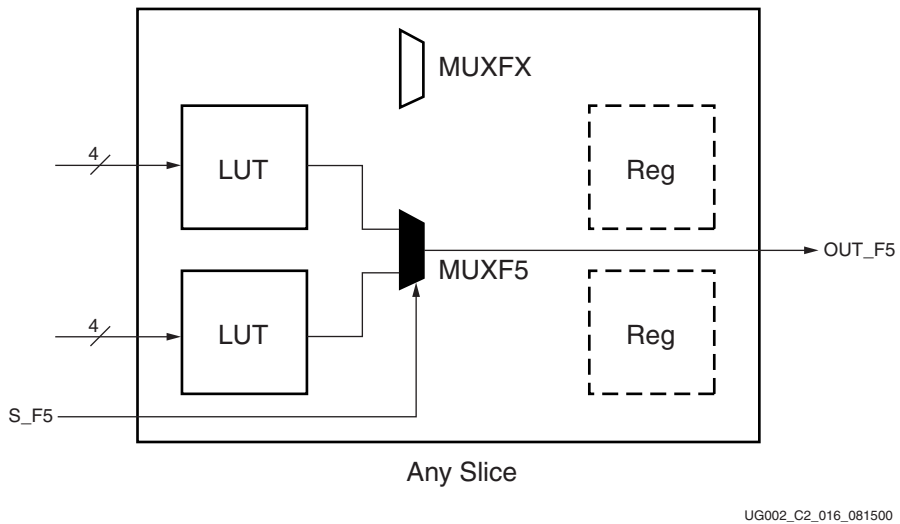


Figure 3-64: **LUTs and MUXF5 in a Slice**

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

Figure 3-65 shows the relative position of the slices in the CLB.

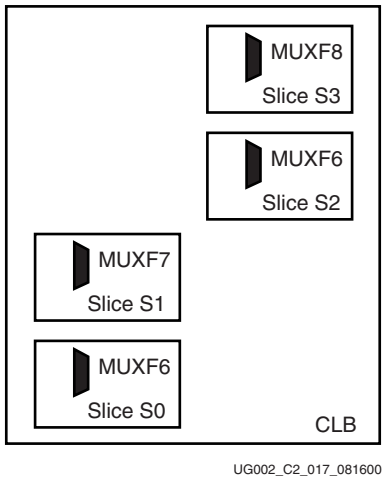


Figure 3-65: **Slice Positions in a CLB**

Slices S0 and S2 have a MUXF6, designed to combine the outputs of two MUXF5 resources. Figure 3-66 illustrates any single combinatorial function up to 6 inputs, or in some functions up to 18 inputs, in slices S0 and S1, or in slices S2 and S3.

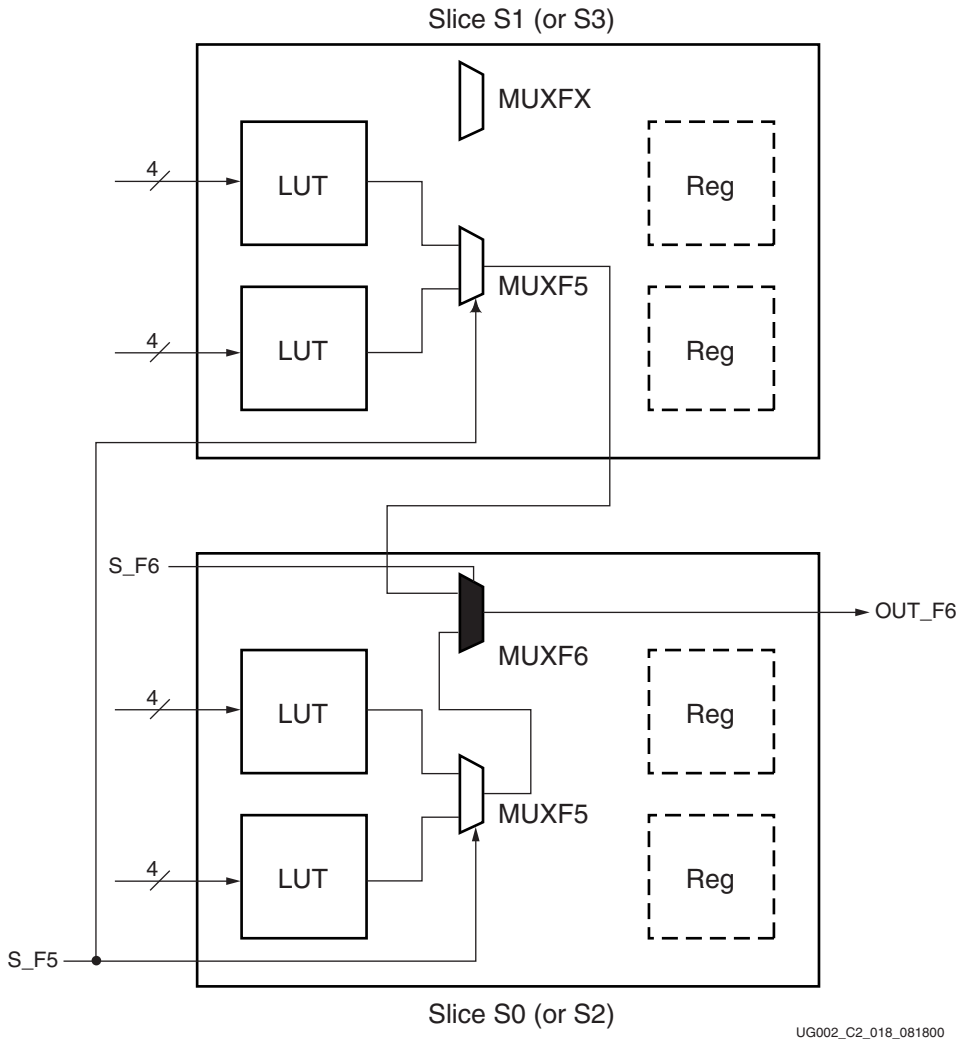


Figure 3-66: **LUTs and (MUXF5 and MUXF6) in Two Slices**

The slice S1 has a MUXF7, designed to combine the outputs of two MUXF6. Figure 3-67 illustrates any single combinatorial function up to 7 inputs, or in some functions up to 35 inputs, in a Virtex-II CLB.

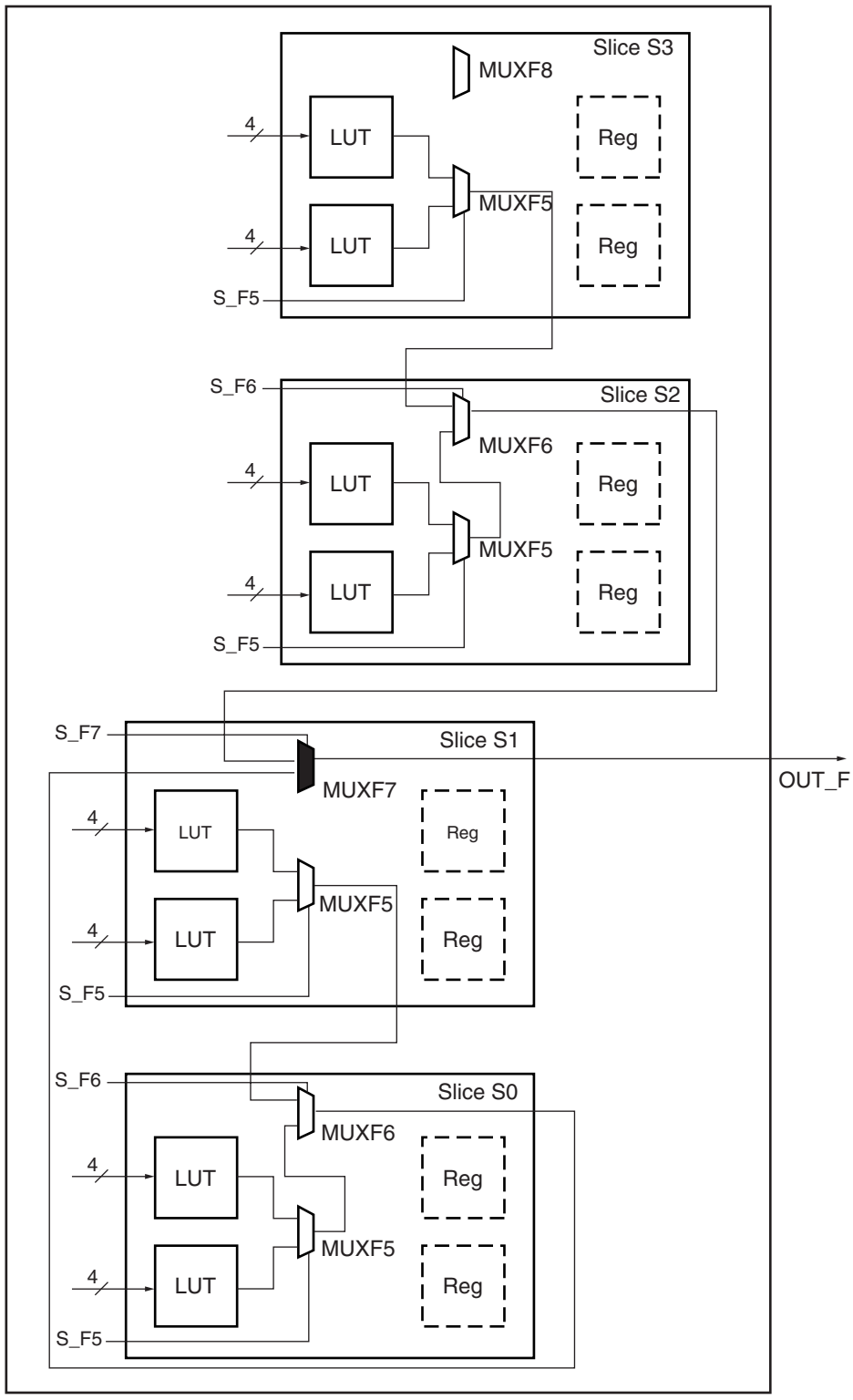


Figure 3-67: **LUTs and (MUXF5, MUXF6, and MUXF7) in One CLB**

The slice S3 of each CLB has a MUXF8. Any single combinatorial functions of up to 8 inputs, or some functions up to 68 inputs, fit in two CLBs as shown in Figure 3-68. The outputs of two MUXF7 are combined through dedicated routing resources between two adjacent CLBs in a column.

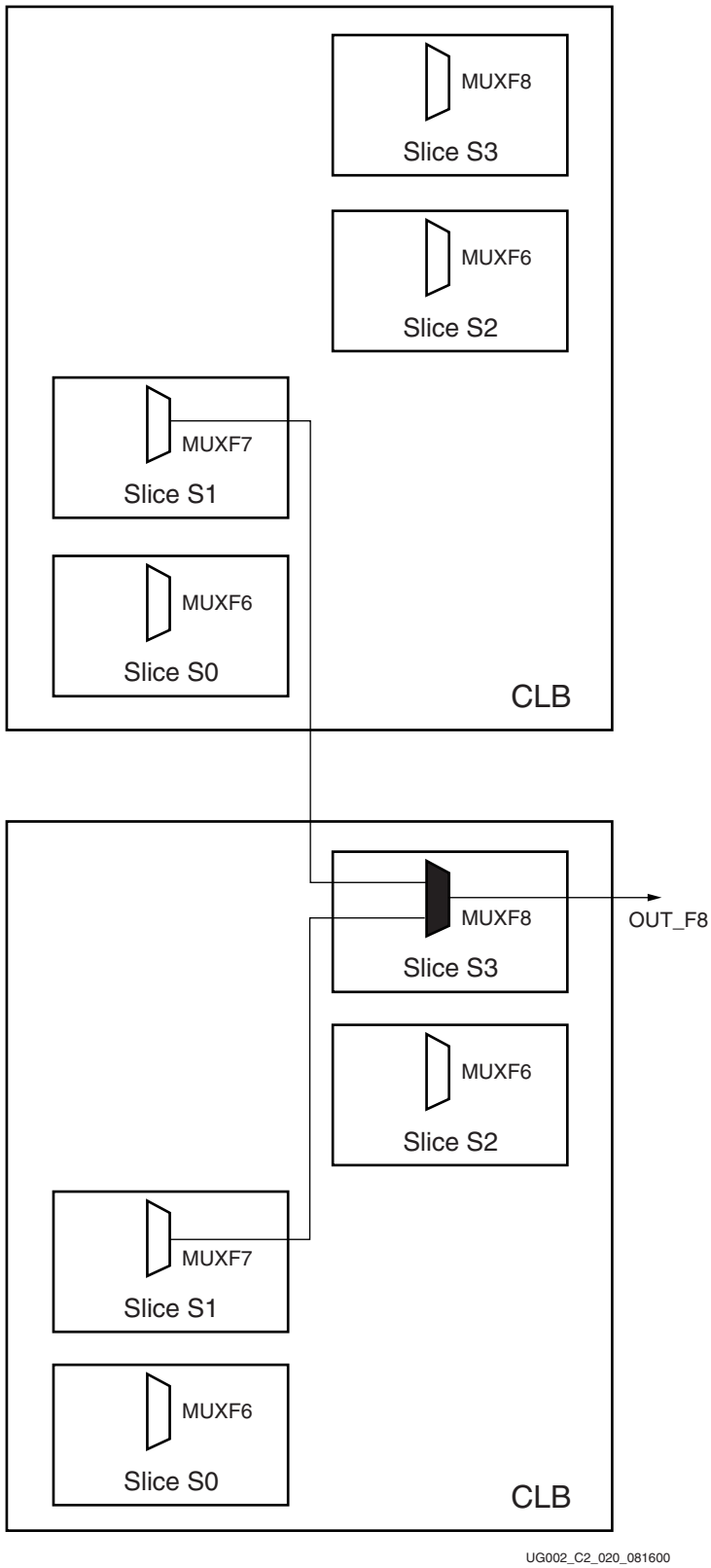


Figure 3-68: **MUXF8 Combining Two Adjacent CLBs**

Wide-Input Multiplexers

Each LUT can implement a 2:1 multiplexer. In each slice, the MUXF5 and two LUTs can implement a 4:1 multiplexer. As shown in Figure 3-69, the MUXF6 and two slices can implement a 8:1 multiplexer. The MUFXF7 and the four slices of any CLB can implement a 16:1 and the MUXF8 and two CLBs can implement a 32:1 multiplexer.

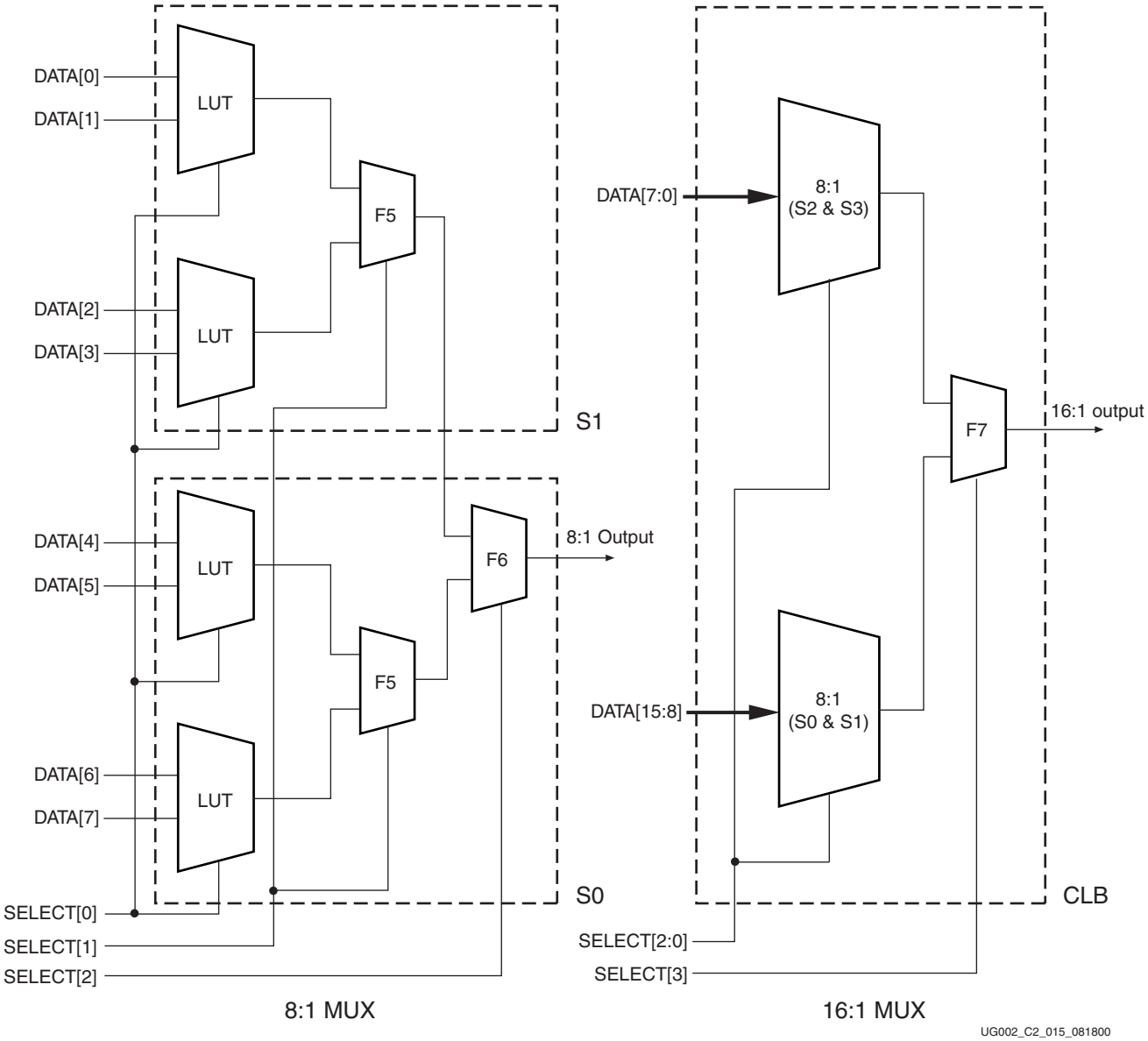


Figure 3-69: **8:1 and 16:1 Multiplexers**

Characteristics

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

Library Primitives and Submodules

Four library primitives are available that offer access to the dedicated MUXFX in each slice.

In the example shown in Table 3-23, MUXF7 is available only in slice S1.

Table 3-23: **MUXFX Resources**

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

In addition to the primitives, five submodules that implement multiplexers from 2:1 to 32:1 are provided in VHDL and Verilog code. Synthesis tools can automatically infer the above primitives (MUXF5, MUXF6, MUXF7, and MUXF8); however, the submodules described in this section used instantiation of the new MUXFX to guarantee an optimized result. Table 3-24 lists available submodules:

Table 3-24: **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 3-25 shows the DATA_I selected for each SELECT_I value.

Table 3-25: **Selected Inputs**

SELECT_I[1:0]	DATA_O
0 0	DATA_I[0]
0 1	DATA_I[1]
1 0	DATA_I[2]
1 1	DATA_I[3]

Data Out - DATA_O

The data output O provides the data value (1 bit) selected by the control inputs.

Applications

Multiplexers are used in various applications. These are often inferred by synthesis tools when a "case" statement is used (see the example below). Comparators, encoder-decoders and wide-input combinatorial functions are optimized when they are based on one level of LUTs and dedicated MUXFX resources of the Virtex-II CLBs.

VHDL and Verilog Instantiation

The primitives (MUXF5, MUXF6, and so forth) can be instantiated in VHDL or Verilog code, to design wide-input functions.

The submodules (MUX_2_1_SUBM, MUX_4_1_SUBM, and so forth) can be instantiated in VHDL or Verilog code to implement multiplexers. However the corresponding submodule must be added to the design directory as hierarchical submodule. For example, if a module is using the MUX_16_1_SUBM, the MUX_16_1_SUBM.vhd file (VHDL code) or MUX_16_1_SUBM.v file (Verilog code) must be compiled with the design source code. The submodule code can also be "cut and pasted" into the designer source code.

VHDL and Verilog Submodules

VHDL and Verilog submodules are available to implement multiplexers up to 32:1. They illustrate how to design with the MUXFX resources. When synthesis infers the corresponding MUXFX resource(s), the VHDL or Verilog code is behavioral code ("case" statement). Otherwise, the equivalent "case" statement is provided in comments and the correct MUXFX are instantiated. However, most synthesis tools support the inference of all of the MUXFX. The following examples can be used as guidelines for designing other wide-input functions.

The following submodules are available:

- MUX_2_1_SUBM (behavioral code)
- MUX_4_1_SUBM
- MUX_8_1_SUBM
- MUX_16_1_SUBM
- MUX_32_1_SUBM

The corresponding submodules have to be synthesized with the design

The submodule MUX_16_1_SUBM in VHDL and Verilog are provided as example.

VHDL Template

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

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

entity MUX_16_1_SUBM is
    port (
        DATA_I: in std_logic_vector (15 downto 0);
        SELECT_I: in std_logic_vector (3 downto 0);
        DATA_O: out std_logic
```

```
  );
end MUX_16_1_SUBM;

architecture MUX_16_1_SUBM_arch of MUX_16_1_SUBM is
-- Component Declarations:
component MUXF7
    port (
    I0: in std_logic;
    I1: in std_logic;
    S: in std_logic;
    O: out std_logic
    );
end component;
--
-- Signal Declarations:
signal DATA_MSB : std_logic;
signal DATA_LSB : std_logic;
--
begin
--
-- If synthesis tools support MUXF7 :
--SELECT_PROCESS: process (SELECT_I, DATA_I)
--begin
--case SELECT_I is
--  when "0000" => DATA_O <= DATA_I (0);
--  when "0001" => DATA_O <= DATA_I (1);
--  when "0010" => DATA_O <= DATA_I (2);
--  when "0011" => DATA_O <= DATA_I (3);
--  when "0100" => DATA_O <= DATA_I (4);
--  when "0101" => DATA_O <= DATA_I (5);
--  when "0110" => DATA_O <= DATA_I (6);
--  when "0111" => DATA_O <= DATA_I (7);
--  when "1000" => DATA_O <= DATA_I (8);
--  when "1001" => DATA_O <= DATA_I (9);
--  when "1010" => DATA_O <= DATA_I (10);
--  when "1011" => DATA_O <= DATA_I (11);
--  when "1100" => DATA_O <= DATA_I (12);
--  when "1101" => DATA_O <= DATA_I (13);
--  when "1110" => DATA_O <= DATA_I (14);
--  when "1111" => DATA_O <= DATA_I (15);
--  when others => DATA_O <= 'X';
--end case;
--end process SELECT_PROCESS;
--
-- If synthesis tools DO NOT support MUXF7 :
SELECT_PROCESS_LSB: process (SELECT_I, DATA_I)
begin
  case SELECT_I (2 downto 0) is
    when "000" => DATA_LSB <= DATA_I (0);
    when "001" => DATA_LSB <= DATA_I (1);
    when "010" => DATA_LSB <= DATA_I (2);
    when "011" => DATA_LSB <= DATA_I (3);
    when "100" => DATA_LSB <= DATA_I (4);
    when "101" => DATA_LSB <= DATA_I (5);
    when "110" => DATA_LSB <= DATA_I (6);
    when "111" => DATA_LSB <= DATA_I (7);
    when others => DATA_LSB <= 'X';
  end case;
end process SELECT_PROCESS_LSB;
--
SELECT_PROCESS_MSB: process (SELECT_I, DATA_I)
begin
```

```
  case SELECT_I (2 downto 0) is
    when "000" => DATA_MSB <= DATA_I (8);
    when "001" => DATA_MSB <= DATA_I (9);
    when "010" => DATA_MSB <= DATA_I (10);
    when "011" => DATA_MSB <= DATA_I (11);
    when "100" => DATA_MSB <= DATA_I (12);
    when "101" => DATA_MSB <= DATA_I (13);
    when "110" => DATA_MSB <= DATA_I (14);
    when "111" => DATA_MSB <= DATA_I (15);
    when others => DATA_MSB <= 'X';
  end case;
end process SELECT_PROCESS_MSB;
--
-- MUXF7 instantiation
U_MUXF7: MUXF7
    port map (
    I0 => DATA_LSB,
    I1 => DATA_MSB,
    S  => SELECT_I (3),
    O  => DATA_O
    );
--
end MUX_16_1_SUBM_arch;
--
```

Verilog Template

```
// Module: MUX_16_1_SUBM
//
// Description: Multiplexer 16:1
// Device: Virtex-II Family
//-----------------------------------------------------------------------
//
module MUX_16_1_SUBM (DATA_I, SELECT_I, DATA_O);

input [15:0]DATA_I;
input [3:0]SELECT_I;

output DATA_O;

wire [2:0]SELECT;

reg DATA_LSB;
reg DATA_MSB;

assign SELECT[2:0] = SELECT_I[2:0];

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

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

```
  4'b1011 : DATA_O <= DATA_I[11];
        4'b1100 : DATA_O <= DATA_I[12];
  4'b1101 : DATA_O <= DATA_I[13];
  4'b1110 : DATA_O <= DATA_I[14];
  4'b1111 : DATA_O <= DATA_I[15];
  default : DATA_O <= 1'bx;
    endcase
*/

always @ (SELECT or DATA_I)

    case (SELECT)
        3'b000 : DATA_LSB <= DATA_I[0];
  3'b001 : DATA_LSB <= DATA_I[1];
  3'b010 : DATA_LSB <= DATA_I[2];
  3'b011 : DATA_LSB <= DATA_I[3];
        3'b100 : DATA_LSB <= DATA_I[4];
  3'b101 : DATA_LSB <= DATA_I[5];
  3'b110 : DATA_LSB <= DATA_I[6];
  3'b111 : DATA_LSB <= DATA_I[7];
  default : DATA_LSB <= 1'bx;
    endcase

always @ (SELECT or DATA_I)

    case (SELECT)
        3'b000 : DATA_MSB <= DATA_I[8];
  3'b001 : DATA_MSB <= DATA_I[9];
  3'b010 : DATA_MSB <= DATA_I[10];
  3'b011 : DATA_MSB <= DATA_I[11];
        3'b100 : DATA_MSB <= DATA_I[12];
  3'b101 : DATA_MSB <= DATA_I[13];
  3'b110 : DATA_MSB <= DATA_I[14];
  3'b111 : DATA_MSB <= DATA_I[15];
  default : DATA_MSB <= 1'bx;
    endcase

// MUXF7 instantiation

MUXF7 U_MUXF7   (.I0(DATA_LSB),
     .I1(DATA_MSB),
     .S(SELECT_I[3]),
     .O(DATA_O)
     );
endmodule

//
*/
```

Implementing Sum of Products (SOP) Logic

Introduction

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

Virtex-II CLB Resources

Each Virtex-II slice has a MUXCY, which uses the output from the LUTs as a SELECT signal. Depending on the width of data desired, several slices can be used to provide the SOP output. Figure 3-70 illustrates the logic involved in designing a 16-input AND gate. It utilizes the 4-input LUT to provide the necessary SELECT signal for the MUXCY. Only when all of the input signals are High, can the V_{CC} at the bottom reach the output. This use of carry logic helps to perform AND functions at high speed and saves logic resources.

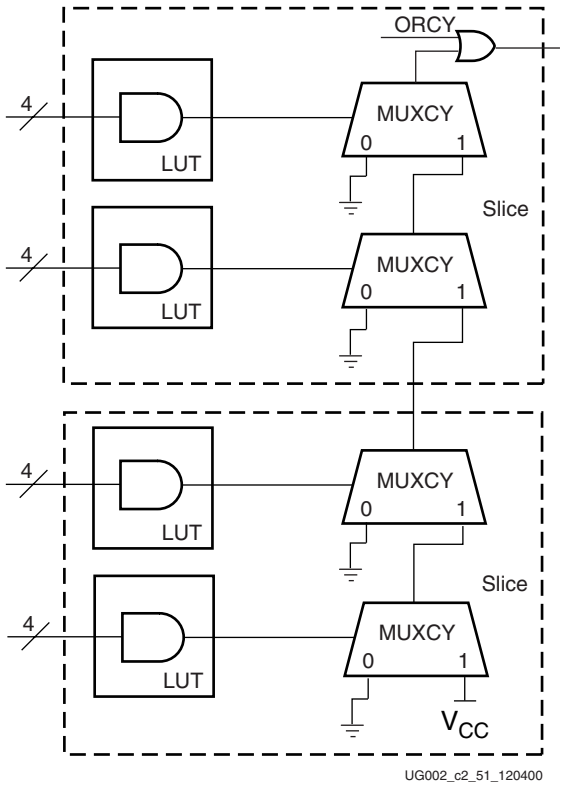


Figure 3-70: **Implementing a 16-bit Wide AND Gate Using MUXCY & ORCY**

The output from the chain of AND gates is passed as one of the inputs of the dedicated OR gate, ORCY. To calculate the SOP, these AND chains can be cascaded vertically across several CLBs, depending on the width of the input data. Figure 3-71 illustrates how the AND outputs are then passed in through the ORCY gates in a horizontal cascade, the sum of which is the Sum of Products.

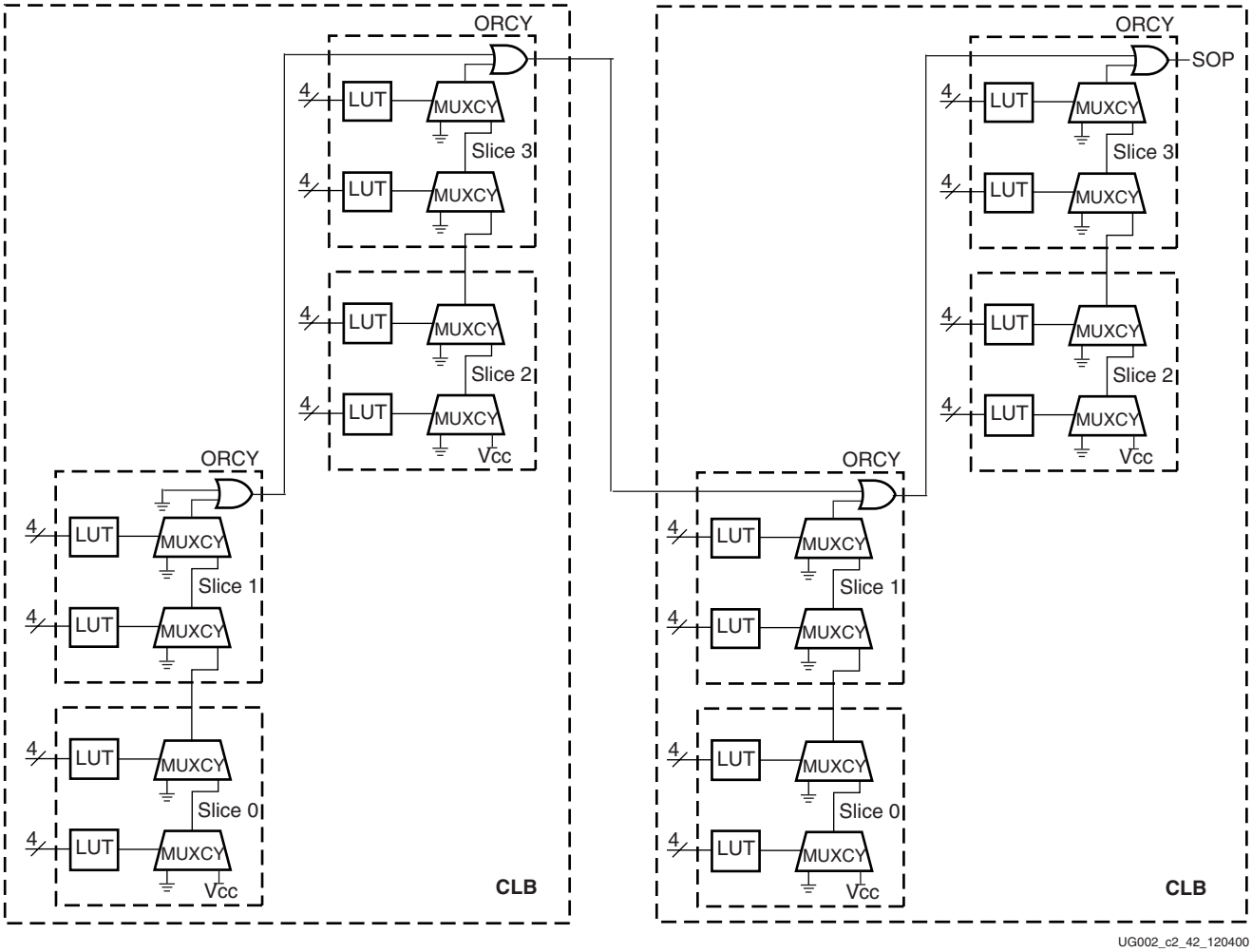


Figure 3-71: **64-bit Input SOP Design**

VHDL Parameters

AND_WIDTH Parameter

The width of each AND gate used in the cascade.

PROD_TERM Parameter

The number of AND gates used along each vertical cascade.

AND_IN Parameter

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

SOP_OUT Parameter

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

Applications

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

VHDL and Verilog Instantiation

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

VHDL and Verilog Submodules

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

VHDL Templates

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

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

architecture AND_CHAIN_arch of AND_CHAIN is

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

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

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

begin

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

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

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

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

out_and_chain <= cout(input_width/4);

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

end AND_CHAIN_arch;

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

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

architecture AND_LOGIC_arch of AND_LOGIC is

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

signal GND : std_logic;
signal sel:std_logic;

begin

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

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

end AND_LOGIC_arch;
```

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

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

architecture SOP_SUBM_arch of SOP_SUBM is

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

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

begin

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

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

end SOP_SUBM_arch;
```

Verilog Templates

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

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

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

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

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

Using Embedded Multipliers

Introduction

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

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

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

Two's-Complement Signed Multiplier

Data Flow

Each embedded multiplier block (MULT18X18 primitive) supports two independent dynamic data input ports: 18-bit signed or 17-bit unsigned. The MULT18X18 primitive is illustrated in Figure 3-72.

In addition, efficient cascading of multipliers up to 35-bit X 35-bit signed can be accomplished by using 4 embedded multipliers, one 36-bit adder, and one 53-bit adder. See Figure 3-73.

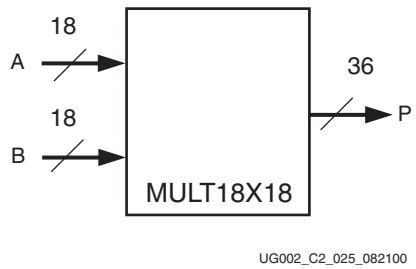


Figure 3-72: **Embedded Multiplier**

Library Primitives and Submodules

One library primitive (MULT18X18) is available. Table 3-26 lists the attributes of this primitive.

Table 3-26: **Embedded Multiplier Primitive**

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

In addition to the primitive, 15 submodules that implement various widths of signed and unsigned multipliers and two's-complement return functions are provided in VHDL and Verilog code. Multipliers using cascaded MULT18X18 primitives are included with registers between stages causing three cycles of latency. Multipliers that make use of the embedded Virtex-II 18-bit by 18-bit two's complement multipliers can be easily generated using V2.0 of the CORE Generator Multiplier module. Table 3-27 lists cascaded multiplier submodules.

Table 3-27: **Embedded Multiplier Submodules - Cascaded MULT18X18**

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

Figure 3-73 represents the cascaded scheme used to implement a 35-bit by 35-bit signed multiplier utilizing four embedded multipliers and two adders.

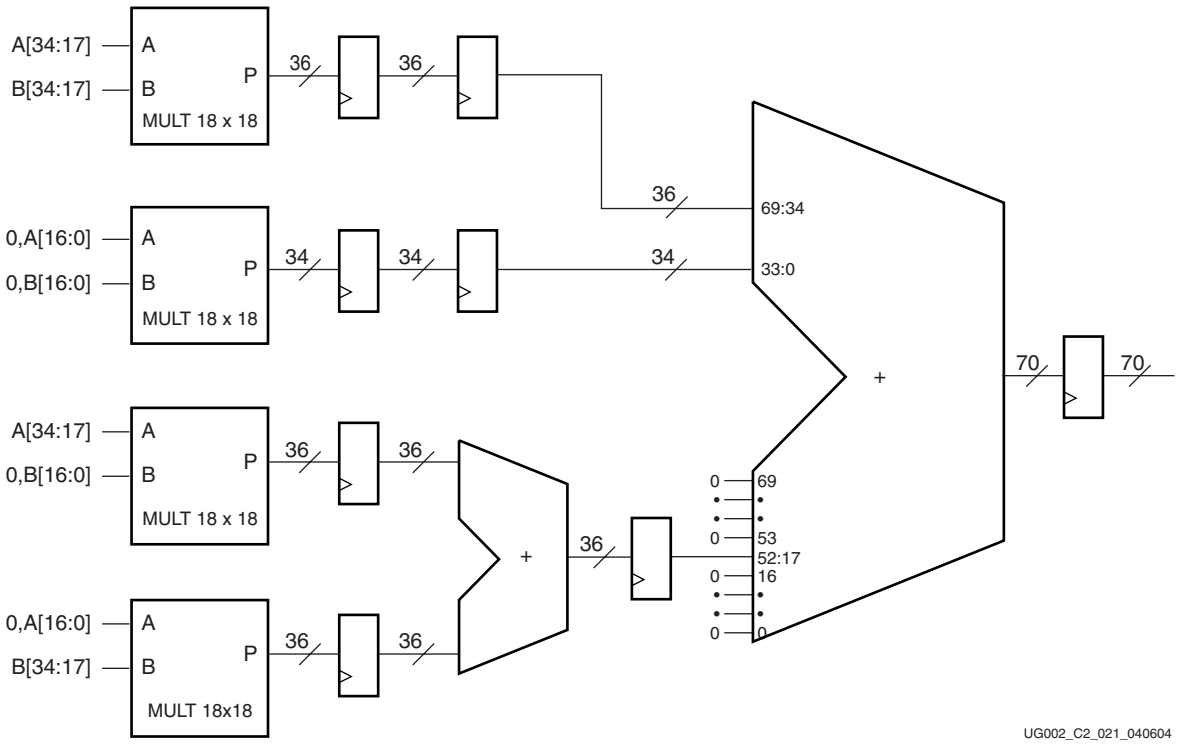


Figure 3-73: **MULT35X35_S Submodule**

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

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

Table 3-27 lists multipliers and two's-complement return functions that utilize one MULT18X18 primitive and are not registered.

Table 3-28: **Embedded Multiplier Submodules - Single MULT18X18**

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

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

Figure 3-74 and Figure 3-75 represent 4-bit by 4-bit signed multiplier and 4-bit by 4-bit unsigned multiplier implementations, respectively.

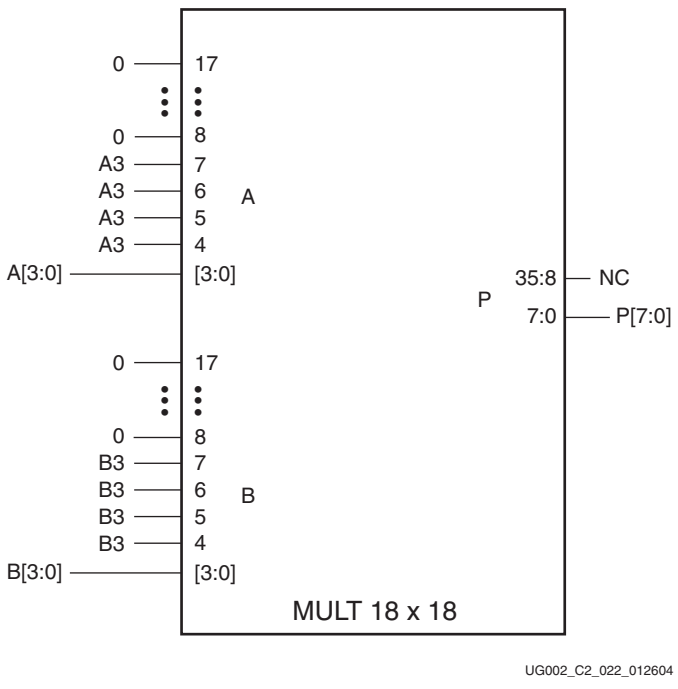


Figure 3-74: **MULT4X4_S Submodule**

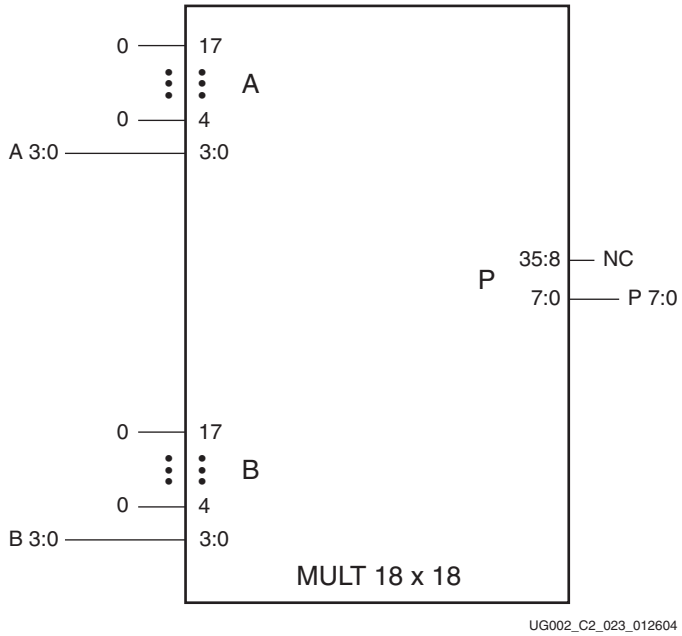


Figure 3-75: **MULT4X4_U Submodule**

Submodule MAGNTD_18 performs a magnitude return (i.e., absolute value) of a two's-complement number. An incoming negative number returns with a positive number, while an incoming positive number remains unchanged. Submodules TWOS_CMP18 and TWOS_CMP9 perform a two's-complement return function. The incoming number in two's-complement form (either signed or unsigned) is complemented when the DO_COMP pin is asserted High. Additional slice logic can be used with these submodules to efficiently convert sign-magnitude to two's-complement or vice-versa. Figure 3-76 shows the connections to a MULT18X18 to create the submodule TWOS_CMP9.

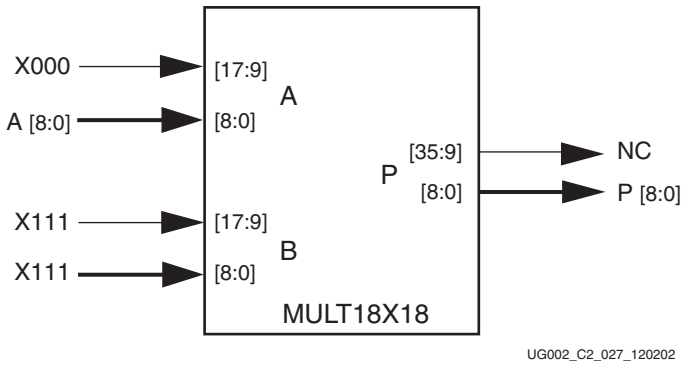


Figure 3-76: **TWOS_CMP9 Submodule**

VHDL and Verilog Instantiation

VHDL and Verilog instantiation templates are available as examples of primitives and sub-modules (see "VHDL and Verilog Templates" on page 172).

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signals names.

Port Signals

Data In - A

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

Data In - B

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

Data Out - P

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

Location Constraints

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

MULT18X18 embedded multiplier instances can have LOC properties attached to them to constrain placement. MULT18X18 placement locations differ from the convention used for naming CLB locations, allowing LOC properties to transfer easily from array to array.

The LOC properties use the following form:

LOC = MULT18X18_X#Y#

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

Routing with BlockRAM

The following input pins are shared among adjacent BlockRAM and multiplier :

- RAMB16 MULT18x18
- DIA16 A0
- DIA17 A1
- DIA18 A2
- DIA19 A3
- DIA20 A4
- DIA21 A5
- DIA22 A6
- DIA23 A7
- DIA24 A8
- DIA25 A9
- DIA26 A10
- DIA27 A11
- DIA28 A12
- DIA29 A13
- DIA30 A14
- DIA31 A15
- DIB16 B0

- DIB17 B1
- DIB18 B2
- DIB19 B3
- DIB20 B4
- DIB21 B5
- DIB22 B6
- DIB23 B7
- DIB24 B8
- DIB25 B9
- DIB26 B10
- DIB27 B11
- DIB28 B12
- DIB29 B13
- DIB30 B14
- DIB31 B15

If the above BlockRAM and multiplier pins do not have common source, the implementation tools will not place the blocks adjacently.

VHDL and Verilog Templates

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

The following is a template for the primitive:

- SIGNED_MULT_18X18 (primitive: MULT18X18)

The following are templates for submodules:

- SIGNED_MULT_35X35 (submodule: MULT35X35_S)
- UNSIGNED_MULT_34X34 (submodule: MULT34X34_U)
- UNSIGNED_MULT_17X17 (submodule: MULT17X17_U)
- SIGNED_MULT_8X8 (submodule: MULT8X8_S)
- UNSIGNED_MULT_8X8 (submodule: MULT8X8_U)
- SIGNED_MULT_4X4 (submodule: MULT4X4_S)
- UNSIGNED_MULT_4X4 (submodule: MULT4X4_U)
- TWOS_COMPLEMENTER_18BIT (submodule: TWOS_CMP18)
- TWOS_COMPLEMENTER_9BIT (submodule: TWOS_CMP9)
- MAGNITUDE_18BIT (submodule: MAGNTD_18)

The corresponding submodules have to be synthesized with the design.

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

VHDL Template:

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

Verilog Template:

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

Using Single-Ended SelectI/O-Ultra Resources

Summary

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

Introduction

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

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

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

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

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

Fundamentals

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

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

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

Table 3-29 illustrates all of the supported single-ended I/O standards in Virtex-II devices. Each buffer type can support a variety of current and voltage requirements.

Table 3-29: **Supported Single-Ended I/O Standards**

I/O Standard	Input Reference Voltage (V_{REF})	Input Source Voltage (V_{CCO})	Output Source Voltage (V_{CCO})	Board Termination Voltage (V_{TT})
LVTTL	N/A	3.3	3.3	N/A
LVCMOS15	N/A	1.5	1.5	N/A
LVCMOS18	N/A	1.8	1.8	N/A
LVCMOS25	N/A	2.5	2.5	N/A
LVCMOS33	N/A	3.3	3.3	N/A
PCI33_3	N/A	3.3	3.3	N/A
PCI66_3	N/A	3.3	3.3	N/A
PCIX	N/A	3.3	3.3	N/A
GTL	0.80	Note (1)	Note (1)	1.2
GTL+	1.0	Note (1)	Note (1)	1.5
HSTL_I	0.75	N/A	1.5	0.75
HSTL_II	0.75	N/A	1.5	0.75
HSTL_III	0.9	N/A	1.5	1.5
HSTL_IV	0.9	N/A	1.5	1.5
HSTL_I_18	0.9	N/A	1.8	0.9
HSTL_II_18	0.9	N/A	1.8	0.9
HSTL_III_18	1.1	N/A	1.8	1.8
HSTL_IV_18	1.1	N/A	1.8	1.8
SSTL3_I	1.5	N/A	3.3	1.5
SSTL3_II	1.5	N/A	3.3	1.5
SSTL2_I	1.25	N/A	2.5	1.25
SSTL2_II	1.25	N/A	2.5	1.25
SSTL18_I(2)	1.8	N/A	0.9	0.9
SSTL18_II	1.8	N/A	0.9	0.9
AGP-2X	1.32	N/A	3.3	N/A

Notes:

1. V_{CCO} of GTL or GTL+ should not be lower than the termination voltage or the voltage seen at the I/O pad. For example, connect V_{CCO} to 1.5V if the pin High level is 1.5V
2. SSTL18_I is not a JEDEC-supported standard.

Overview of Supported I/O Standards

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

While most I/O standards specify a range of allowed voltages, this document records typical voltage values only. Detailed information on each specification can be found on the Electronic Industry Alliance JEDEC website at: http://www.jedec.org

LVTTL - Low-Voltage TTL

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

LVCMOS33 - 3.3 Volt Low-Voltage CMOS

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

LVCMOS25 - 2.5 Volt Low-Voltage CMOS

This standard is an extension of the LVCMOS standard (JESD 8.-5). It is used in general purpose 2.5 volts or lower applications. This standard requires a 2.5 V input /output source voltage (V_{CCO}), but does not require the use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}).

LVCMOS18 - 1.8 Volt Low-Voltage CMOS

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

LVCMOS15 - 1.5 Volt Low-Voltage CMOS

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

PCI - Peripheral Component Interface

The PCI standard specifies support for 33 MHz, 66 MHz and 133 MHz PCI bus applications. It uses a LVTTL input buffer and a Push-Pull output buffer. This standard does not require the use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}), however, it does require 3.3 V input output source voltage (V_{CCO}).

GTL -Gunning Transceiver Logic Terminated

The GTL standard is a high-speed bus standard (JESD8.3) invented by Xerox. Xilinx has implemented the terminated variation for this standard. This standard requires a differential amplifier input buffer and a open Drain output buffer.

GTL+ - Gunning Transceiver Logic Plus

The Gunning Transceiver Logic Plus, or GTL+ standard is a high-speed bus standard (JESD8.3) first used by the Pentium Pro Processor.

HSTL - High-speed Transceiver Logic

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

SSTL3 - Stub Series Terminated Logic for 3.3V

The Stub Series Terminated Logic for 3.3V, or SSTL3 standard is a general purpose 3.3V memory bus standard also sponsored by Hitachi and IBM (JESD8-8). This standard has

two classes, I and II. Virtex-II SelectI/O-Ultra supports both classes for the SSTL3 standard. This standard requires a Differential Amplifier input buffer and a Push-Pull output buffer.

SSTL2 - Stub Series Terminated Logic for 2.5V

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

SSTL18 - Stub Series Terminated Logic for 1.8V

The Stub Series Terminated Logic for 1.8V, or SSTL18 is a general purpose 1.8V memory bus standard. Similar to SSTL2, it has two classes, I and II. (Only Class II is supported by JESD8-15.) Virtex-II SelectIO-Ultra supports both classes for the SSTL18 standard. This standard requires a differential amplifier input buffer and a push-pull output buffer.

AGP-2X - Advanced Graphics Port

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

Library Symbols

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

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

IBUF

Signals used as inputs to a Virtex-II device must source an input buffer (IBUF) via an external input port. The generic Virtex-II IBUF symbol is shown in Figure 3-77. The extension to the base name defines which I/O standard the IBUF uses. The assumed standard is LVTTL when the generic IBUF has no specified extension.

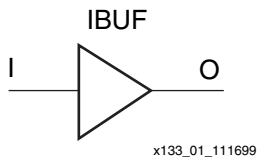


Figure 3-77: **Input Buffer (IBUF) Symbols**

Table 3-30 details variations of the IBUF symbol for single-ended Virtex-II I/O standards:

Table 3-30: **Variations of the IBUF Symbol**

IBUF	IBUF_HSTL_IV
IBUF_LVCMOS15	IBUF_SSTL18_I
IBUF_LVCMOS18	IBUF_SSTL18_II

Table 3-30: **Variations of the IBUF Symbol**

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

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

The voltage reference signal is "banked" within the Virtex-II device on a half-edge basis, such that for all packages there are eight independent V_{REF} banks internally. For a representation of the Virtex-II I/O banks, see Figure 3-79. Within each bank approximately one of every six I/O pins is automatically configured as a V_{REF} input. After placing a differential amplifier input signal within a given V_{REF} bank, the same external source must drive all I/O pins configured as a V_{REF} input.

IBUF placement restrictions require that any differential amplifier input signals within a bank be of the same standard. How to specify a specific location for the IBUF via the LOC property is described below. Table 3-31 summarizes compatibility requirements of Virtex-II input standards.

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

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

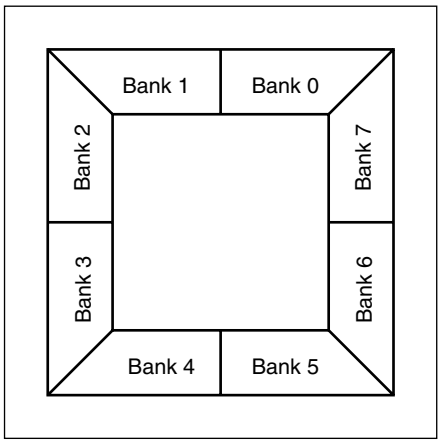


Figure 3-78: **Virtex-II I/O Banks: Top View for Flip-Chip Packages (FF & BF)**

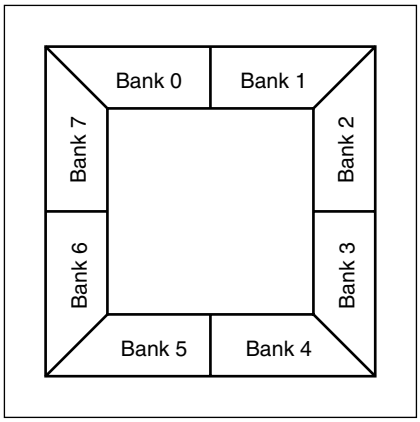


Figure 3-79: **Virtex-II I/O Banks: Top View for Wire-Bond Packages (CS, FG, & BG)**

Table 3-31: **Xilinx Input Standard Compatibility Requirements**

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

Each bank has its own V_{CCO} and V_{REF} voltage. Details on compatible input standards for each V_{CCO} / V_{REF} voltage combination are available in the **Virtex-II Data Sheet (DS031)** .

OBUF

An OBUF must drive outputs through an external output port. Figure 3-80 shows the generic output buffer (OBUF) symbol.

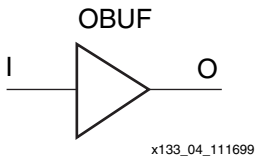


Figure 3-80: **Virtex-II Output Buffer (OBUF) Symbol**

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

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

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

OBUF_<slew_rate>_<drive_strength>

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

Table 3-32 details variations of the OBUF symbol.

Table 3-32: **Variations of the OBUF Symbol**

OBUF	OBUF_LVCMOS18_S_2	OBUF_LVCMOS33_S_4
OBUF_S_2	OBUF_LVCMOS18_S_4	OBUF_LVCMOS33_S_6
OBUF_S_4	OBUF_LVCMOS18_S_6	OBUF_LVCMOS33_S_8
OBUF_S_6	OBUF_LVCMOS18_S_8	OBUF_LVCMOS33_S_12
OBUF_S_8	OBUF_LVCMOS18_S_12	OBUF_LVCMOS33_S_16
OBUF_S_12	OBUF_LVCMOS18_S_16	OBUF_LVCMOS33_S_24
OBUF_S_16	OBUF_LVCMOS18_F_2	OBUF_LVCMOS33_F_2
OBUF_S_24	OBUF_LVCMOS18_F_4	OBUF_LVCMOS33_F_4
OBUF_F_2	OBUF_LVCMOS18_F_6	OBUF_LVCMOS33_F_6
OBUF_F_4	OBUF_LVCMOS18_F_8	OBUF_LVCMOS33_F_8
OBUF_F_6	OBUF_LVCMOS18_F_12	OBUF_LVCMOS33_F_12
OBUF_F_8	OBUF_LVCMOS18_F_16	OBUF_LVCMOS33_F_16
OBUF_F_12	OBUF_LVCMOS25	OBUF_LVCMOS33_F_24
OBUF_F_16	OBUF_LVCMOS25_S_2	OBUF_PCI33_3
OBUF_F_24	OBUF_LVCMOS25_S_4	OBUF_PCI66-3
OBUF_LVCMOS15	OBUF_LVCMOS25_S_6	OBUF_PCIX
OBUF_LVCMOS15_S_2	OBUF_LVCMOS25_S_8	OBUF_GTL
OBUF_LVCMOS15_S_4	OBUF_LVCMOS25_S_12	OBUF_GTLP
OBUF_LVCMOS15_S_6	OBUF_LVCMOS25_S_16	OBUF_HSTL_I
OBUF_LVCMOS15_S_8	OBUF_LVCMOS25_S_24	OBUF_HSTL_II
OBUF_LVCMOS15_S_12	OBUF_LVCMOS25_F_2	OBUF_HSTL_III
OBUF_LVCMOS15_S_16	OBUF_LVCMOS25_F_4	OBUF_HSTL_IV
OBUF_LVCMOS15_F_2	OBUF_LVCMOS25_F_6	OBUF_SSTL3_I
OBUF_LVCMOS15_F_4	OBUF_LVCMOS25_F_8	OBUF_SSTL3_II
OBUF_LVCMOS15_F_6	OBUF_LVCMOS25_F_12	OBUF_SSTL2_I
OBUF_LVCMOS15_F_8	OBUF_LVCMOS25_F_16	OBUF_SSTL2_II
OBUF_LVCMOS15_F_12	OBUF_LVCMOS25_F_24	OBUF_SSTL18_I
OBUF_LVCMOS15_F_16	OBUF_LVCMOS33	OBUF_SSTL18_II
OBUF_LVCMOS18	OBUF_LVCMOS33_S_2	OBUF_AGP

OBUF placement restrictions require that within a given V_{CCO} bank each OBUF share the same output source drive voltage. Input buffers with the same V_{CCO} and output buffers that do not require V_{CCO} can be placed within any V_{CCO} bank. Table 3-33 summarizes Virtex-II output compatibility requirements. The LOC property can specify a location for the OBUF.

Table 3-33: **Output Standards Compatibility Requirements**

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

Each bank has its own V_{CCO} voltage. Details on compatible output standards for each V_{CCO} voltage combination are available in the **Virtex-II Data Sheet (DS031)**.

OBUFT

The generic 3-state output buffer OBUFT, shown in Figure 3-81, typically implements 3-state outputs or bidirectional I/O.

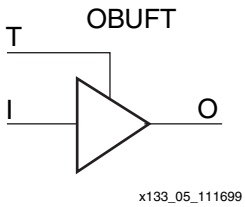


Figure 3-81: **3-State Output Buffer Symbol (OBUFT)**

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

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

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

OBUFT_<slew_rate>_<drive_strength>

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

Table 3-34 details variations of the OBUFT symbol.

Table 3-34: **Variations of the OBUFT Symbol**

OBUFT	OBUFT_LVCMOS18_S_2	OBUFT_LVCMOS33_S_4
OBUFT_S_2	OBUFT_LVCMOS18_S_4	OBUFT_LVCMOS33_S_6
OBUFT_S_4	OBUFT_LVCMOS18_S_6	OBUFT_LVCMOS33_S_8
OBUFT_S_6	OBUFT_LVCMOS18_S_8	OBUFT_LVCMOS33_S_12
OBUFT_S_8	OBUFT_LVCMOS18_S_12	OBUFT_LVCMOS33_S_16
OBUFT_S_12	OBUFT_LVCMOS18_S_16	OBUFT_LVCMOS33_S_24
OBUFT_S_16	OBUFT_LVCMOS18_F_2	OBUFT_LVCMOS33_F_2
OBUFT_S_24	OBUFT_LVCMOS18_F_4	OBUFT_LVCMOS33_F_4
OBUFT_F_2	OBUFT_LVCMOS18_F_6	OBUFT_LVCMOS33_F_6
OBUFT_F_4	OBUFT_LVCMOS18_F_8	OBUFT_LVCMOS33_F_8
OBUFT_F_6	OBUFT_LVCMOS18F_12	OBUFT_LVCMOS33_F_12
OBUFT_F_8	OBUFT_LVCMOS18_F_16	OBUFT_LVCMOS33_F_16
OBUFT_F_12	OBUFT_LVCMOS25	OBUFT_LVCMOS33_F_24
OBUFT_F_16	OBUFT_LVCMOS25_S_2	OBUFT_PCI33_3
OBUFT_F_24	OBUFT_LVCMOS25_S_4	OBUFT_PCI66-3
OBUFT_LVCMOS15	OBUFT_LVCMOS25_S_6	OBUFT_PCIX
OBUFT_LVCMOS15_S_2	OBUFT_LVCMOS25_S_8	OBUFT_GTL

Table 3-34: **Variations of the OBUFT Symbol *(Continued)***

OBUFT_LVCMOS15_S_4	OBUFT_LVCMOS25_S_12	OBUFT_GTLP
OBUFT_LVCMOS15_S_6	OBUFT_LVCMOS25_S_16	OBUFT_HSTL_I
OBUFT_LVCMOS15_S_8	OBUFT_LVCMOS25_S_24	OBUFT_HSTL_II
OBUFT_LVCMOS15_S_12	OBUFT_LVCMOS25_F_2	OBUFT_HSTL_III
OBUFT_LVCMOS15_S_16	OBUFT_LVCMOS25_F_4	OBUFT_HSTL_IV
OBUFT_LVCMOS15_F_2	OBUFT_LVCMOS25_F_6	OBUFT_SSTL3_I
OBUFT_LVCMOS15_F_4	OBUFT_LVCMOS25_F_8	OBUFT_SSTL3_II
OBUFT_LVCMOS15_F_6	OBUFT_LVCMOS25_F_12	OBUFT_SSTL2_I
OBUFT_LVCMOS15_F_8	OBUFT_LVCMOS25_F_16	OBUFT_SSTL2_II
OBUFT_LVCMOS15_F_12	OBUFT_LVCMOS25_F_24	OBUFT_SSTL18_I
OBUFT_LVCMOS15_F_16	OBUFT_LVCMOS33	OBUFT_SSTL18_II
OBUFT_LVCMOS18	OBUFT_LVCMOS33_S_2	OBUFT_AGP

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

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

The weak "keeper" circuit requires the input buffer within the IOB to sample the I/O signal. Thus, OBUFTs programmed for an I/O standard that requires a V_{REF} have automatic placement of a V_{REF} in the bank with an OBUFT configured with a weak "keeper" typically implement a bidirectional I/O. In this case, the IBUF (and the corresponding V_{REF}) are placed explicitly.

IOBUF

Use the IOBUF symbol for bidirectional signals that require both an input buffer and a 3-state output buffer with an active High 3-state pin. Figure 3-82 shows the generic input/output IOBUF buffer.

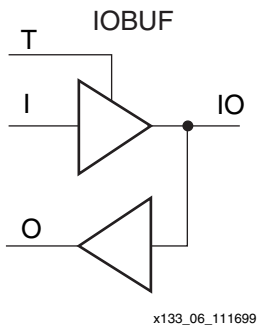


Figure 3-82: **Input/Output Buffer Symbol (IOBUF)**

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

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

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

IOBUF_<slew_rate>_<drive_strength>

<slew_rate> is either F (fast) or S (slow) and <drive_strength> is specified in milliamperes. For LVTTL, LVCMOS25 and LVCMOS33, the supported drive strengths are 2, 4, 6, 8, 12, 16, and 24. For LVCMOS15, and LVCMOS18, the supported drive strengths are 2, 4, 6, 8, 12, and 16. Table 3-35 details variations of the IOBUF symbol.

Table 3-35: **Variations of the IOBUF Symbol**

IOBUF	IOBUF_LVCMOS15_F_16	IOBUF_LVCMOS25_F_16
IOBUF_S_2	IOBUF_LVCMOS18	IOBUF_LVCMOS25_F_24
IOBUF_S_4	IOBUF_LVCMOS18_S_2	IOBUF_LVCMOS33
IOBUF_S_6	IOBUF_LVCMOS18_S_4	IOBUF_LVCMOS33_S_2
IOBUF_S_8	IOBUF_LVCMOS18_S_6	IOBUF_LVCMOS33_S_4
IOBUF_S_12	IOBUF_LVCMOS18_S_8	IOBUF_LVCMOS33_S_6
IOBUF_S_16	IOBUF_LVCMOS18_S_12	IOBUF_LVCMOS33_S_8
IOBUF_S_24	IOBUF_LVCMOS18_S_16	IOBUF_LVCMOS33_S_12
IOBUF_F_2	IOBUF_LVCMOS18_F_2	IOBUF_LVCMOS33_S_16
IOBUF_F_4	IOBUF_LVCMOS18_F_4	IOBUF_LVCMOS33_S_24
IOBUF_F_6	IOBUF_LVCMOS18_F_6	IOBUF_LVCMOS33_F_2
IOBUF_F_8	IOBUF_LVCMOS18_F_8	IOBUF_LVCMOS33_F_4
IOBUF_F_12	IOBUF_LVCMOS18F_12	IOBUF_LVCMOS33_F_6
IOBUF_F_16	IOBUF_LVCMOS18_F_16	IOBUF_LVCMOS33_F_8
IOBUF_F_24	IOBUF_LVCMOS25	IOBUF_LVCMOS33_F_12
IOBUF_LVCMOS15	IOBUF_LVCMOS25_S_2	IOBUF_LVCMOS33_F_16
IOBUF_LVCMOS15_S_2	IOBUF_LVCMOS25_S_4	IOBUF_LVCMOS33_F_24
IOBUF_LVCMOS15_S_4	IOBUF_LVCMOS25_S_6	IOBUF_PCI33_3
IOBUF_LVCMOS15_S_6	IOBUF_LVCMOS25_S_8	IOBUF_PCI66-3
IOBUF_LVCMOS15_S_8	IOBUF_LVCMOS25_S_12	IOBUF_PCIX
IOBUF_LVCMOS15_S_12	IOBUF_LVCMOS25_S_16	IOBUF_GTL
IOBUF_LVCMOS15_S_16	IOBUF_LVCMOS25_S_24	IOBUF_GTLP
IOBUF_LVCMOS15_F_2	IOBUF_LVCMOS25_F_2	IOBUF_HSTL_II
IOBUF_LVCMOS15_F_4	IOBUF_LVCMOS25_F_4	IOBUF_HSTL_IV
IOBUF_LVCMOS15_F_6	IOBUF_LVCMOS25_F_6	IOBUF_SSTL18_I
IOBUF_LVCMOS15_F_8	IOBUF_LVCMOS25_F_8	IOBUF_SSTL18_II
IOBUF_LVCMOS15_F_12	IOBUF_LVCMOS25_F_12	IOBUF_AGP

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

The voltage reference signal is "banked" within the Virtex-II device on a half-edge basis, such that for all packages there are eight independent V_{REF} banks internally. For a representation of the Virtex-II I/O banks, see Figure 3-79. Within each bank approximately one of every twelve I/O pins is automatically configured as a V_{REF} input. After placing a differential amplifier input signal within a given V_{REF} bank, the same external source must drive all I/O pins configured as a V_{REF} input.

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

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

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

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

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

SelectI/O-Ultra Properties

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

Input Delay Properties

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

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

IOB Flip-Flop/Latch Properties

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

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

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

The two registers for each path makes designing double-data-rate (DDR) logic much simpler. Each pair of the registers has separate clock inputs, which can be driven by either the

positive edge or the negative edge of the clock. Users can use both edges of the clocks to clock data in and out from the IOB. For details on DDR, see "Using Double-Data-Rate (DDR) I/O" on page 227.

Location Constraints

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

The LOC properties use the following form:

- LOC=A42;
- LOC=P37;

Output Slew Rate Property

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

- SLEW = SLOW
- SLEW = FAST

Output Drive Strength Property

The desired output drive strength can be additionally specified by choosing the appropriate library symbol. The Xilinx library also provides an alternative method for specifying this feature. For the LVTTL, and LVCMOS output buffers (OBUF, OBUFT, and IOBUF), the desired drive strength can be specified with the DRIVE = property. This property could have one of the following values:

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

Design Considerations

Reference Voltage (V_{REF}) Pins

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

The voltage reference signal is “banked” within the Virtex-II device on a half-edge basis such that for all packages there are eight independent V_{REF} banks internally. See Figure 3-79 for a representation of the Virtex-II I/O banks. Within each bank approximately one of every twelve I/O pins is automatically configured as a V_{REF} input. After placing a differential amplifier input signal within a given V_{REF} bank, the same external source must drive all I/O pins configured as a V_{REF} input.

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

Output Drive Source Voltage (V_{CCO}) Pins

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

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

It is recommended to power V_{CCO} even for non-V_{CCO} dependent input standards such as LVDS, LVPECL, SSTL, HSTL, and GTL. The input clamp diodes, which are always present regardless of I/O standard, become forward-biased if pin voltage exceeds V_{CCO} by the diode threshold voltage (~0.5V). Leaving V_{CCO} disconnected, or setting it to an excessively low voltage, can lead to a distorted input signal. To avoid this, V_{CCO} should be greater than the input high voltage (plus any overshoot), less the 0.5V diode voltage.

Furthermore, if a bank is unused, tie V_{CCO} to V_{CCAUX} or any available V_{CCO}. Do not leave it floating or connected to ground.

Transmission Line Effects

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

Termination Techniques

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

The following are output termination techniques:

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

The following are input termination techniques:

- None
- Parallel (Shunt)

These termination techniques can be applied in any combination. A generic example of each combination of termination methods appears in Figure 3-83.

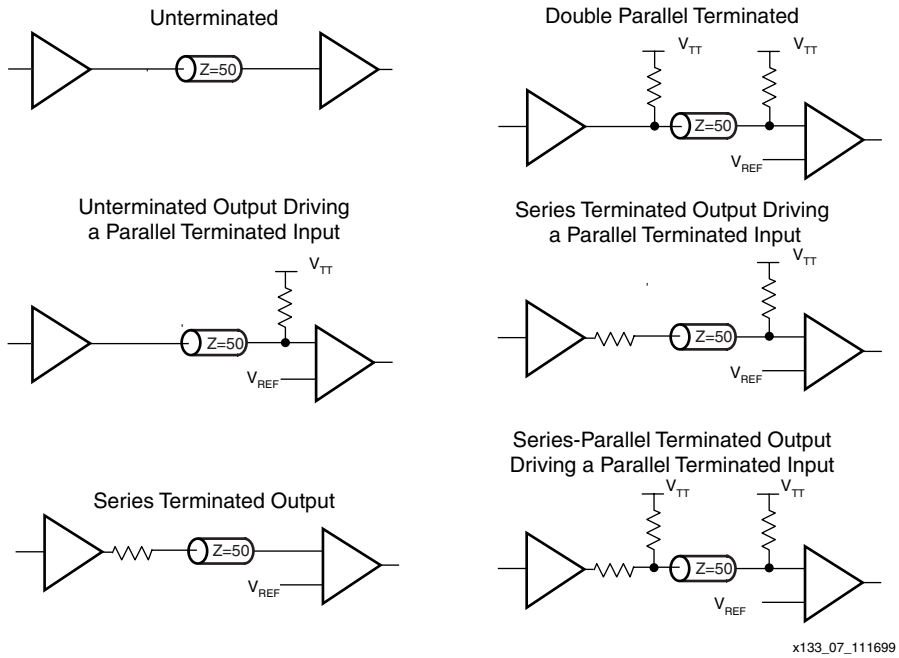


Figure 3-83: **Overview of Standard Input and Output Termination Methods**

Simultaneous Switching Output (SSO) Guidelines

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

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

Ground bounce affects stable low outputs and all inputs because they interpret the incoming signal by comparing it to the internal ground. If the ground bounce amplitude exceeds the actual instantaneous noise margin, then a non-changing input can be interpreted as changing. Table 3-36 provides the guidelines for the maximum number of simultaneously switching outputs allowed per output power/ground pair to avoid the effects of ground bounce. Refer to Table 3-37 for the number of effective output power/ground pairs for each Virtex-II device and package combination.

Note on LVDCI Drivers

All entries in the SSO table for LVDCI assume 50Ω reference resistors. To calculate the SSO limit for an LVDCI driver with different resistors, the following formula is used:

Higher Reference Resistor (RR) Value = Less Drive Strength = Higher SSO Limit

SSO Limit for New RR = (RR Value / 50Ω) × (SSO Limit for 50Ω)

Example:

If using the LVDCI_18 driver with 65Ω reference resistors, the LVDCI_18 SSO limit is taken from the table. The SSO limit for LVDCI_18 at 50Ω is 11 per power/ground pin pair. The SSO limit for LVDCI_18 at 65Ω is (65Ω / 50Ω) × 11 = 14 per power/ground pin pair.

Table 3-36: **Guidelines for Max Number of Simultaneously Switching Outputs per Power/Ground Pair**

Standard	Package			
	FG,BG,FF,BF	CS	XC2V40-FG	XC2V40-CS
LVTTL_S2	68	51	51	34
LVTTL_S4	41	31	31	21
LVTTL_S6	29	22	22	15
LVTTL_S8	22	17	17	11
LVTTL_S12	15	11	11	8
LVTTL_S16	11	8	8	6
LVTTL_S24	7	5	5	4
LVTTL_F2	40	30	30	20
LVTTL_F4	24	18	18	12
LVTTL_F6	17	13	13	9
LVTTL_F8	13	10	10	7
LVTTL_F12	10	8	8	5
LVTTL_F16	8	6	6	4
LVTTL_F24	5	4	4	3
LVDCI_15 50Ω impedance	10	8	8	5
LVDCI_DV2_15 25Ω impedance	5	4	4	3
LVCMOS15_S2	51	38	38	26
LVCMOS15_S4	31	23	23	16
LVCMOS15_S6	22	17	17	11
LVCMOS15_S8	17	13	13	9
LVCMOS15_S12	11	8	8	6
LVCMOS15_S16	8	6	6	4
LVCMOS15_F2	30	23	23	15
LVCMOS15_F4	18	14	14	9
LVCMOS15_F6	13	10	10	7
LVCMOS15_F8	10	8	8	5
LVCMOS15_F12	8	6	6	4
LVCMOS15_F16	6	5	5	3
LVDCI_18, 50Ω impedance	11	8	8	6
LVDCI_DV2_18, 25Ω impedance	6	4	4	3
LVCMOS18_S2	58	44	44	29
LVCMOS18_S4	35	26	26	18
LVCMOS18_S6	25	19	19	13
LVCMOS18_S8	19	14	14	10
LVCMOS18_S12	13	10	10	7
LVCMOS18_S16	10	8	8	5

Table 3-36: **Guidelines for Max Number of Simultaneously Switching Outputs per Power/Ground Pair *(Continued)***

Standard	Package			
	FG,BG,FF,BF	CS	XC2V40-FG	XC2V40-CS
LVCMOS18_F2	34	26	26	17
LVCMOS18_F4	20	15	15	10
LVCMOS18_F6	15	11	11	8
LVCMOS18_F8	11	8	8	6
LVCMOS18_F12	9	7	7	5
LVCMOS18_F16	7	5	5	4
LVDCI_25, 50 Ω impedance	13	10	10	7
LVDCI_DV2,_25 25 Ω impedance	7	5	5	3
LVCMOS25_S2	68	51	51	34
LVCMOS25_S4	41	31	31	21
LVCMOS25_S6	29	22	22	15
LVCMOS25_S8	22	17	17	11
LVCMOS25_S12	15	11	11	8
LVCMOS25_S16	11	8	8	6
LVCMOS25_S24	7	5	5	4
LVCMOS25_F2	40	30	30	20
LVCMOS25_F4	24	18	18	12
LVCMOS25_F6	17	13	13	9
LVCMOS25_F8	13	10	10	7
LVCMOS25_F12	10	8	8	5
LVCMOS25_F16	8	6	6	4
LVCMOS25_F24	5	4	4	2
LVDCI_33, 50Ω impedance	13	10	10	7
LVDCI_DV2_33, 25Ω impedance	7	5	5	3
LVCMOS33_S2	68	51	51	34
LVCMOS33_S4	41	31	31	21
LVCMOS33_S6	29	22	22	15
LVCMOS33_S8	22	17	17	11
LVCMOS33_S12	15	11	11	8
LVCMOS33_S16	11	8	8	6
LVCMOS33_S24	7	5	5	4
LVCMOS33_F2	40	30	30	20
LVCMOS33_F4	24	18	18	12
LVCMOS33_F6	17	13	13	9
LVCMOS33_F8	13	10	10	7
LVCMOS33_F12	10	8	8	5

Table 3-36: **Guidelines for Max Number of Simultaneously Switching Outputs per Power/Ground Pair *(Continued)***

Standard	Package			
	FG,BG,FF,BF	CS	XC2V40-FG	XC2V40-CS
LVCMOS33_F16	8	6	6	4
LVCMOS33_F24	5	4	4	2
PCI_33, PCI_66, PCIX	8	6	6	4
GTL	4	3	3	2
GTL_DCI	3	2	2	1
GTLP	4	3	3	2
GTLP_DCI	3	2	2	1
HSTL_I	20	15	15	10
HSTL_I_DCI	20	15	15	10
HSTL_II	10	8	8	5
HSTL_II_DCI	6	5	5	3
HSTL_III	8	6	6	4
HSTL_III_DCI	8	6	6	4
HSTL_IV	4	3	3	2
HSTL_IV_DCI	4	3	3	2
SSTL18_I	20	15	15	10
SSTL18_I_DCI	20	15	15	10
SSTL18_II	13	10	10	6
SSTL18_II_DCI	6	5	5	3
SSTL2_I	15	11	11	8
SSTL2_I_DCI	15	11	11	8
SSTL2_II	10	8	8	5
SSTL2_II_DCI	5	4	4	3
SSTL3_I	12	9	9	6
SSTL3_I_DCI	12	9	9	6
SSTL3_II	8	6	6	4
SSTL3_II_DCI	4	3	3	2
AGP	9	7	7	5
HSTL_I_18	16	12	12	8
HSTL_I_DCI_18	16	12	12	8
HSTL_II_18	8	6	6	4
HSTL_II_DCI_18	6	5	5	3
HSTL_III_18	8	6	6	4
HSTL_III_DCI_18	8	6	6	4
HSTL_IV_18	4	3	3	2
HSTL_IV_DCI_18	4	3	3	2

Table 3-37: **Virtex-II Equivalent Power/Ground Pairs per Bank**

Package	XC2V Device										
	40	80	250	500	1000	1500	2000	3000	4000	6000	8000
CS144/CSG144	1	1	1	-	-	-	-	-	-	-	-
FG256/FGG256	1	2	3	3	3	-	-	-	-	-	-
FG456/FGG456	-	-	3	4	5	-	-	-	-	-	-
FG676/FGG676	-	-	-	-	-	6	7	7	-	-	-
BG575/BGG575	-	-	-	-	5	6	6	-	-	-	-
BG728/BGG728	-	-	-	-	-	-	-	8	-	-	-
FF896	-	-	-	-	7	8	10	-	-	-	-
FF1152	-	-	-	-	-	-	-	11	13	13	13
FF1517	-	-	-	-	-	-	-	-	14	17	17
BF957	-	-	-	-	-	-	10	10	10	11	-

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

Refer to **XAPP689** for further details on managing ground bounce in large FPGAs.

Application Example

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

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

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

For example, to enter PCIX standard, use

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

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

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

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

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

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

Termination Example

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

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

GTL

A sample circuit illustrating a valid termination technique for GTL is shown in Figure 3-84.

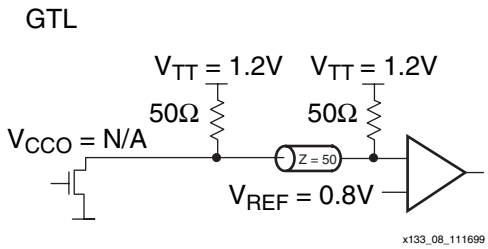


Figure 3-84: **GTL Terminated**

Table 3-38 lists DC voltage specifications.

Table 3-38: **GTL Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	-	N/A	-
$V_{REF} = N \times V_{TT}$ (1)	0.74	0.8	0.86
V_{TT}	1.14	1.2	1.26
$V_{IH} \geq V_{REF} + 0.05$	0.79	0.85	-
$V_{IL} \leq V_{REF} - 0.05$	-	0.75	0.81
V_{OH}	-	-	-
V_{OL}	-	0.2	0.4
I_{OH} at V_{OH} (mA)	-	-	-
I_{OL} at V_{OL} (mA) at 0.4 V	32	-	-
I_{OL} at V_{OL} (mA) at 0.2 V	-	-	40

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

GTL +

Figure 3-85 shows a sample circuit illustrating a valid termination technique for GTL+.

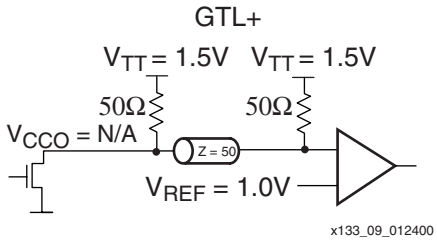


Figure 3-85: **GTL+ Terminated**

Table 3-39 lists DC voltage specifications.

Table 3-39: **GTL+ Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	-	-	-
$V_{REF} = N \times V_{TT}$ (1)	0.88	1.0	1.12
V_{TT}	1.35	1.5	1.65
$V_{IH} \geq V_{REF} + 0.1$	0.98	1.1	-
$V_{IL} \leq V_{REF} - 0.1$	-	0.9	1.02
V_{OH}	-	-	-
V_{OL}	0.3	0.45	0.6
I_{OH} at V_{OH} (mA)	-	-	-
I_{OL}at V_{OL} (mA) at 0.6V	36	-	-
I_{OL}at V_{OL} (mA) at 0.3V	-	-	48

Notes:

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

HSTL Class I

Figure 3-86 shows a sample circuit illustrating a valid termination technique for HSTL_I.

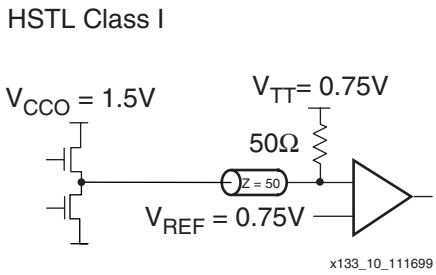


Figure 3-86: **Terminated HSTL Class I**

Table 3-40 lists DC voltage specifications.

Table 3-40: **HSTL Class I Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.40	1.50	1.60
V_{REF}	0.68	0.75	0.90
V_{TT}	-	$V_{CCO} \times 0.5$	-
V_{IH}	V_{REF} + 0.1	-	-
V_{IL}	-	-	V_{REF} – 0.1
V_{OH}	V_{CCO} – 0.4	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–8	-	-
I_{OL} at V_{OL} (mA)	8	-	-

HSTL Class II

Figure 3-87 shows a sample circuit illustrating a valid termination technique for HSTL_II.

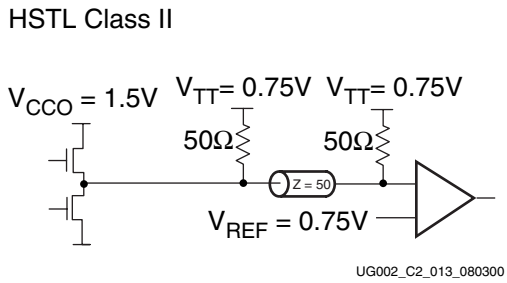


Figure 3-87: **Terminated HSTL Class II**

Table 3-41 lists DC voltage specifications.

Table 3-41: **HSTL Class II Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.40	1.50	1.60
V_{REF} (1)	-	0.75	-
V_{TT}	-	$V_{CCO} \times 0.5$	-
V_{IH}	$V_{REF} + 0.1$	-	-
V_{IL}	-	-	$V_{REF} - 0.1$
V_{OH}	$V_{CCO} - 0.4$	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–16	-	-
I_{OL} at V_{OL} (mA)	16	-	-

Notes:

1. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

HSTL Class III

Figure 3-88 shows a sample circuit illustrating a valid termination technique for HSTL_III.

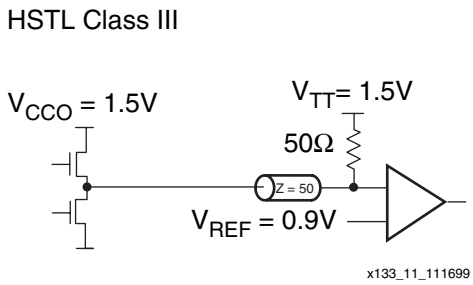


Figure 3-88: **Terminated HSTL Class III**

Table 3-42 lists DC voltage specifications.

Table 3-42: **HSTL Class III Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.40	1.50	1.60
V_{REF} (1)	-	0.90	-
V_{TT}	-	V_{CCO}	-
V_{IH}	V_{REF} + 0.1	-	-
V_{IL}	-	-	V_{REF} – 0.1
V_{OH}	V_{CCO} – 0.4	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–8	-	-
I_{OL}at V_{OL} (mA)	24	-	-

Notes:

1. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

HSTL Class IV

Figure 3-89 shows a sample circuit illustrating a valid termination technique for HSTL_IV.

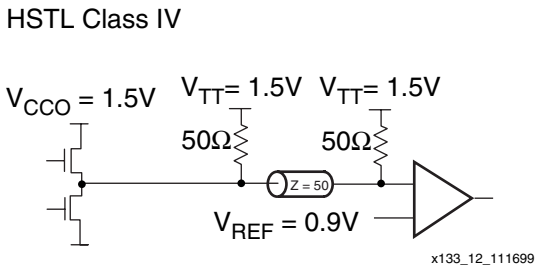


Figure 3-89: **Terminated HSTL Class IV**

Table 3-43 lists DC voltage specifications.

Table 3-43: **HSTL Class IV Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.40	1.50	1.60
V_{REF}	-	0.90	-
V_{TT}	-	V_{CCO}	-
V_{IH}	V_{REF} + 0.1	-	-
V_{IL}	-	-	V_{REF} – 0.1
V_{OH}	V_{CCO} – 0.4	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–8	-	-
I_{OL}at V_{OL} (mA)	48	-	-

Notes:

1. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user.

HSTL Class I (1.8V)

Figure 3-90 shows a sample circuit illustrating a valid termination technique for HSTL_I.

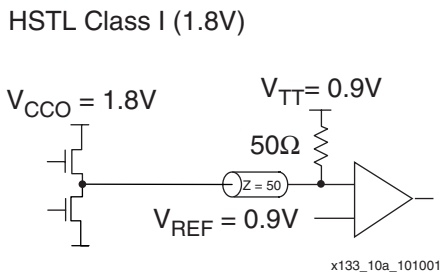


Figure 3-90: **Terminated HSTL Class I (1.8V)**

Table 3-44 lists DC voltage specifications.

Table 3-44: **HSTL Class I (1.8V) Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.7	1.8	1.9
V_{REF}	0.8	0.9	1.1
V_{TT}	-	$V_{CCO} \times 0.5$	-
V_{IH}	$V_{REF} + 0.1$	-	-
V_{IL}	-	-	$V_{REF} - 0.1$
V_{OH}	$V_{CCO} - 0.4$	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–8	-	-
I_{OL} at V_{OL} (mA)	8	-	-

HSTL Class II (1.8V)

Figure 3-91 shows a sample circuit illustrating a valid termination technique for HSTL_II.

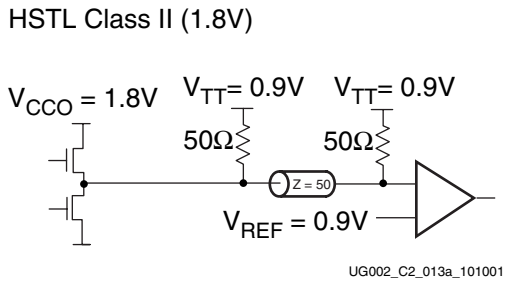


Figure 3-91: **Terminated HSTL Class II (1.8V)**

Table 3-45 lists DC voltage specifications.

Table 3-45: **HSTL Class II (1.8V) Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.7	1.8	1.9
V_{REF} (1)	-	0.9	-
V_{TT}	-	$V_{CCO} \times 0.5$	-
V_{IH}	$V_{REF} + 0.1$	-	-
V_{IL}	-	-	$V_{REF} - 0.1$
V_{OH}	$V_{CCO} - 0.4$	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–16	-	-
I_{OL} at V_{OL} (mA)	16	-	-

Notes:

1. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

HSTL Class III (1.8V)

Figure 3-92 shows a sample circuit illustrating a valid termination technique for HSTL_III.

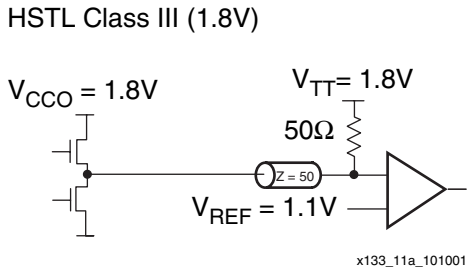


Figure 3-92: **Terminated HSTL Class III (1.8V)**

Table 3-46 lists DC voltage specifications.

Table 3-46: **HSTL Class III (1.8V) Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.7	1.8	1.9
V_{REF} (1)	-	1.1	-
V_{TT}	-	V_{CCO}	-
V_{IH}	V_{REF} + 0.1	-	-
V_{IL}	-	-	V_{REF} – 0.1
V_{OH}	V_{CCO} – 0.4	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–8	-	-
I_{OL} at V_{OL} (mA)	24	-	-

Notes:

1. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

HSTL Class IV (1.8V)

Figure 3-93 shows a sample circuit illustrating a valid termination technique for HSTL_IV.

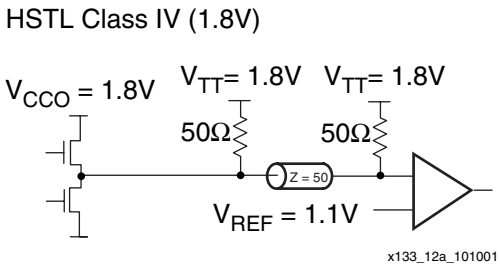


Figure 3-93: **Terminated HSTL Class IV (1.8V)**

Table 3-47 lists DC voltage specifications.

Table 3-47: **HSTL Class IV (1.8V) Voltage Specification**

Parameter	MIN	TYP	MAX
V_{CCO}	1.7	1.8	1.9
V_{REF}	-	1.1	-
V_{TT}	-	V_{CCO}	-
V_{IH}	V_{REF} + 0.1	-	-
V_{IL}	-	-	V_{REF} – 0.1
V_{OH}	V_{CCO} – 0.4	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	–8	-	-
I_{OL}at V_{OL} (mA)	48	-	-

Notes:

1. Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user.

SSTL3_I

Figure 3-94 shows a sample circuit illustrating a valid termination technique for SSTL3_I.

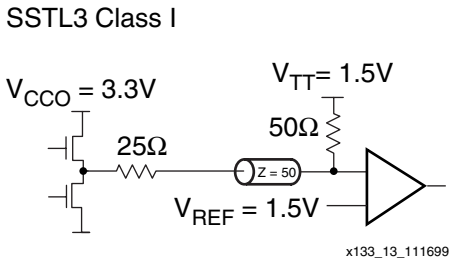


Figure 3-94: **Terminated SSTL3_I**

Table 3-48 lists DC voltage specifications.

Table 3-48: **SSTL3_I Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
$V_{REF} = 0.45 \times V_{CCO}$	1.3	1.5	1.7
$V_{TT} = V_{REF}$	1.3	1.5	1.7
$V_{IH} \geq V_{REF} + 0.2$	1.5	1.7	3.9 (1)
$V_{IL} \leq V_{REF} - 0.2$	–0.3 (2)	1.3	1.5
$V_{OH} \geq V_{REF} + 0.6$ (3)	1.9	2.1	-
$V_{OL} \leq V_{REF} - 0.6$ (3)	-	0.9	1.1
I_{OH} at V_{OH} (mA)	–8	-	-
I_{OL}at V_{OL} (mA)	8	-	-

Notes:
1. V_{IH} maximum is V_{CCO} + 0.3
2. V_{IL} minimum does not conform to the formula
3. V_{OH} and V_{OL} values are different for SSTL3_I_DCI, which uses a controlled impedance driver.

SSTL3_II

Figure 3-95 shows a sample circuit illustrating a valid termination technique for SSTL3_II.

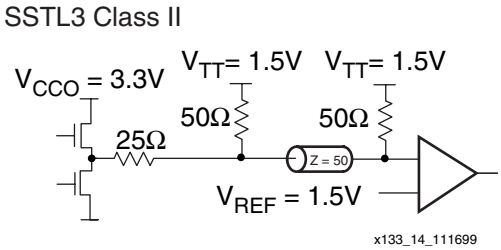


Figure 3-95: **Terminated SSTL3_II**

Table 3-49 lists DC voltage specifications.

Table 3-49: **SSTL3_II Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
$V_{REF} = 0.45 \times V_{CCO}$	1.3	1.5	1.7
$V_{TT} = V_{REF}$	1.3	1.5	1.7
$V_{IH} \geq V_{REF} + 0.2$	1.5	1.7	3.9 (1)
$V_{IL} \leq V_{REF} - 0.2$	–0.3 (2)	1.3	1.5
$V_{OH} \geq V_{REF} + 0.8$ (3)	2.1	2.3	-
$V_{OL} \leq V_{REF} - 0.8$ (3)	-	0.7	0.9
I_{OH} at V_{OH} (mA)	–16	-	-
I_{OL}at V_{OL} (mA)	16	-	-

Notes:

1. V_{IH} maximum is V_{CCO} + 0.3.
2. V_{IL} minimum does not conform to the formula.
3. V_{OH} and V_{OL} values are different for SSTL3_II_DCI, which uses a controlled impedance driver.

SSTL2_I

Figure 3-96 shows a sample circuit illustrating a valid termination technique for SSTL2_I.

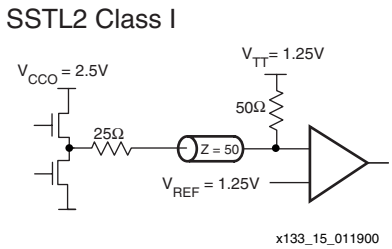


Figure 3-96: **Terminated SSTL2_I**

Table 3-50 lists DC voltage specifications.

Table 3-50: **SSTL2_I Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
$V_{REF} = 0.5 \times V_{CCO}$	1.15	1.25	1.35
$V_{TT} = V_{REF} + N$ (1)	1.11	1.25	1.39
$V_{IH} \geq V_{REF} + 0.18$	1.33	1.43	3.0 (2)
$V_{IL} \leq V_{REF} - 0.18$	–0.3 (3)	1.07	1.17
$V_{OH} \geq V_{REF} + 0.61$ (4)	1.76	1.82	1.96
$V_{OL} \leq V_{REF} - 0.61$ (4)	0.54	0.64	0.74
I_{OH} at V_{OH} (mA)	–7.6	-	-
I_{OL}at V_{OL} (mA)	7.6	-	-

Notes:

1. N must be greater than or equal to -0.04 and less than or equal to 0.04.
2. V_{IH} maximum is V_{CCO} + 0.3.
3. V_{IL} minimum does not conform to the formula.
4. V_{OH} and V_{OL} values are different for SSTL2_I_DCI, which uses a controlled impedance driver.

SSTL2_II

Figure 3-97 shows a sample circuit illustrating a valid termination technique for SSTL2_II.

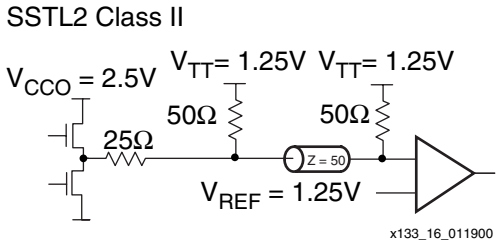


Figure 3-97: **Terminated SSTL2_II**

Table 3-51 lists DC voltage specifications.

Table 3-51: **SSTL2_II Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
$V_{REF} = 0.5 \times V_{CCO}$	1.15	1.25	1.35
$V_{TT} = V_{REF} + N$ (1)	1.11	1.25	1.39
$V_{IH} \geq V_{REF} + 0.18$	1.33	1.43	3.0 (2)
$V_{IL} \leq V_{REF} - 0.18$	–0.3 (3)	1.07	1.17
$V_{OH} \geq V_{REF} + 0.8$ (4)	1.95	2.05	-
$V_{OL} \leq V_{REF} - 0.8$ (4)	-	0.45	0.55
I_{OH} at V_{OH} (mA)	–15.2	-	-
I_{OL}at V_{OL} (mA)	15.2	-	-

Notes:

1. N must be greater than or equal to –0.04 and less than or equal to 0.04.
2. V_{IH} maximum is V_{CCO} + 0.3.
3. V_{IL} minimum does not conform to the formula.
4. V_{OH} and V_{OL} values are different for SSTL2_II_DCI, which uses a controlled impedance driver.

PCI33_3, PCI66_3, and PCIX

Table 3-52 lists DC voltage specifications.

Table 3-52: **PCI33_3, PCI66_3, and PCIX Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.5
V_{REF}	-	-	-
V_{TT}	-	-	-
$V_{IH} = 0.5 \times V_{CCO}$	1.5	1.65	$V_{CCO} + 0.5$
$V_{IL} = 0.3 \times V_{CCO}$	– 0.5	0.99	1.08
$V_{OH} = 0.9 \times V_{CCO}$	2.7	-	-
$V_{OL} = 0.1 \times V_{CCO}$	-	-	0.36
I_{OH} at V_{OH} (mA)	Note 1	-	-
I_{OL} at V_{OL} (mA)	Note 1	-	-

Notes:
1. Tested according to the relevant specification.

LVTTL

Table 3-53 lists DC voltage specifications.

Table 3-53: **LVTTL Voltage Specifications**

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

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

LVCMOS15

Table 3-54 lists DC voltage specifications.

Table 3-54: **LVCMOS15 Voltage Specifications**

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

LVCMOS18

Table 3-55 lists DC voltage specifications.

Table 3-55: **LVCMOS18 Voltage Specifications**

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

LVCMOS25

Table 3-56 lists DC voltage specifications.

Table 3-56: **LVCMOS25 Voltage Specifications**

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

LVCMOS33

Table 3-57 lists DC voltage specifications.

Table 3-57: **LVCMOS33 Voltage Specifications**

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

AGP-2X

Table 3-58 lists DC voltage specifications.

Table 3-58: **AGP-2X Voltage Specifications**

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

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

5V Tolerance in Virtex-II Devices

Virtex-II I/Os are not 5V tolerant without adding an external current-limiting resistor. Each Virtex-II I/O has a pair of clamp diodes that connects to V_{CCO} and GND, as shown in the Virtex-II *Data Sheet*.

However, for LVCMOS/LVTTL I/O standards, a workaround for this problem is to use a resistor in series to limit the current into the clamp diode. (This works only for a higher voltage that is driving Virtex-II input. It does not work for a Virtex-II output or bidirectional signal, because the resulting V_{OH} is lower than the specification of the other device.) Given that the forward-bias voltage of the clamp diode is 0.5V, and the limit that any I/O pin can be overdriven above or below the limits of GND and V_{CCO} is 10 mA, the following is a calculation example:

Assume that driving 5V from a device to the LVCMOS33 input in a Virtex-II device is wanted:

- V_{CCO} minimum for LVCMOS33 is 3.0V.
- V_{OH} maximum for the LVCMOS5 is 5.25V.

This means that the diode is forward-biased at a voltage of 3.0V + 0.5V = 3.5V or higher.

- The maximum voltage difference is 5.25V - 3.5V = 1.75V.
- To limit the current to 10 mA, a resistor (R) of 1.75/10 mA = 175 ohms is needed.

Notes:
1. The minimum V_{CCO} value is specified in this section.
2. The minimum V_{CCO} was used for the worst-case calculation.
3. A stable, clean, and properly bypassed V_{CCO} must be maintained at all times.
4. Placing the resistor closer to the driver provides better signal integrity.
5. IBIS simulation can be performed to verify the result.
6. The clamp diodes are always there (programmed, unprogrammed, during configuration), and there is never a need to add external clamp diodes in the event that the Virtex-II device is unprogrammed and has 5V signals driving the pins.

Using Digitally Controlled Impedance (DCI)

Introduction

As FPGAs get bigger and system clock speeds get faster, PCB board design and manufacturing has become more difficult. With ever faster edge rates, maintaining signal integrity becomes a critical issue. Designers must make sure that most PC board traces are terminated properly to avoid reflections or ringing.

To terminate a trace, resistors are traditionally added to make the output and/or input match the impedance of the receiver or driver to the impedance of the trace. However, due to the increase in the device I/O counts, adding resistors close to the device pins increases the board area and component count and might even be physically impossible. To address these issues and to achieve better signal integrity, Xilinx developed a new I/O technology for the Virtex-II device family, Digitally Controlled Impedance (DCI).

DCI adjusts the output impedance or input termination to accurately match the characteristic impedance of the transmission line. DCI actively adjusts the impedance of the I/O to equal an external reference resistance. This compensates for changes in I/O impedance due to process variation. It also continuously adjusts the impedance of the I/O to compensate for variations of temperature and supply voltage fluctuations.

In the case of controlled impedance drivers, DCI controls the driver impedance to match two reference resistors, or optionally, to match half the value of these reference resistors. DCI eliminates the need for external series termination resistors.

DCI provides parallel or series termination for transmitters or receivers. This eliminates the need for termination resistors on the board, reduces board routing difficulties and component count, and improves signal integrity by eliminating stub reflection. Stub reflection occurs when termination resistors are located too far from the end of the transmission line. With DCI, the termination resistors are as close as possible to the output driver or the input buffer, thus, eliminating stub reflections completely.

Xilinx DCI

DCI uses two multi-purpose reference pins in each bank to control the impedance of the driver or the parallel termination value for all of the I/Os of that bank. The N reference pin (VRN) must be pulled up to V_{CCO} by a reference resistor, and the P reference pin (VRP) must be pulled down to ground by another reference resistor. The value of each reference resistor should be equal to the characteristic impedance of the PC board traces, or should be twice that value (configuration option).

When a DCI I/O standard is used on a particular bank, the two multi-purpose reference pins cannot be used as regular I/Os. However, if DCI I/O standards are not used in the bank, these pins are available as regular I/O pins. Check the Virtex-II pinout for detailed pin descriptions.

DCI adjusts the impedance of the I/O by selectively turning transistors in the I/Os on or off. The impedance is adjusted to match the external reference resistors. The impedance adjustment process has two phases. The first phase, which compensates for process variations, is done during the device startup sequence. The second phase, which maintains the impedance in response to temperature and supply voltage changes, begins immediately after the first phase and continues indefinitely, even while the part is operating. By default, the DONE pin does not go High until the first phase of the impedance adjustment process has completed. If users do not want to have the second phase enabled, they should use the FreezeDCI option in BitGen. If FreezeDCI is used, phase one is the only impedance adjustment.

For controlled impedance output drivers, the impedance can be adjusted either to match the reference resistors or half the resistance of the reference resistors. For on-chip termination, the termination is always adjusted to match the reference resistors.

DCI can configure output drivers to be the following types:

1. Controlled Impedance Driver (Source Termination)
2. Controlled Impedance Driver with Half Impedance (Source Termination)

It can also configure inputs to have he following types of on-chip terminations:

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

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

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

Alternatively, bidirectional point-to-point lines can use controlled-impedance drivers (with 3-state buffers) on both ends.

Controlled Impedance Driver (Source Termination)

Some I/O standards, such as LVTTL, LVCMOS, etc., must have a drive impedance that matches the characteristic impedance of the driven line. DCI can provide a controlled impedance output drivers that eliminate reflections without an external source termination. The impedance is set by the external reference resistors, whose resistance should be equal to the trace impedance. Figure 3-98 illustrates a controlled impedance driver inside Virtex-II device. The DCI I/O standards that support Controlled Impedance Driver are: LVDCI_15, LVDCI_18, LVDCI_25, and LVDCI_33.

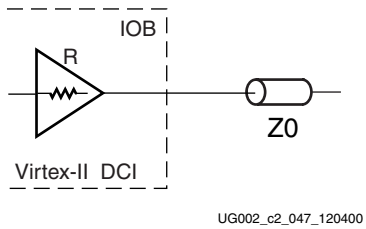


Figure 3-98: **Controlled Impedance Driver**

Controlled Impedance Driver With Half Impedance (Source Termination)

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

Figure 3-99 illustrates a controlled driver with half impedance inside a Virtex-II device.

Note that to match the drive impedance to Z0 when using a driver with half impedance, reference resistor R must be 2(Z0).

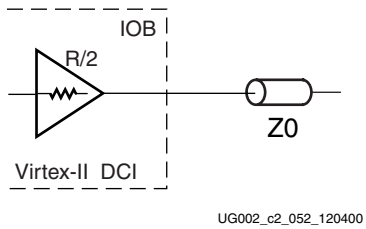


Figure 3-99: **Controlled Impedance Driver With Half Impedance**

Termination to V_{CCO} (Single Termination)

Some I/O standards, such as HSTL Class III, IV, etc., require an input termination to V_{CCO}. See Figure 3-100.

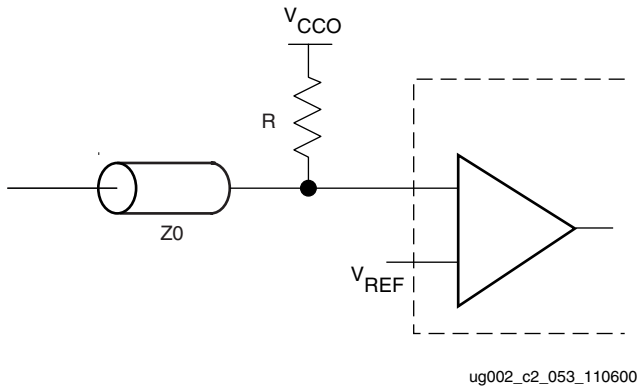


Figure 3-100: **Single Termination Without DCI**

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

Figure 3-101 illustrates single termination inside a Virtex-II device.

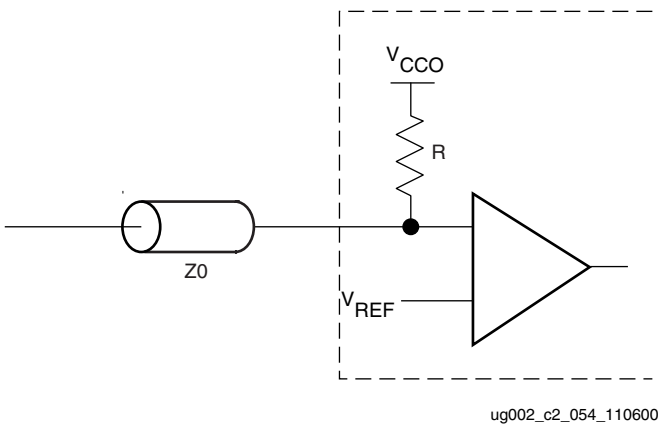


Figure 3-101: **Single Termination Using DCI**

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

Some I/O standards, such as HSTL Class I, II, SSTL3_I, etc., require an input termination voltage of $V_{CCO}/2$. See Figure 3-102.

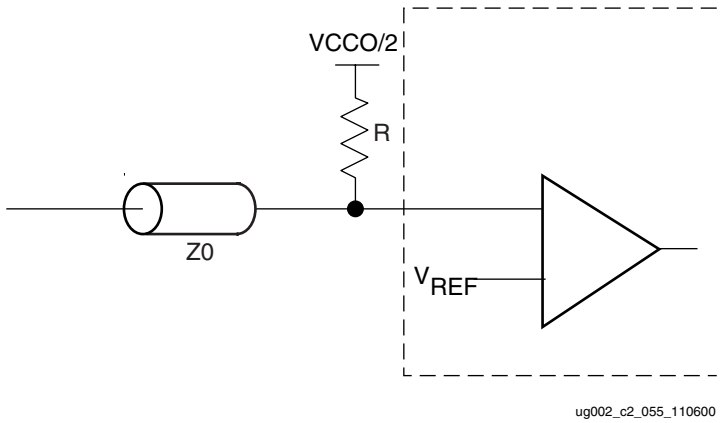


Figure 3-102: **Split Termination Without DCI**

This is equivalent to having a split termination composed of two resistors. One terminates to V_{CCO}, the other to ground. The resistor values are 2R. DCI provides termination to $V_{CCO}/2$ using split termination. The termination resistance is set by the external reference resistors, i.e., the resistors to V_{CC} and ground are each twice the reference resistor value. If users are planning to use HSTL or SSTL standards, the reference resistors should be 50-ohms. The DCI I/O standards that support split termination are: HSTL_I_DCI, HSTL_II_DCI, SSTL2_I_DCI, SSTL2_II_DCI, SSTL3_I_DCI, and SSTL3_II_DCI.

Figure 3-103 illustrates split termination inside a Virtex-II device.

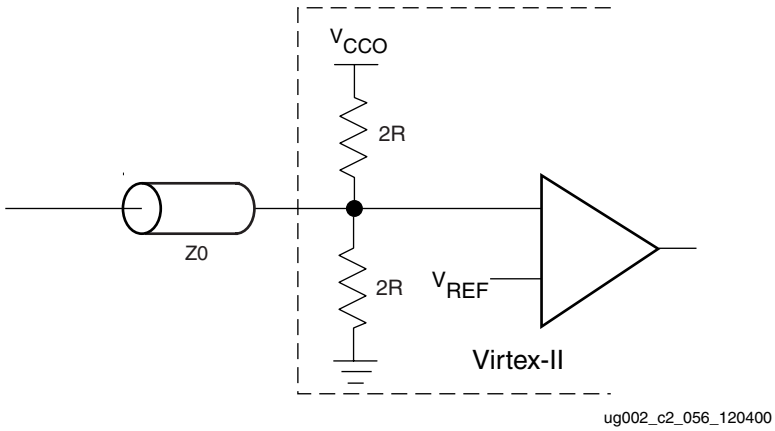


Figure 3-103: **Split Termination Using DCI**

Driver With Single Termination

Some I/O standards, such as HSTL Class IV, require an output termination to V_{CCO}. Figure 3-104 illustrates the output termination to V_{CCO}.

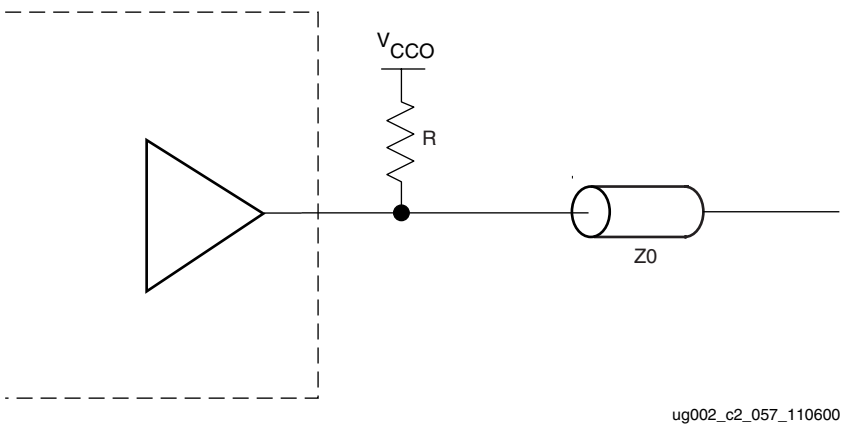


Figure 3-104: **Driver With Single Termination Without DCI**

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

Figure 3-105 illustrates a driver with single termination inside a Virtex-II device

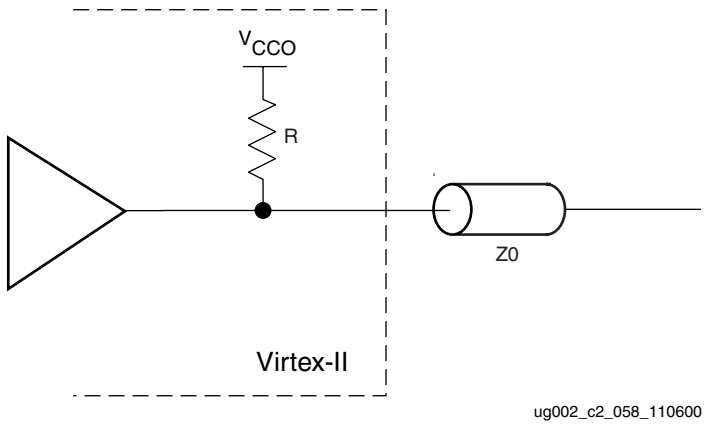


Figure 3-105: **Driver With Single Termination Using DCI**

Driver With Split Termination

Some I/O standards, such as HSTL Class II, require an output termination to $V_{CCO}/2$. See Figure 3-106.

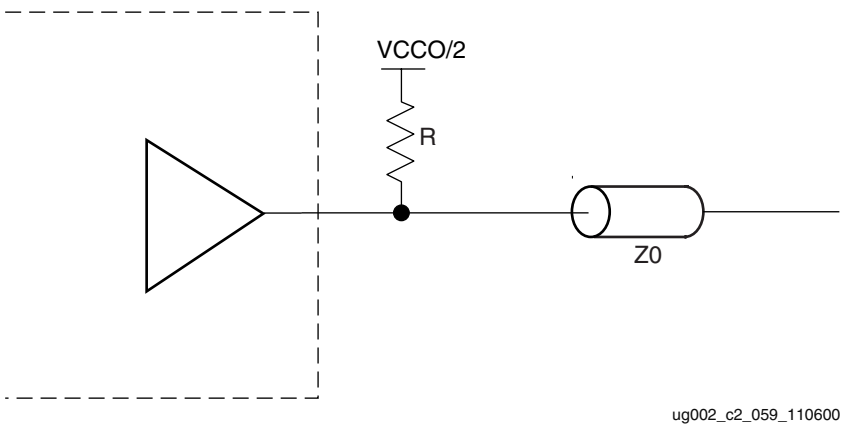


Figure 3-106: **Driver With Split Terminating**

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

Figure 3-107 illustrates a driver with split termination inside a Virtex-II device.

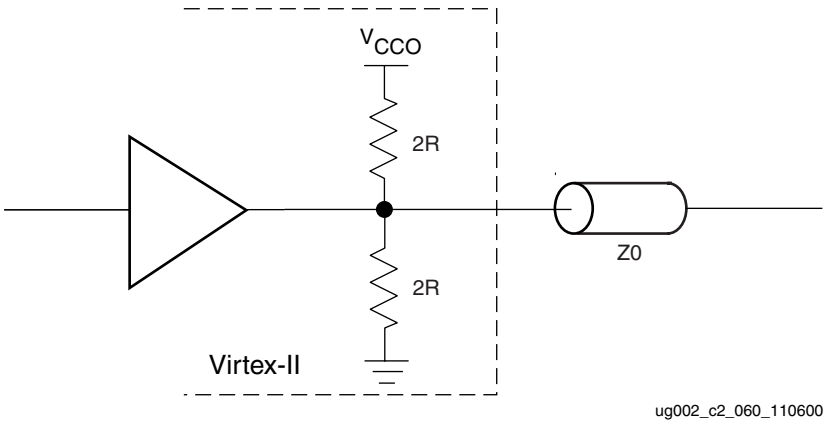


Figure 3-107: **Driver With Split Termination Using DCI**

Software Support

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

DCI I/O Buffer Library Components

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

- IBUFG_LVDS_25_DCI
- IBUFG_LVDSEXT_25_DCI
- IBUF_LVDS_25_DCI
- IBUF_LVDSEXT_25_DCI
- IBUFG_GTLP_DCI
- IBUFG_GTL_DCI
- IBUFG_HSTL_I_DCI
- IBUFG_HSTL_II_DCI
- IBUFG_HSTL_III_DCI
- IBUFG_HSTL_IV_DCI
- IBUFG_LVDCI_15
- IBUFG_LVDCI_18
- IBUFG_LVDCI_25
- IBUFG_LVDCI_33
- IBUFG_LVDCI_DV2_15
- IBUFG_LVDCI_DV2_18
- IBUFG_LVDCI_DV2_25
- IBUFG_LVDCI_DV2_33
- IBUFG_SSTL2_I_DCI
- IBUFG_SSTL2_II_DCI
- IBUFG_SSTL3_I_DCI
- IBUFG_SSTL3_II_DCI
- IBUF_GTLP_DCI
- IBUF_GTL_DCI
- IBUF_HSTL_I_DCI
- IBUF_HSTL_II_DCI
- IBUF_HSTL_III_DCI
- IBUF_HSTL_IV_DCI
- IBUF_LVDCI_15
- IBUF_LVDCI_18
- IBUF_LVDCI_25
- IBUF_LVDCI_33
- IBUF_LVDCI_DV2_15
- IBUF_LVDCI_DV2_18
- IBUF_LVDCI_DV2_25
- IBUF_LVDCI_DV2_33

- IBUF_SSTL2_I_DCI
- IBUF_SSTL2_II_DCI
- IBUF_SSTL3_I_DCI
- IBUF_SSTL3_II_DCI

The following are DCI output buffer library components:

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

The following are DCI 3 state output buffer library components:

- OBUFT_GTLP_DCI
- OBUFT_GTL_DCI
- OBUFT_HSTL_I_DCI
- OBUFT_HSTL_II_DCI
- OBUFT_HSTL_III_DCI
- OBUFT_HSTL_IV_DCI
- OBUFT_LVDCI_15
- OBUFT_LVDCI_18
- OBUFT_LVDCI_25
- OBUFT_LVDCI_33
- OBUFT_LVDCI_DV2_15
- OBUFT_LVDCI_DV2_18
- OBUFT_LVDCI_DV2_25
- OBUFT_LVDCI_DV2_33

- OBUFT_SSTL2_I_DCI
- OBUFT_SSTL2_II_DCI
- OBUFT_SSTL3_I_DCI
- OBUFT_SSTL3_II_DCI

The following are DCI I/O buffer library components:

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

How to Use DCI in the Software

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

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

IOSTANDARD Attribute

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

NET <net name> IOSTANDARD = LVDCI_25;

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

The following are valid DCI attributes for output drivers:

- LVDCI_15
- LVDCI_18
- LVDCI_25
- LVDCI_33
- LVDCI_DV2_15
- LVDCI_DV2_18
- LVDCI_DV2_25
- LVDCI_DV2_33

The following are valid DCI attributes for terminations:

- GTL_DCI
- GTLP_DCI
- HSTL_I_DCI
- HSTL_II_DCI
- HSTL_III_DCI
- HSTL_IV_DCI
- SSTL2_I_DCI
- SSTL2_II_DCI
- SSTL3_I_DCI
- SSTL3_II_DCI

VHDL Example

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

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

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

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

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

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

architecture dci_arch of dci_test is

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

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

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

```
begin

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

process (controls, A, B, DOUT_INT)

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

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

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

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

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

end dci_arch;
```

DCI in Virtex-II Hardware

DCI only works with certain single-ended and differential I/O standards. DCI supports the following Virtex-II standards:

LVDCI, LVDCI_DV2, GTL_DCI, GTLP_DCI, HSTL_I_DCI, HSTL_II_DCI, HSTL_III_DCI, HSTL_IV_DCI, SSTL2_I_DCI, SSTL2_II_DCI, SSTL3_I_DCI, and SSTL3_II_DCI, LVDS_25, LVDSEXT_25, LVDS_25_DCI, and LVDSEXT_25_DCI.

To correctly use DCI in a Virtex-II device, users must follow the following rules:

1. V_{CCO} pins must be connected to the appropriate V_{CCO} voltage based on the IOSTANDARDs in that bank.
2. Correct DCI I/O buffers must be used in the software either by using IOSTANDARD attributes or instantiations in the HDL code.
3. External reference resistors must be connected to multi-purpose pins (VRN and VRP) in the bank cannot be used as regular I/Os. Refer to the Virtex-II pinouts for the specific pin locations. Pin VRN must be pulled up to V_{CCO} by its reference resistor. Pin VRP must be pulled down to ground by its reference resistor.
4. The value of the external reference resistors should be selected to give the desired output impedance. If using GTL_DCI, HSTL_DCI, or SSTL_DCI I/O standards, then they should be 50Ω.
5. The values of the reference resistors must be within the supported range. Availability of this range is planned for the next release of the **Virtex-II Data Sheet (DS031)**. (~30 to 100 Ω)
6. Follow the DCI I/O banking rules:
   a. V_{REF} must be compatible for all of the inputs in the same bank.
   b. V_{CCO} must be compatible for all of the inputs and outputs in the same bank.
   c. No more than one DCI I/O standard using Single Termination type is allowed per bank.
   d. No more than one DCI I/O standard using Split Termination type is allowed per bank.
   e. Single Termination and Split Termination, Controlled Impedance Driver, and Controlled Impedance Driver with Half Impedance can co-exist in the same bank.
7. Avoid the simultaneous occurrence of these three conditions:
   a. The D0-D7, INIT, or DOUT pins are used as I/O after configuration
   b. One (or more) of those pins is a DCI input or output
   c. The "FreezeDCI" option is set to "Yes"

   Otherwise, these I/Os will not function properly. In the case of an output, the I/O will be put into a 3-state condition, or will not have the correct termination. In the case of an input, the I/O will not have the correct termination.

   This does not affect the RDWR_B and CS_B signals, which are also dual-purpose configuration pins.
8. The "FreezeDCI" bitgen option cannot be set to "Yes" when bitstream is encrypted because they are not compatible.
9. For CS144 package (available in XC2V40 and XC2V80) only: The "FreezeDCI" BitGen option cannot be used when ALT_VRP/ALT_VRN pins are used.

The behavior of DCI 3-state outputs is as follows:

If a LVDCI or LVDCI_DV2 driver is in 3-state, the driver is 3-stated. If a Driver with Single or Split Termination is in 3-state, the driver is 3-stated but the termination resistor remains.

The following section lists any special care actions that must be taken for each DCI I/O standard.

LVDCI_15, LVDCI_18, LVDCI_25, LVDCI_33

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

LVDCI_DV2_15, LVDCI_DV2_18, LVDCI_DV2_25, LVDCI_DV_33

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

GTL_DCI

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

GTLP_DCI

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

HSTL_ I_DCI, HSTL_ III_DCI

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

HSTL_ II_DCI, HSTL_ IV_DCI

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

SSTL2_ I_DCI, SSTL3_I_DCI

SSTL2_I_DCI and SSTL3_I_DCI provide split termination to $V_{CCO}/2$ for inputs. Then I/O standards are SSTL compatible. SSTL2_I_DCI and SSTL3_I_DCI drivers have different V_{OH} and V_{OL} values than non-DCI SSTL2_I and SSTL3_I drivers.

SSTL2_II_DCI, SSTL3_II_DCI

SSTL2_II_DCI and SSTL3_II_DCI provide split termination to $V_{CCO}/2$ for inputs. Then I/O standards are SSTL compatible. SSTL2_II_DCI and SSTL3_II_DCI drivers have different V_{OH} and V_{OL} values than non-DCI SSTL2_II and SSTL3_II drivers.

Answer Record 13012, available at **support.xilinx.com**, provides additional information on using DCI.

Figure 3-108 provides examples illustrating the use of the HSTL_I_DCI, HSTL_II_DCI, HSTL_III_DCI, and HSTL_IV_DCI I/O standards.

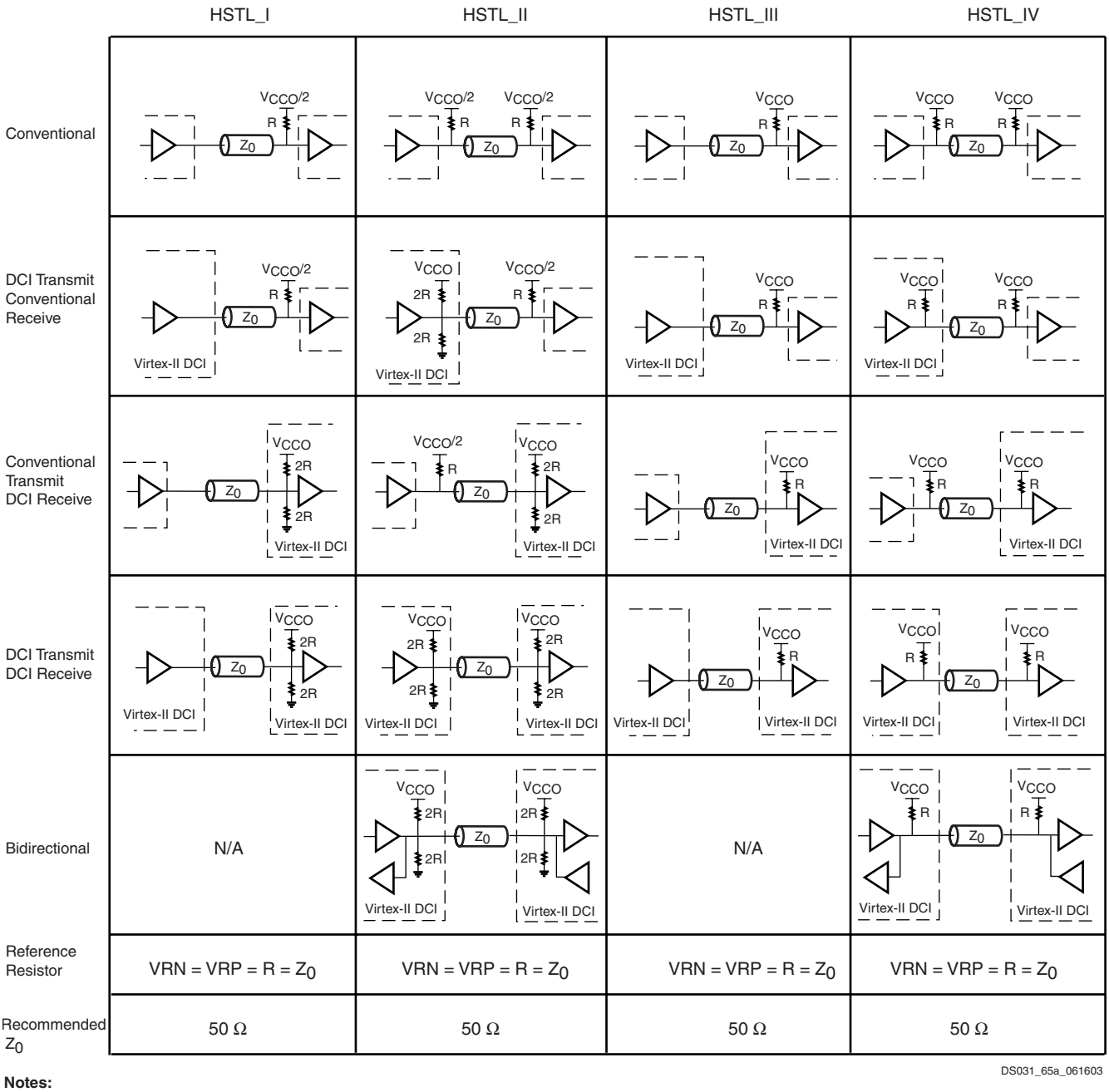
LVDS_25_DCI, LVDSEXT_25_DCI

LVDS_25_DCI and LVDSEXT_25_DCI provide split termination for the P and N inputs only. VRP and VRN should connect to 50Ω resistors.

Equivalently, it provides 100Ω differential impedance between the LVDS inputs.

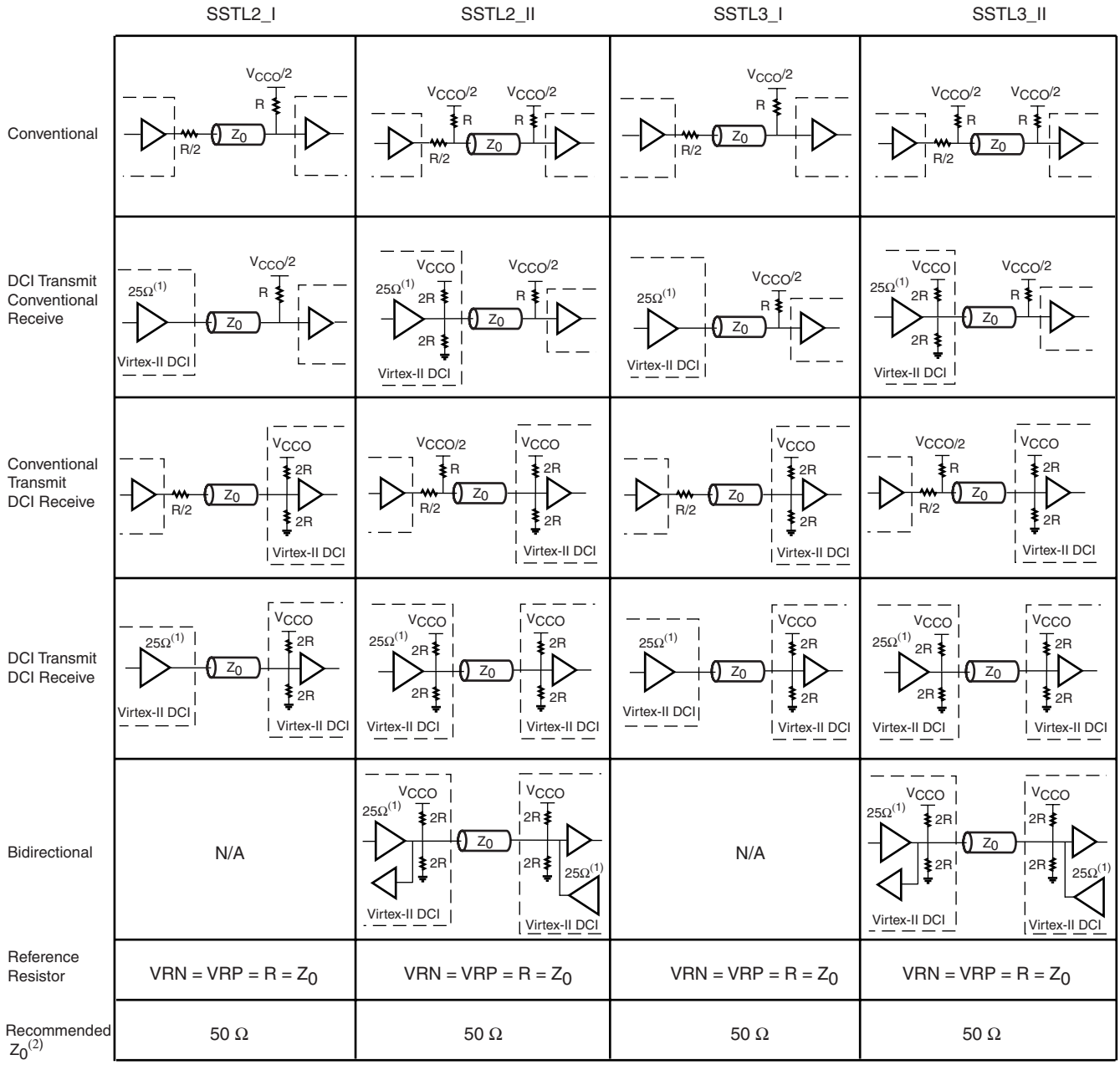
DCI Usage Examples

- Figure 3-108 provides examples illustrating the use of the HSTL_I_DCI, HSTL_II_DCI, HSTL_III_DCI, and HSTL_IV_DCI I/O standards.
- Figure 3-109 provides examples illustrating the use of the SSTL2_I_DCI, SSTL2_II_DCI, SSTL3_I_DCI, and SSTL3_II_DCI I/O standards.
- Figure 3-110 provides examples illustrating the use of the LVDS_25_DCI and LVDSEXT_25_DCI I/O standards.



Notes:
1. Z_0 is the recommended PCB trace impedance.

Figure 3-108: **HSTL DCI Usage Examples**



Notes:
1. The SSTL-compatible 25Ω series resistor is accounted for in the DCI buffer, and it is not DCI controlled.
2. Z_0 is the recommended PCB trace impedance.

DS031_65b_112502

Figure 3-109: **SSTL DCI Usage Examples**

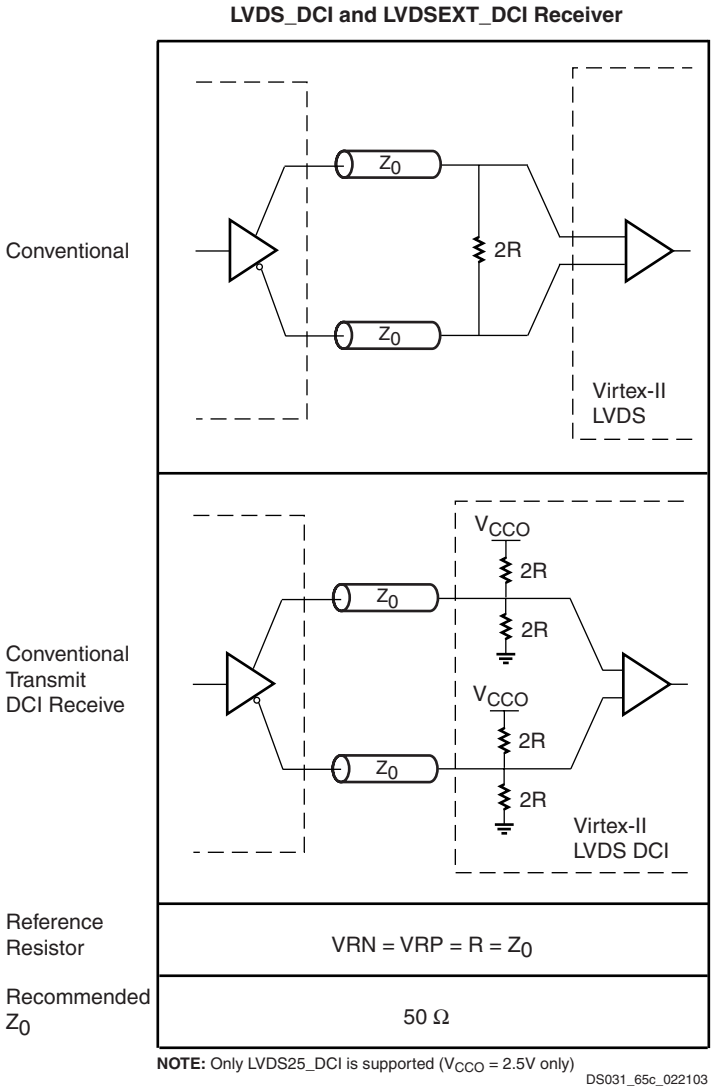


Figure 3-110: **LVDS DCI Usage Examples**

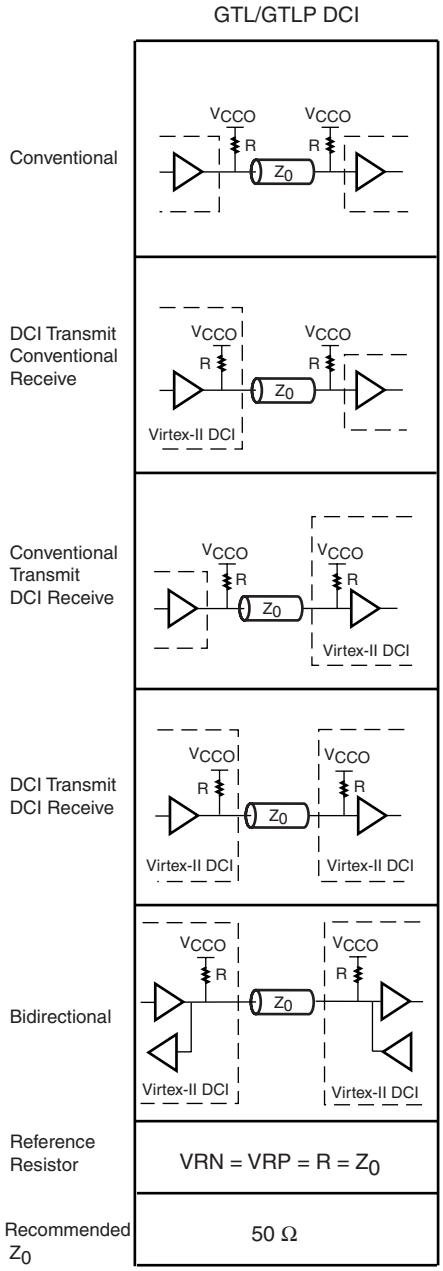


Figure 3-111: **GTL/GTLP DCI Usage Examples**

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

Introduction

Virtex-II devices have dedicated registers in a single IOB to implement input, output, and output with 3-state control Double-Data-Rate (DDR) registers. Input and output DDR is directly accomplished by the two registers on each path, clocked by the rising edges (or falling edges) from two different clock nets. The two clock signals are generated by the DCM and must be 180 degrees out of phase. Output DDR with 3-state requires the use of four registers in the IOB clocked in a similar fashion. Since the introduction of DLLs, Xilinx devices can generate low-skew clock signals that are 180 degrees out of phase, with a 50/50 duty cycle. These clocks reach the DDR registers in the IOB via dedicated routing resources.

Data Flow

Input DDR

Input DDR is accomplished via a single input signal driving two registers in the IOB. Both registers are clocked on the rising edge of their respective clocks. With proper clock forwarding, alternating bits from the input signal are clocked in on the rising edge of the two clocks, which are 180 degrees out of phase. Figure 3-112 depicts the input DDR registers and the signals involved.

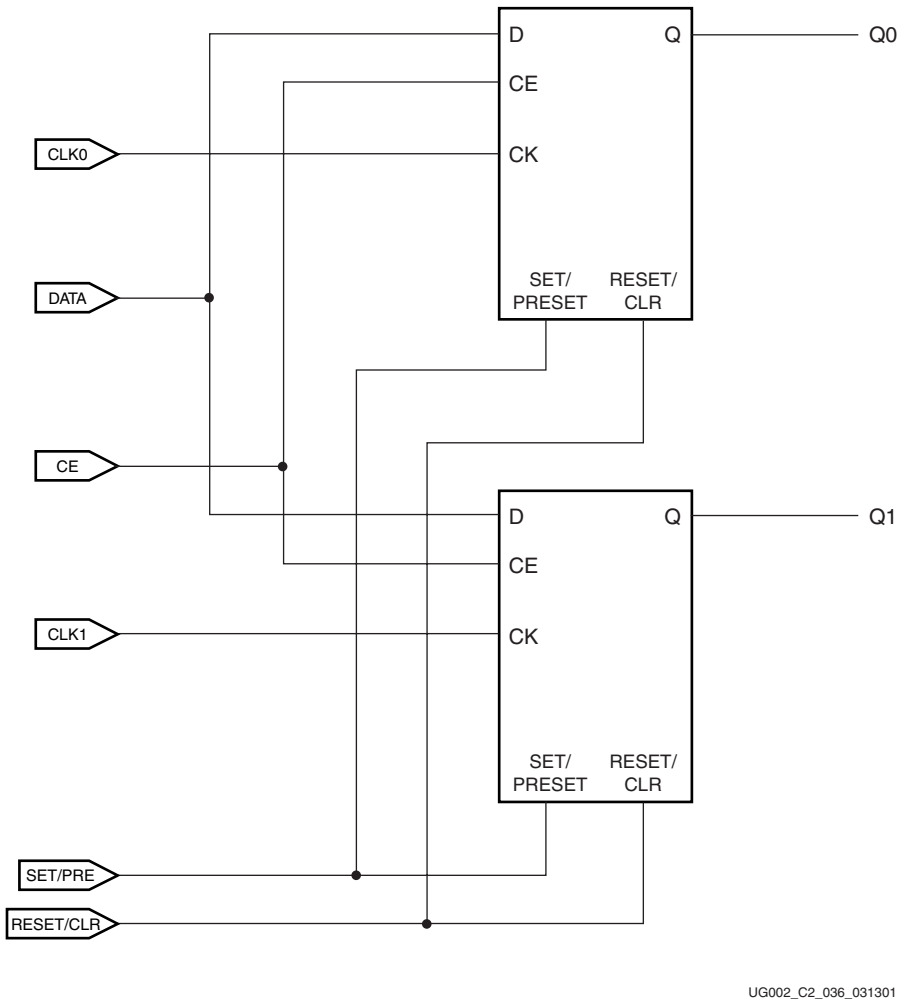


Figure 3-112: **Input DDR**

CLK0 and CLK1 are 180 degrees out of phase. Both registers share the SET/PRE and RESET/CLR lines. As shown in Figure 3-113, alternating bits on the DATA line are clocked in via Q0 and Q1 while CE is High. The clocks are shifted out of phase by the DCM (CLK0 and CLK180 outputs) or by the inverter available on the CLK1 clock input.

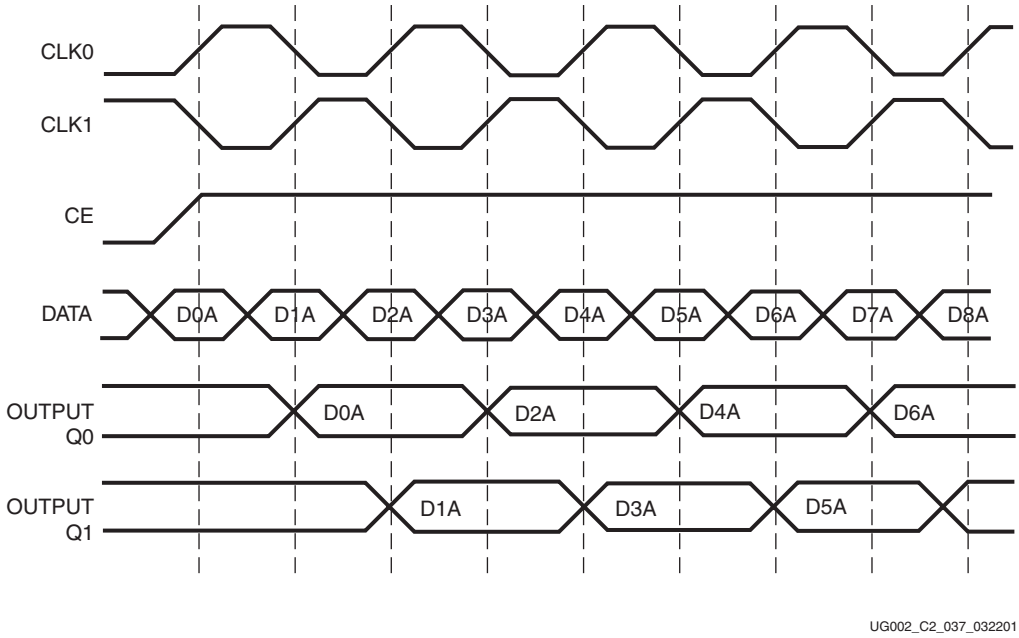


Figure 3-113: **Input DDR Timing Diagram**

Output DDR

Output DDR registers are used to clock output from the chip at twice the throughput of a single rising-edge clocking scheme. Clocking for output DDR is the same as input DDR. The clocks driving both registers are 180 degrees out of phase. The DDR MUX selects the register outputs. The output consists of alternating bits from DATA_1 and DATA_2. Figure 3-114 depicts the output DDR registers and the signals involved.

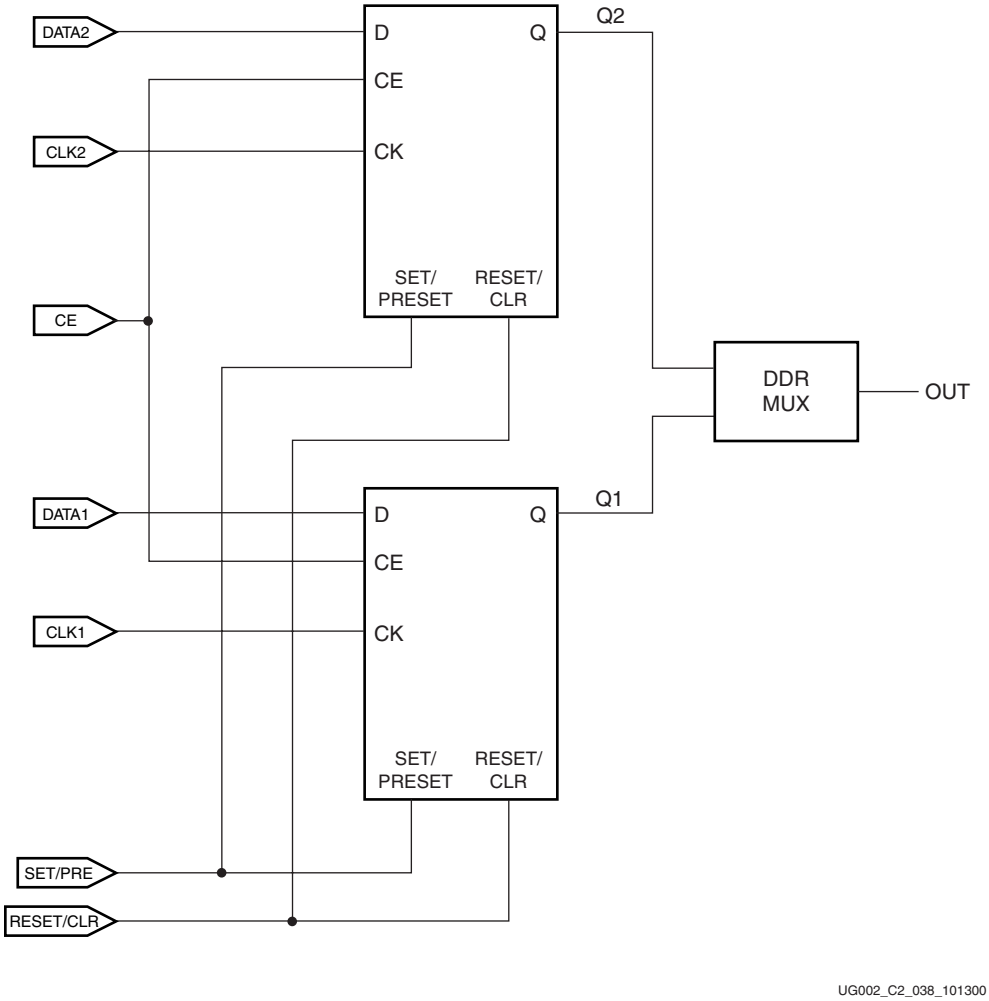


Figure 3-114: **Output DDR**

Both registers share the SET/PRE and RESET/CLR line. Both registers share the CE line which must be High for outputs to be seen on Q1 and Q2. Figure 3-115 shows the data flow for the output DDR registers.

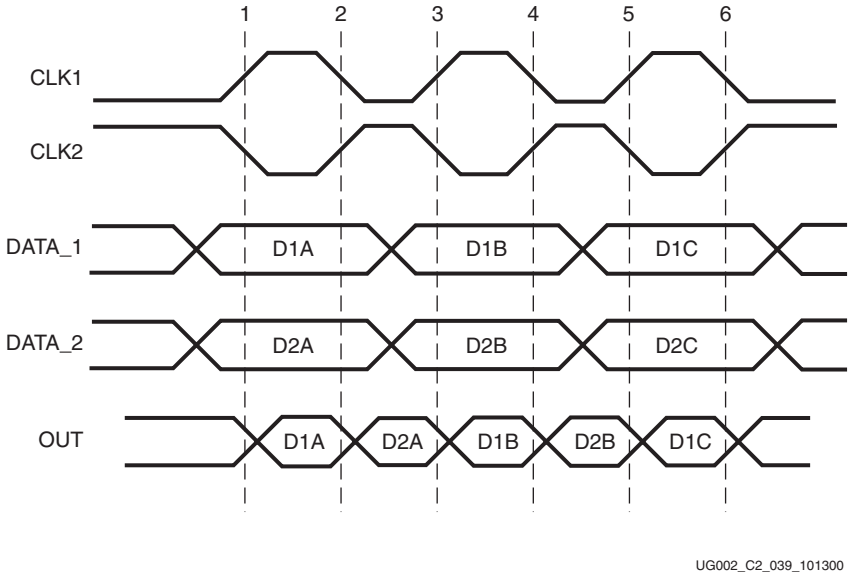


Figure 3-115: **Output DDR Timing Diagram**

Output DDR With 3-State Control

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

The Enable signal is driven by a second DDR MUX (Figure 3-116). This application requires the instantiation of two output DDR primitives.

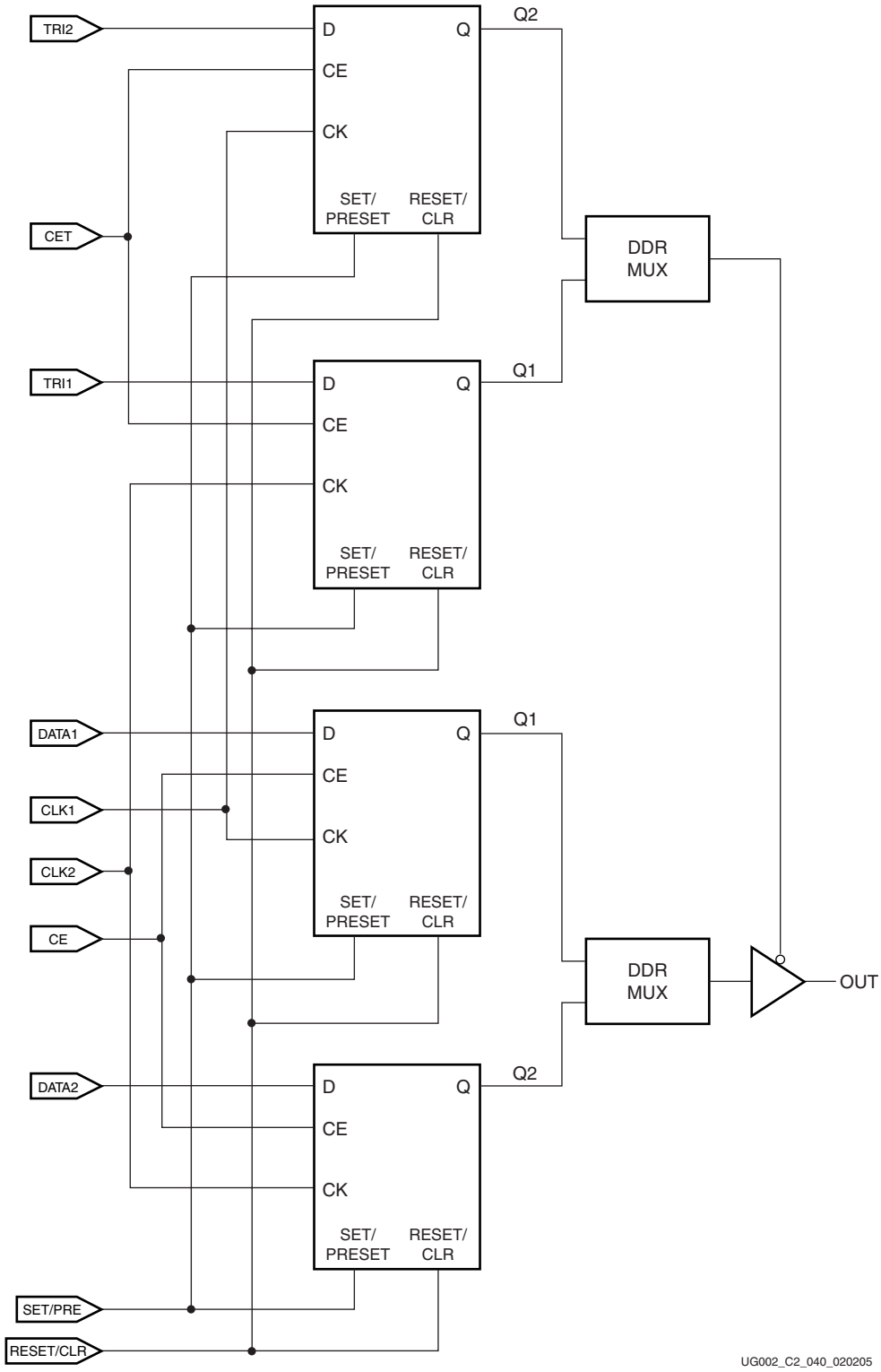


Figure 3-116: **Output DDR With 3-State Control**

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

The DDR registers and 3-state registers are associated by the clock that is driving them. Therefore, the DDR register that is clocked by CLK1 is associated to the 3-state register being clocked by CLK1. The remaining two registers are associated by CLK2. If both 3-state registers are driving a logic High, the output sees a high impedance. If both 3-state registers are driving a logic Low, the output sees the values from the DDR MUX see Figure 3-117).

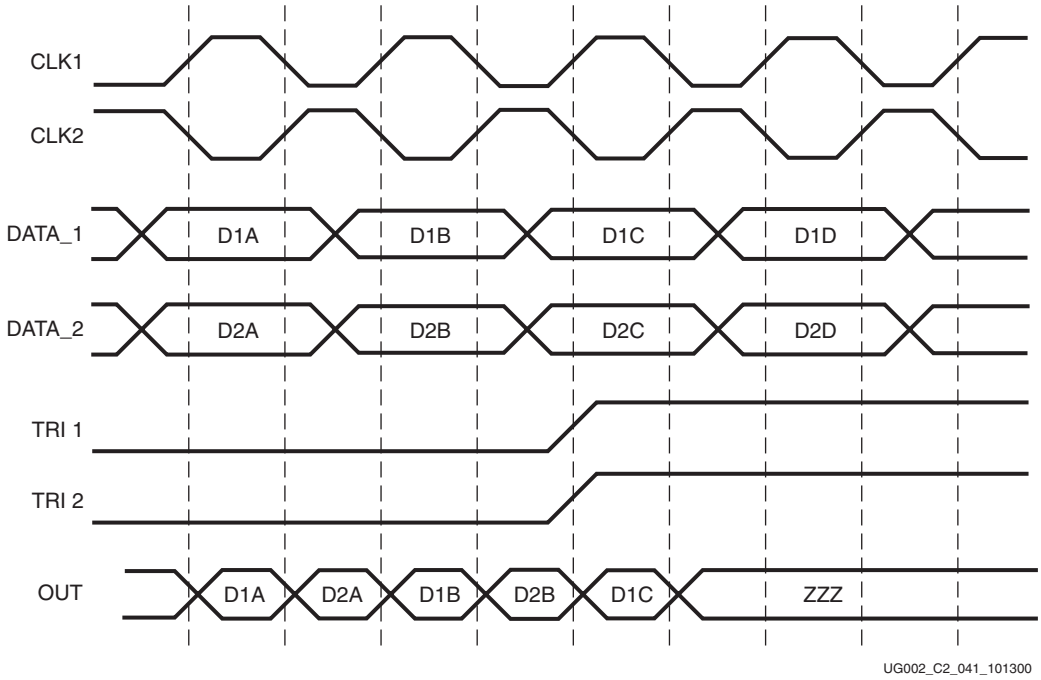


Figure 3-117: **Timing Diagram for Output DDR With 3-State Control**

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

Characteristics

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

Library Primitives

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

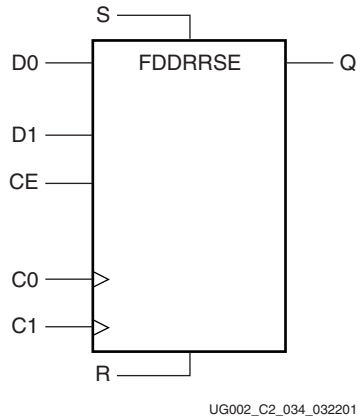


Figure 3-118: **FDDRRSE Symbol: DDR Flip-Flop With Clock Enable and Synchronous Reset and Set**

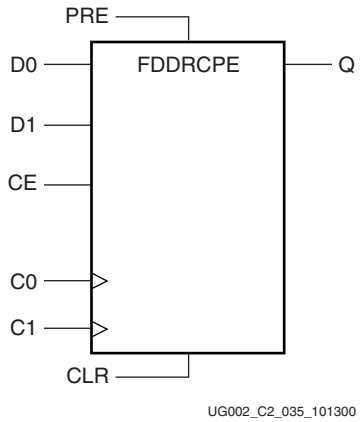


Figure 3-119: **FDDRCPE Symbol: DDR Flip-Flop With Clock Enable and Asynchronous PRESET and CLR**

VHDL and Verilog Instantiation

Examples are available in "VHDL and Verilog Templates" on page 235.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

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

Port Signals

FDDRRSE

Data inputs - D0 and D1

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

Clock Enable - CE

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

Clocks - C0 and C1

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

Synchronous Set - S and Synchronous Reset - R

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

Data Output - Q

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

During normal operation, The value of Q is either D0 or D1. The Data Inputs description above states how the value of Q is chosen.

FDDRCPE

Data inputs - D0 and D1

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

Clock Enable - CE

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

Clocks - C0 and C1

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

Asynchronous Preset - PRE and Asynchronous Clear - CLR

The Preset (PRE) input, when High, sets the Q output High. When the Clear (CLR) input is High, the output is reset to Low.

Data Output - Q

When power is applied, the flip-flop is asynchronously cleared and the output is Low. During normal operation, The value of Q is either D0 or D1. The Data Inputs description above states how the value of Q is chosen.

Initialization in VHDL or Verilog

Output DDR primitives can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the output DDR instantiation and are copied in the EDIF output file to be compiled by Xilinx tools. The VHDL code simulation uses a `generic` parameter to pass the attributes. The Verilog code simulation uses the `defparam` parameter to pass the attributes.

The DDR code examples (in VHDL ad Verilog) illustrate the following techniques.

Location Constraints

DDR instances can have LOC properties attached to them to constrain pin placement. The LOC constraint uses the following form.

```
NET <net_name> LOC=A8;
```

where "A8" is a valid I/O pin location.

This device has a shared routing resource connecting the ICLK and OTCLK pins on pairs of IOBs. If two adjacent IOBs using DDR registers do not share the same clock signals on their clock pins (ICLK1, ICLK2, OTCLK1, and OTCLK2), or a DDR register locates next to a single data rate register that uses a different clock, one of the clock signals will be unroutable.

The IOB pairing is identical to the LVDS IOB pairs. Hence, the package pin-out table can also be used for pin assignment to avoid conflict.

Applications

DDR SDRAM

The DDR SDRAM is an enhancement to the Synchronous DRAM by effectively doubling the data throughput of the memory device. Commands are registered at every positive clock edge. Input data is registered on both edges of the data strobe, and output data is referenced to both edges of the data strobe, as well as both edges of the clock.

Clock Forwarding

DDR can be used to forward a copy of the clock on the output. This can be useful for propagating a clock along with double-data-rate data that has an identical delay. It is also useful for multiple clock generation, where there is a unique clock driver for every clock load.

VHDL and Verilog Templates

VHDL and Verilog templates are available for output, output with 3-state enable, and input DDR registers.

Input DDR

To implement an Input DDR application, paste the following template in your code.

DDR_input.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DDR_Input is

  Port (
    clk : in std_logic;
    d : in std_logic;
    rst : in std_logic;
    q1 : out std_logic;
```

```
    q2 : out std_logic
       );

end DDR_Input;

--Describe input DDR registers (behaviorally) to be inferred

architecture behavioral of DDR_Input is

begin

q1reg : process (clk, d, rst)

begin
   if rst='1' then --asynchronous reset, active high
     q1 <= '0';
   elsif clk'event and clk='1' then  --Clock event - posedge
     q1 <= d;

   end if;
end process;

q2reg : process (clk, d, rst)

begin
   if rst='1' then --asynchronous reset, active high
      q2 <= '0';
   elsif clk'event and clk='0' then  --Clock event - negedge
      q2 <= d;
   end if;
end process;

end behavioral;


-- NOTE: You must include the following constraints in the .ucf
-- file when running back-end tools,
-- in order to ensure that IOB DDR registers are used:
--
-- INST "q2_reg" IOB=TRUE;
-- INST "q1_reg" IOB=TRUE;
--
-- Depending on the synthesis tools you use, it may be required to
-- check the edif file for modifications to
-- original net names...in this case, Synopsys changed the
-- names: q1 and q2 to q1_reg and q2_reg
```

DDR_input.v

```
module DDR_Input (data_in , q1, q2, clk, rst);

input data_in, clk, rst;
output q1, q2;
reg q1, q2;


//Describe input DDR registers (behaviorally) to be inferred

always @ (posedge clk or posedge rst) //rising-edge DDR reg. and
asynchronous reset

  begin
```

```
  if (rst)
     q1 = 1'b0;
  else
     q1 = data_in;
   end

always @ (negedge clk or posedge rst) //falling-edge DDR reg. and
asynchronous reset

begin
  if (rst)
     q2 = 1'b0;
  else
     q2 = data_in;
   end

endmodule

/* NOTE: You must include the following constraints in the .ucf file when
running back-end tools, \
  in order to ensure that IOB DDR registers are used:

INST "q2_reg" IOB=TRUE;
INST "q1_reg" IOB=TRUE;

Depending on the synthesis tools you use, it may be required to check the
edif file for modifications to
original net names...in this case, Synopsys changed the names: q1 and q2
to q1_reg and q2_reg
*/
```

Output DDR

To implement an Output DDR application, paste the following template in your code.

DDR_out.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- pragma translate_off
LIBRARY UNISIM;
use UNISIM.VCOMPONENTS.ALL;
--pragma translate_on

entity DDR_Output is
  Port(
   clk : in std_logic; --clk and clk180 can be outputs from the DCM or
clk180 can be the
   clk180 : in std_logic;  --logical inverse of clk (the inverter is
located in the IOB and will be inferred.
   d0 : in std_logic;  --data in to fddr
   d1 : in std_logic;  --data in to fddr
   ce : in std_logic;  --clock enable
   rst : in std_logic;  --reset
   set : in std_logic;  --set
   q : out std_logic  --DDR output
   );

end DDR_Output;

architecture behavioral of DDR_Output is

component FDDRRSE
```

```
  port(
    Q  : out std_logic;
    D0 : in std_logic;
    D1 : in std_logic;
    C0 : in std_logic;
    C1 : in std_logic;
    CE : in std_logic;
    R  : in std_logic;
    S : in std_logic
    );
end component;

begin

U0: FDDRRSE
  port map (
    Q => q,
    D0 => d0,
    D1 => d1,
    C0 => clk,
    C1 => clk180,
    CE => ce,
    R => rst,
    S => set
    );


end behavioral;
```

DDR_out.v

```
module DDR_Output (d0 , d1, q, clk, clk180, rst, set, ce);

input d0, d1, clk, clk180, rst, set, ce;
output q;


//Synchronous Output DDR primitive instantiation

FDDRRSE U1  ( .D0(d0),
              .D1(d1),
              .C0(clk),
              .C1(clk180),
              .CE(ce),
              .R(rst),
              .S(set),
              .Q(q)
            );
endmodule
```

Output DDR With 3-State Enable

To implement an Output DDR with 3-state Enable, paste the following template in your code:

DDR_3state.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- pragma translate_off
LIBRARY UNISIM;
use UNISIM.VCOMPONENTS.ALL;
--pragma translate_on
```

```
entity DDR_3state is
  Port(
    clk : in std_logic; --clk and clk180 can be outputs from the DCM or
clk180 can be the
    clk180 : in std_logic; --logical inverse of clk (the inverter is
located in the IOB and will be inferred.
    d0 : in std_logic; --data in to fddr
    d1 : in std_logic; --data in to fddr
    ce : in std_logic; --clock enable
    set : in std_logic; --set
    rst : in std_logic; --reset
    en0 : in std_logic; --enable signal
    en1 : in std_logic; --enable signal
    data_out : out std_logic --data seen at pad
  );

end DDR_3state;

architecture behavioral of DDR_3state is

signal ddr_out, tri : std_logic;

component FDDRRSE
   port  (
    Q   : out std_logic;
    D0  : in std_logic;
    D1  : in std_logic;
    C0  : in std_logic;
    C1  : in std_logic;
    CE  : in std_logic;
    R   : in std_logic;
    S   : in std_logic
          );
end component;


begin

--Instantiate Ouput DDR registers
U0: FDDRRSE port map(Q  => tri,
       D0 => en0,
       D1 => en1,
       C0 => clk,
       C1 => clk180,
       CE => ce,
       R  => rst,
       S  => set
      );

--Instantiate three-state DDR registers
U1: FDDRRSE port map(   Q  => ddr_out,
       D0 => d0,
       D1 => d1,
       C0 => clk,
       C1 => clk180,
       CE => ce,
       R  => rst,
       S  => set
      );

--inferr the 3-State buffer
```

```
process(tri, ddr_out)
begin
  if tri = '1' then
     data_out <= 'Z';
  elsif tri = '0' then
     data_out <= ddr_out;
  end if;
end process;

end behavioral;
```

DDR_3state.v

```
module DDR_3state (d0 , d1, data_out, en_0, en_1, clk, clk180, rst, set,
ce);

input d0, d1, clk, clk180, rst, set, ce, en_0, en_1;

output data_out;
reg data_out;

wire q, q_tri;

//Synchronous Output DDR primitive instantiation

FDDRRSE U1  ( .D0(d0),
              .D1(d1),
              .C0(clk),
              .C1(clk180),
              .CE(ce),
              .R(rst),
              .S(set),
              .Q(q)
            );

//Synchronous 3-State DDR primitive instantiation

FDDRRSE U2  ( .D0(en_0),
              .D1(en_1),
              .C0(clk),
              .C1(clk180),
              .CE(ce),
              .R(rst),
              .S(set),
              .Q(q_tri)
            );

//3-State buffer description

always @ (q_tri or q)
  begin
  if (q_tri)
    data_out = 1'bz;
  else
    data_out = q;
  end

endmodule
```

Using LVDS I/O

Introduction

Low Voltage Differential Signaling (LVDS) is a very popular and powerful high-speed interface in many system applications. Virtex-II I/Os are designed to comply with IEEE electrical specifications for LVDS to make system and board design easier. With the addition of an LVDS current-mode driver in the IOBs, which eliminates the need for external source termination in point-to-point applications, and with the choice of two different voltage modes and an extended mode, Virtex-II devices provide the most flexible solution for doing an LVDS design in an FPGA.

Table 3-59 lists all LVDS standards that are available for Virtex-II devices.

Table 3-59: **Available LVDS Standards**

IOSTANDARD	Input (IBUFDS)	Output (OBUFDS)	3-State (OBUFTDS)	Clock (IBUFGDS)
LVDS_25	√	√	√	√
LVDS_33	√	√	√	√
LVDSEXT_25(1)	√	√	√	√
LVDSEXT_33(1)	√	√	√	√
LVDS_25_DCI	√			√
LVDSEXT_25_DCI(1)	√			√

Notes:

1. **LVDSEXT = Extended mode LVDS buffer.** This buffer provides a higher drive capability and voltage swing (350 - 750 mV), which makes it ideal for long-distance or cable LVDS links. The output AC characteristics of this LVDS Extended Mode driver are not within the EIA/TIA specifications. This LVDS Extended Mode driver is intended for situations that require higher drive capabilities in order to produce an LVDS signal that is within EIA/TIA specification at the receiver.

Figure 3-120 illustrates the LVDS input and clock buffer primitives shown in Table 3-60. The pin names used are the same as those used in the HDL library primitives.

Table 3-60: **LVDS Input and Clock Buffer Primitives**

LVDS Inputs	LVDS Clocks
IBUFDS_LVDS_25	IBUFGDS_LVDS_25
IBUFDS_LVDS_33	IBUFGDS_LVDS_33
IBUFDS_LVDSEXT_25	IBUFGDS_LVDSEXT_25
IBUFDS_LVDSEXT_33	IBUFGDS_LVDSEXT_33
IBUFDS_LVDS_25_DCI	IBUFGDS_LVDS_25_DCI
IBUFDS_LVDSEXT_25_DCI	IBUFGDS_LVDSEXT_25_DCI

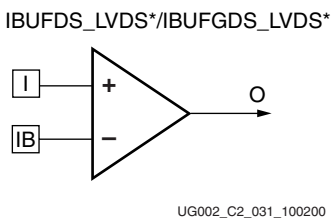


Figure 3-120: **LVDS Input and Clock Primitives**

To create an LVDS input, instantiate the desired mode (2.5 V, 3.3 V, or Extended) LVDS input buffer. Notice that the P and N channels are included in the primitive (I = P, IB = N). Software automatically uses the appropriate pin from an adjacent IOB for the N channel. The same applies to LVDS clocks: Use IBUFGDS_LVDS*

LVDS Input HDL Examples

VHDL Instantiation

```
U1: IBUFDS_LVDS_25
  port map  (
             I => data_in_P,
             IB => data_in_N
             O => data_in
            );
```

Verilog Instantiation

```
IBUFDS_LVDS_25   U1    ( .I(data_in_P),
                         .IB(data_in_N),
                         .O(data_in)
                       );
```

Port Signals

I = P-channel data input to the LVDS input buffer

IB = N-channel data input to the LVDS input buffer

O = Non-differential input data from LVDS input buffer

Location Constraints

```
NET "data_in_P" LOC= "<pin_location>";
```

LVDS Receiver Termination

All LVDS receivers require standard termination. Figure 3-121 is an example of a typical termination for an LVDS receiver on a board with 50Ω transmission lines.

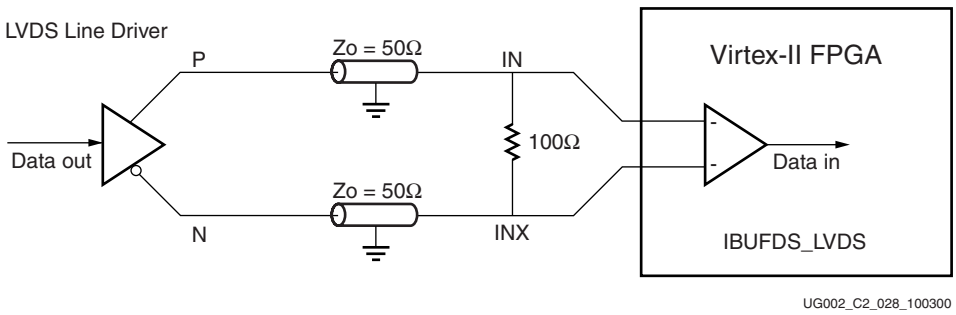


Figure 3-121: **LVDS Receiver Termination**

Creating an LVDS Output Buffer

Figure 3-122 illustrates the LVDS output buffer primitives:

- OBUFDS_LVDS_25
- OBUFDS_LVDS_33
- OBUFDS_LVDSEXT_25
- OBUFDS_LVDSEXT_33

The pin names used are the same as those used in the HDL library primitives.

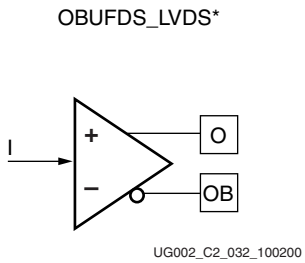


Figure 3-122: **LVDS Output Buffer Primitives**

To create an LVDS output, instantiate the desired mode (2.5, 3.3V, or Extended) LVDS output buffer. Notice that the P and N channels are included in the primitive (O = P, OB = N). Software automatically uses the appropriate pin from an adjacent IOB for the N channel.

LVDS Output HDL Examples

VHDL Instantiation

```
U1: OBUFDS_LVDS_25
  port map  (
              I => data_out,
              O => data_out_P,
              OB => data_out_N
            );
```

Verilog Instantiation

```
OBUFDS_LVDS_25 U1  ( .I(data_out),
                     .O(data_out_P),
                     .OB(data_out_N)
                   );
```

Port Signals

I = data input to the LVDS input buffer

O = P-channel data output

OB = N-channel data output

Location Constraints

```
NET "data_out_P" LOC= "<pin_location>";
```

LVDS Transmitter Termination

The Virtex-II LVDS transmitter does not require any termination. Table 3-59 lists primitives that correspond to the Virtex-II LVDS current-mode drivers. Virtex-II LVDS current-mode drivers are a true current source and produce the proper (IEEE/EIA/TIA compliant) LVDS

signal. Figure 3-123 illustrates a Virtex-II LVDS transmitter on a board with 50Ω transmission lines.

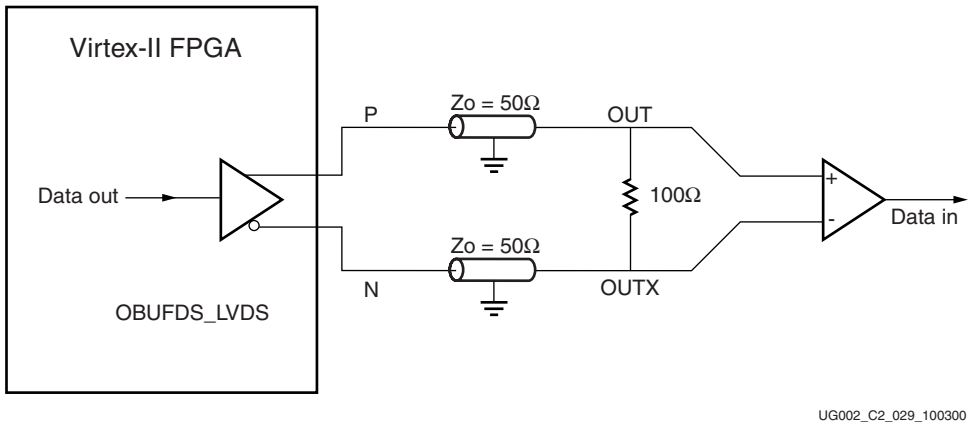


Figure 3-123: **LVDS Transmitter Termination**

Creating an LVDS Output 3-State Buffer

Figure 3-124 illustrates the LVDS 3-State buffer primitives:

- OBUFTDS_LVDS_25
- OBUFTDS_LVDS_33
- OBUFTDS_LVDSEXT_25
- OBUFTDS_LVDSEXT_33

The pin names used are the same as those used in the HDL library primitives.

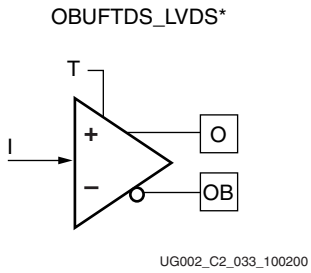


Figure 3-124: **LVDS 3-State Primitives**

To create an LVDS 3-State output, instantiate the desired mode (2.5V, 3.3V, or Extended) LVDS 3-State buffer. Notice that the P and N channels are included in the primitive (O = P, OB = N). Software automatically uses the appropriate pin from an adjacent IOB for the N channel.

LVDS 3-State HDL Example

VHDL Instantiation

```
U1: OBUFTDS_LVDS_25
  port map  (
              I => data_out,
              T => tri,
              O => data_out_P,
              OB => data_out_N
            );
```

Verilog Instantiation

```
OBUFTDS_LVDS_25  U1  (  .I(data_out),
                        .T(tri),
                        .O(data_out_P),
                        .OB(data_out_N)
                     );
```

Port Signals

I = data input to the 3-state output buffer

T = 3-State control signal

O = P-channel data output

OB = N-channel data output

Location Constraints

```
NET "data_out_P" LOC = "<pin_location>";
```

LVDS 3-State Termination

The Virtex-II LVDS 3-state buffer does not require any termination. Table 3-59 lists primitives that correspond to Virtex-II LVDS current-mode drivers. These drivers are a true current source, and they produce the proper (IEEE/EIA/TIA compliant) LVDS signal. Figure 3-125 illustrates a simple redundant point-to-point LVDS solution with two LVDS 3-state transmitters sharing a bus with one LVDS receiver and the required termination for the circuit.

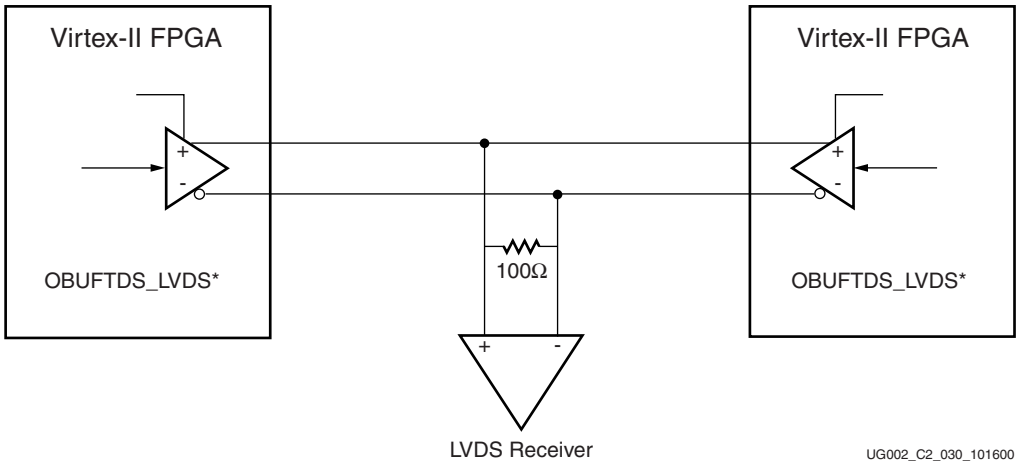


Figure 3-125: **LVDS 3-State Termination**

Creating a Bidirectional LVDS Buffer

Since LVDS is intended for point-to-point applications, BLVDS (Bus-LVDS) is not an IEEE/EIA/TIA standard implementation and requires careful adaptation of I/O and PCB layout design rules. The primitive supplied in the software library for bi-directional LVDS does not use the Virtex-II LVDS current-mode driver. Therefore, source termination is required. Refer to **XAPP243** for examples of BLVDS termination.

The following are VHDL and Verilog instantiation examples of Virtex-II BLVDS primitves.

VHDL Instantiation

```
blvds_io: IOBUFDS_BLVDS_25
port map  (
           I => data_out,
           O => data_in,
           T => tri,
           IO => data_IO_P,
           IOB => data_IO_N
          );
```

Verilog Instantiation

```
IOBUFDS_BLVDS_25  blvds_io  (  .I(data_out),
                               .O(data_in),
                               .T(tri),
                               .IO(data_IO_P),
                               .IOB(data_IO_N)
                            );
```

Port Signals

I = data output: internal logic to LVDS I/O buffer

T = 3-State control to LVDS I/O buffer

IO = P-channel data I/O to or from BLVDS pins

IOB = N-channel data I/O to or from BLVDS pins

O = Data input: off-chip data to LVDS I/O buffer

Location Constraints

Only the P or N channel must be constrained. Software automatically places the corresponding channel of the pair on the appropriate pin.

LDT

Lightning Data Transport (LDT) is a new high speed interface and protocol introduced by Advanced Micro Devices. LDT is a differential signaling based interface that is very similar to LVDS. Virtex-II IOBs are equipped with LDT buffers. These buffers also have corresponding software primitives as follows:

IBUFDS_LDT_25

IBUFGDS_LDT_25

OBUFDS_LDT_25

OBUFTDS_LDT_25

LDT Implementation

LDT implementation is the same as LVDS with DDR, so follow all of the rules and guidelines set forth earlier in this chapter for LVDS-DDR, and replace the LVDS buffer with the corresponding LDT buffer. For more information on Virtex-II LDT electrical specification, refer to the **Virtex-II Data Sheet (DS031)**.

Using LVPECL I/O

Introduction

Low Voltage Positive Emitter-Coupled Logic (LVPECL) is a very popular and powerful high-speed interface in many system applications. Virtex-II I/Os are designed to comply with the EIA/TIA electrical specifications for 3.3V LVPECL to make system and board design easier.

Table 3-61 lists all LVPECL primitives that are available for Virtex-II devices.

Table 3-61: **Available Virtex-II LVPECL Primitives**

Input	Output	3-State	Clock	Bi-Directional
IBUFDS_LVPECL_33	OBUFDS_LVPECL_33	OBUFTDS_LVPECL_33	IBUFGDS_LVPECL_33	

Creating an LVPECL Input/Clock Buffer

Figure 3-126 illustrates the LVPECL input and clock buffer primitives shown in Table 3-62. The pin names used are the same as those used in the HDL library primitives.

Table 3-62: **LVPECL Input and Clock Buffer Primitives**

LVPECL Inputs	LVPECL Clocks
IBUFDS_LVPECL_33	IBUFGDS_LVPECL_33

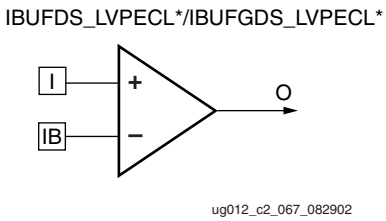


Figure 3-126: **LVPECL Input and Clock Primitives**

To create an LVPECL input, instantiate the desired mode (3.3V or Extended) LVPECL input buffer. Notice that the P and N channels are included in the primitive (I = P, IB = N). Software automatically uses the appropriate pin from an adjacent IOB for the N channel. The same applies to LVPECL clocks: Use IBUFGDS_LVPECL*.

LVPECL Input HDL Examples

VHDL Instantiation

```
U1: IBUFDS_LVPECL_33
  port map  (
              I => data_in_P,
              IB => data_in_N
              O => data_in
            );
```

Verilog Instantiation

```
IBUFDS_LVPECL_33 U1    ( .I(data_in_P),
                         .IB(data_in_N),
                         .O(data_in)
                       );
```

Port Signals

I = P-channel data input to the LVPECL input buffer

IB = N-channel data input to the LVPECL input buffer

O = Non-differential input data from LVPECL input buffer

Location Constraints

```
NET "data_in_P" LOC= "<pin_location>";
```

LVPECL Receiver Termination

All LVPECL receivers require standard termination. Figure 3-127 is an example of a typical termination for an LVPECL receiver on a board with 50Ω transmission lines.

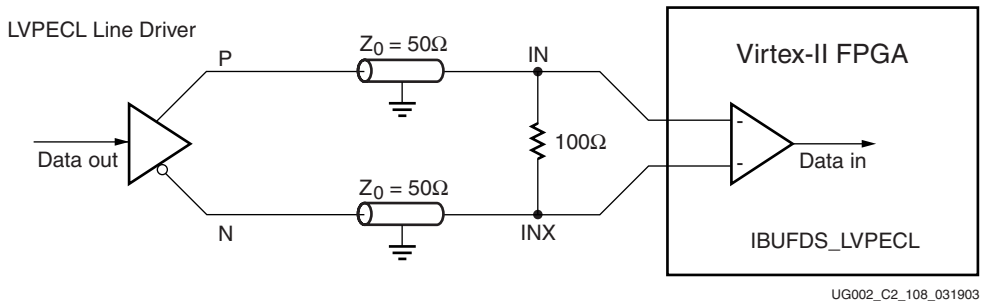


Figure 3-127: **LVPECL Receiver Termination**

Creating an LVPECL Output Buffer

Figure 3-128 illustrates the LVDS output buffer primitives:

- OBUFDS_LVPECL_33
- OBUFTDS_LVPECL_33

The pin names used are the same as those used in the HDL library primitives.

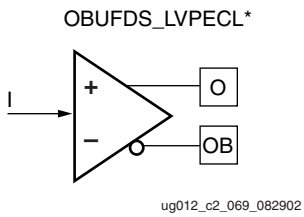


Figure 3-128: **LVPECL Output Buffer Primitives**

To create an LVPECL output, instantiate the desired mode LVPECL output buffer. Notice that the P and N channels are included in the primitive (O = P, OB = N). Software automatically uses the appropriate pin from an adjacent IOB for the N channel.

LVPECL Output HDL Examples

VHDL Instantiation

```
U1: OBUFDS_LVPECL_33
  port map  (
              I => data_out,
              O => data_out_P,
              OB => data_out_N
```

```
);
```

Verilog Instantiation

```
OBUFDS_LVPECL_33U1 ( .I(data_out),
                     .O(data_out_P),
                     .OB(data_out_N)
                   );
```

Port Signals

I = data input to the LVPECL input buffer

O = P-channel data output

OB = N-channel data output

Location Constraints

```
NET "data_out_P" LOC= "<pin_location>";
```

LVPECL Transmitter Termination

The Virtex-II LVPECL transmitter requires the termination shown in Figure 3-129, which illustrates a Virtex-II LVPECL transmitter on a board with 50Ω transmission lines. Table 3-61 lists primitives that correspond to the Virtex-II LVPECL drivers.

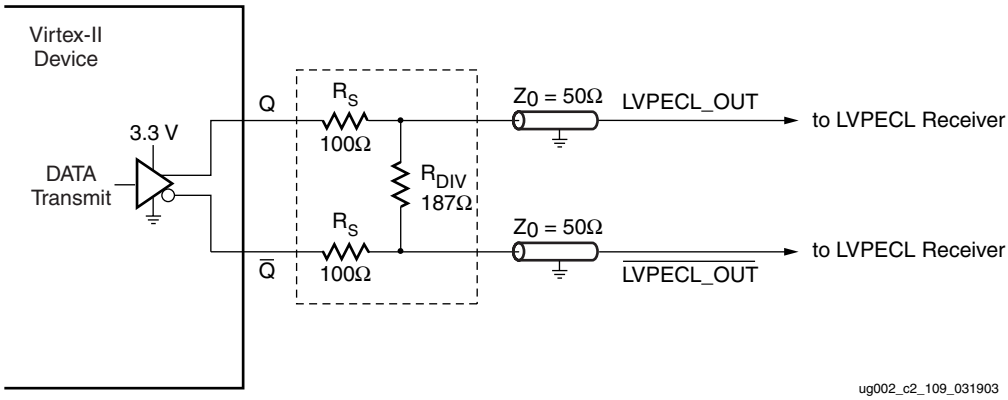


Figure 3-129: **LVPECL Transmitter Termination**

Using Bitstream Encryption

Virtex-II devices have an on-chip decryptor that can be enabled to make the configuration bitstream (and thus the whole logic design) secure. The user can encrypt the bitstream in the Xilinx software, and the Virtex-II chip then performs the reverse operation, decrypting the incoming bitstream, and internally recreating the intended configuration.

This method provides a very high degree of design security. Without knowledge of the encryption/decryption key or keys, potential pirates cannot use the externally intercepted bitstream to analyze, or even to clone the design. System manufacturers can be sure that their Virtex-II implemented designs cannot be copied and reverse engineered.

The Virtex-II devices store the internal decryption keys in a few hundred bits of dedicated RAM, backed up by a small externally connected battery. At <100 nA load, the endurance of the battery is only limited by its shelf life. Refer to Appendix C: Choosing the Battery for V_{BATT} for suggestions on choosing a backup battery.

The method used to encrypt the data is Data Encryption Standard (DES). This is an official standard supported by the National Institute of Standards and Technology (NIST) and the U. S. Department of Commerce. DES is a symmetric encryption standard that utilizes a 56-bit key. Because of the increased sophistication and speed of today's computing hardware, single DES is no longer considered to be secure. However, the Triple Data Encryption Algorithm (TDEA), otherwise known as triple DES, is authorized for use by U. S. federal organizations to protect sensitive data and is used by many financial institutions to protect their transactions. Triple DES has yet to be cracked. Both DES and triple DES are available in Virtex-II devices.

What DES Is

DES and triple DES are symmetric encryption algorithms. This means that the key to encrypt and the key to decrypt are the same. The security of the data is kept by keeping the key secret. This contrasts to a public key system, like RSA or PGP. One thing to note is that Virtex-II devices use DES in Cipher Block Chaining mode. This means that each block is combined with the previous encrypted block for added security. DES uses a single 56-bit key to encrypt 64-bit blocks one at a time.

How Triple DES is Different

Triple DES uses three keys (known as a key bundle or key set), and the encryption algorithm is repeated for each of those keys. If $E_K(I)$ and $D_K(I)$ denote the encryption and decryption of a data block I using key K, the Triple DES encryption algorithm is as follows (known as E-D-E):

$$Output_{encrypted} = E_{K3}(D_{K2}(E_{K1}(I)))$$

And the decryption algorithm is as follows (known as D-E-D):

$$Output_{decrypted} = D_{K1}(E_{K2}(D_{K3}(I)))$$

$K_1 = K_2 = K_3$ gives the same result as single DES.

For a detailed description of the DES standard, refer to:

http://www.itl.nist.gov/fipspubs/fip46-2.htm

For a popular description of the origin and the basic concept of DES and many other older and newer encryption schemes, see the recent best-seller:

The Code Book by Simon Singh, Doubleday 1999, ISBN 0-385-49531-5

Classification and Export Considerations

Virtex-II FPGAs have been classified by the U. S. Department of Commerce as an FPLD (3A001.a.7), which is the same classification as current FPGAs. Only the decryptor is on-chip and can only be used to decrypt an incoming bitstream, so the classification has not changed and no new paperwork is required. The software has been classified under ECCN#:5D002 and can be exported globally under license exception ENC. No changes to current export practices are necessary.

Creating Keys

For Virtex-II, DES or triple DES (TDEA) can be used. DES uses a single 56-bit key, where triple-DES always uses three such keys. All of the keys can be chosen by the BitGen program at random, or can be explicitly specified by the user.

Virtex-II devices can have six separate keys programmed into the device. A particular Virtex-II device can store two sets of triple-DES keys and can thus accept alternate bitstreams from two competing IP vendors, without providing access to each other's design. However, all of the keys must be programmed at once.

An encrypted bitstream is created by the BitGen program. Keys and key options can be chosen in two ways: by command-line arguments to BitGen, or by specifying a KeyFile (with the –g KeyFile command-line option). The BitGen options relevant to encryption are listed in Table 3-63.:

Table 3-63: **BitGen Encryption Options**

Option	Description	Values (default first where appropriate)
Encrypt	Whether to encrypt the bitstream	No, Yes
Key0	DES Key 0	pick, <*hex string*>
Key1	DES Key 1	pick, <*hex string*>
Key2	DES Key 2	pick, <*hex string*>
Key3	DES Key 3	pick, <*hex string*>
Key4	DES Key 4	pick, <*hex string*>
Key5	DES Key 5	pick, <*hex string*>
KeyFile	Location of separate key definition file	<*string*>
Keyseq0	Set the key sequence for key 0 (S = single, F = first, M = middle, L = last)	S,F,M,L
Keyseq1	Set the key sequence for key 1	S,F,M,L
Keyseq2	Set the key sequence for key 2	S,F,M,L
Keyseq3	Set the key sequence for key 3	S,F,M,L
Keyseq4	Set the key sequence for key 4	S,F,M,L
Keyseq5	Set the key sequence for key 5	S,F,M,L
StartKey	Key number to start decryption	0,3
StartCBC	Constant Block Chaining start value	pick, <*string*>

The key sequence (Keyseq) is set to S for single key encryption, F for first key in multi-key encryption, M for middle key in multi-key encryption, and L for last key in multi-key

encryption. When the KeyFile option is specified, BitGen looks in that file for all other DES key options listed above. An example for the input KeyFile using triple DES is:

```
# Comment for key file
Key 0 0x9ac28ebeb2d83b;
Key 1 pick;
Key 2 string for my key;
Key 3 0x00000000000000;
Key 4 8774eb3ebb4f84;
Keyseq 0 F;
Keyseq 1 M;
Keyseq 2 L;
Keyseq 3 F;
Keyseq 4 M;
Keyseq 5 L;
Key StartCBC 503f2f655b1b2f82;
StartKey 0;
```

Every key is given in the output key file, with unused key locations set to "0x0000000000000000." The proper key sequence prefix is added for all used keys. The prefix is preserved for unused keys, if the user specified a value. The output key file has the same base file name as the **.bit** file, but with a **.nky** file extension.

The command line equivalent of the input key file above is as follows:

```
bitgen -g security:level1 -g Encrypt:Yes -g Key0: 0x9ac28ebeb2d83b -g
Key1:pick -g Key2:- string for my key- -g Key30x00000000000000 -g
Key4:8774eb3ebb4f84 -g Keyseq0:F, -g Keyseq1:M, -gKeyseq2:L -g
Keyseq3:F -g Keyseq4:M -g Keyseq5:L -g StartCBC:503f2f655b1b2f82 -g
StartKey:0 myinput.ncd
```

If the key file is used, the command line is as follows:

```
Bitgen -g security:level1 -g Encrypt:Yes -g KeyFile:mykeyfile
myinput.ncd
```

The output key file from either of the above inputs looks something like this:

```
Device 2v40CS144;
Key 0 0x9ac28ebeb2d83b;
Key 1 0xdb1adb5f08b972;
Key 2 0x5452032773c286;
Key 3 0x00000000000000;
Key 4 0x8774eb3ebb4f84;
Key 5 0x00000000000000;
Keyseq 0 F;
Keyseq 1 M;
Keyseq 2 L;
Keyseq 3 F;
Keyseq 4 M;
Keyseq 5 L;
Key StartCBC 0x503f2f655b1b2f82;
StartKey 0;
```

In the case of the string for Key2, if the keyvalue is a character string, BitGen encodes the string into a 56-bit hex string. The same character string gives the same 56-bit hex string every time. This enables passwords or phrases to be used instead of hex strings.

The above keys are all specified as 64 bits each. The first 8 bits are used by Xilinx as header information and the following 56 bits as the key. BitGen accepts 64 bit keys, but automatically overrides the header, if necessary.

Because of security issues, the ?g Compress option cannot be used with bitstream encryption, and the Bitgen security level must be set for "level1" or "level2" (**-g security:level1** or **-g security:level2**) to prevent readback. Setting the security level to "none" enables readback, allowing the decrypted bitstream to be read back from a configured device.. Also, partial reconfiguration is not allowed.

In order to guarantee security of a bitstream, the Constant Block Chaining (CBC) initial value should be different for each design. The actual value of the CBC is not important, only that it be a different value for every different bitstream that is encrypted with the same key or every re-encryption of the same bitstream with a different key.

Loading Keys

DES keys can only be loaded through JTAG. The JTAG Programmer and iMPACT™ tools have the capability to take a .nky file and program the device with the keys. In order to program the keys, a "key-access mode" is entered. When this mode is entered, all of the FPGA memory, including the keys and configuration data, is cleared. Once the keys are programmed, they cannot be reprogrammed without clearing the entire device. This "key access mode" is completely transparent to most users.

Keys are programmed using the ISC_PROGRAM instruction, as detailed in the JTAG 1532 specification. SVF generation is also supported, if keys are to be programmed using a different method, such as a microprocessor or JTAG test software.

Loading Encrypted Bitstreams

Once the device has been programmed with the correct keys, the device can be configured with an encrypted bitstream. Non-encrypted bitstreams may also be used to configure the device, and the stored keys are ignored. The method of configuration is not at all affected by encryption. Any of the modes may be used, and the signaling does not change (refer to Chapter 4: Configuration). However, *all* bitstreams must configure the entire device, since partial reconfiguration is not permitted.

Once the device has been configured with an encrypted bitstream, it cannot be reconfigured without toggling the PROG pin, cycling power, or performing the JTAG JPROG_B instruction. All of these events fully clear the configuration memory, but none of these events reset the keys as long as V_{BATT} or V_{CCAUX} are maintained.

V_{BATT}

V_{BATT} is a separate battery voltage to allow the keys to remain programmed in the Virtex-II device. V_{BATT} draws very little current (on the order of nA) to keep the keys programmed. A small watch battery is suitable (refer to V_{BATT} DC Characteristics in the **Virtex-II Data Sheet (DS031)** and the battery's specifications to estimate its lifetime).

While the auxiliary voltage (V_{CCAUX}) is applied, V_{BATT} does not draw any current, and the battery can be removed or exchanged.

Temperature-Sensing Diode (DXP/DXN)

The Virtex-II Pro FPGA *temperature-sensing diode* can be accessed through the DXP (Anode) and DXN (Cathode) pins. The DXP and DXN pins are wired internally to a diode-connected transistor, which creates a remote temperature sensor.

NOTE: These additional pins are not available on all packages.

DXP and DXN are dedicated pins attached to the substrate/die itself. They cannot be accessed through the software tools. They are always available, and no special actions in the design need be taken in order to use them. If the temperature-sensing feature is not used, the DXP and DXN pins can be left unconnected.

The temperature-sensing diode is one part of a two-part system. To complete the system, a *temperature sensor interface device* is also required. Most temperature sensor interface devices provide corresponding pins that can connect directly to the DXP and DXN pins on the Virtex-II Pro FPGA. Once the upper and lower temperature limits have been be set up and the device is operated in conjunction with the sensing diode in the FPGA, the temperature sensor interface device creates an output signal when these bounds are exceeded.

This output can be used as an interrupt to turn off the clock, turn on a fan, or perform some other operation intended to reduce heat.

The accuracy of the temperature measurement achieved by this two-part system does not depend on the temperature-sensing diode itself (DXN/DXP pins), because the voltage-versus-temperature curve is determined by the physical nature of the diode. Numerical readout accuracy relies instead upon the temperature sensor interface device, which translates the IV-versus- temperature curves into an actual temperature reading. The data sheets of the temperature sensors listed below contain the accuracy specifications.

Examples of Temperature Sensors

1. **Maxim Remote/Local Temperature Sensors.** For general information on these devices, go to the Maxim website at **http://www.maxim-ic.com**.

   Specifically, the data sheets for these devices can be found at:

   - **http://pdfserv.maxim-ic.com/ds/en/MAX1617.pdf**
   - **http://www.maxim-ic.com/quick_view2.cfm/qv_pk/3000**
   - **http://pdfserv.maxim-ic.com/en/ds/MAX6627-MAX6628.pdf**

   Refer to the "PC Board Layout" section of these data sheets for important design considerations.

2. **National Semiconductor LM83 or LM86 Triple-Diode Input and Local Digital Temperature Sensors with Two-Wire Interface.** For general information on these devices, go to the National Semiconductor website at http://www.national.com.

   Specifically, the data sheets for these devices can be found at:

   - **http://www.national.com/ds/LM/LM83.pdf**
   - **http://www.national.com/ds/LM/LM86.pdf**

   Refer to the "Application Hints" section of these data sheets for important design considerations.

Using the CORE Generator System

Introduction

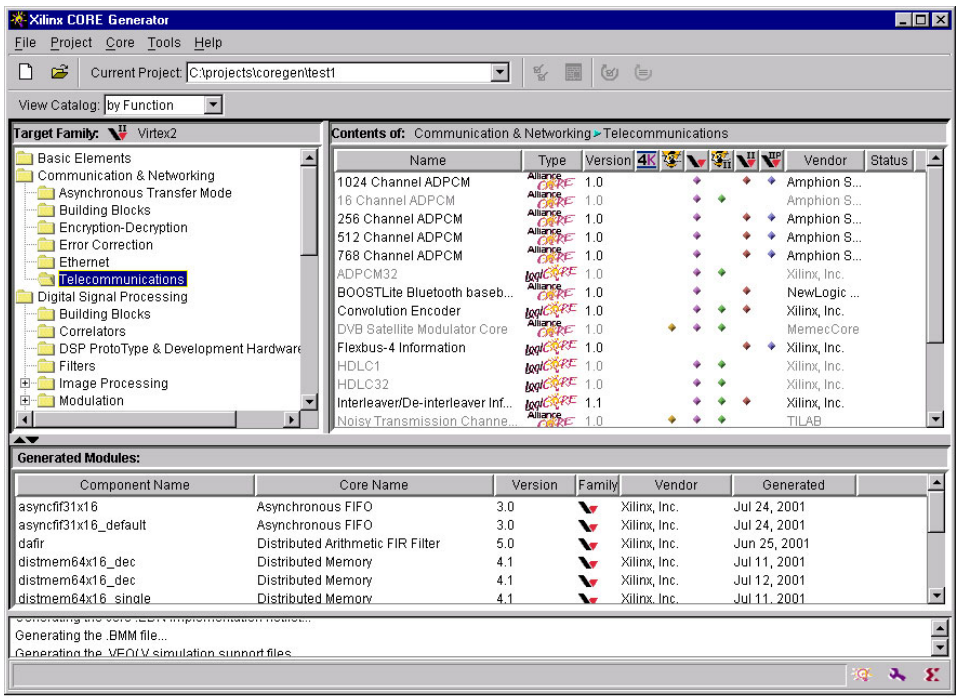
This section on the Xilinx CORE Generator System™ and the Xilinx Intellectual Property (IP) Core offerings is provided as an overview of products that facilitate the Virtex-II design process. For more detailed and complete information, consult the *CORE Generator Guide*, which can be accessed online in the Xilinx software installation, as well as at the http://toolbox.xilinx.com/docsan/xilinx4/manuals.htm site, under the "Design Entry Tools" heading.

The CORE Generator System

The Xilinx CORE Generator System is the cataloging, customization, and delivery vehicle for IP cores targeted to Xilinx FPGAs. This tool is included with all Xilinx ISE BaseX, ISE Foundation, and ISE Alliance Series software packages. The CORE Generator provides centralized access to a catalog of ready-made IP functions ranging in complexity from simple arithmetic operators, such as adders, accumulators, and multipliers, to system-level building blocks, such as filters, transforms, and memories. Cores can be displayed alphabetically, by function, by vendor, or by type. Each core comes with its own data sheet, which documents the core's functionality in detail.

The CORE Generator User Interface (see Figure 3-130) has direct links to key Xilinx web support pages, such as the Xilinx IP Center website (www.xilinx.com/ipcenter) and Xilinx Technical Support, making it very easy to access the latest Virtex-II IP releases and get helpful, up-to-date specifications and information on technical issues. Links to partner IP providers are also built into the informational GUIs for the various partner-supplied AllianceCORE products described under "AllianceCORE Program" on page 259.

The use of CORE Generator IP cores in Virtex-II designs enables designers to shorten design time, and it also helps them realize high levels of performance and area efficiency without any special knowledge of the Virtex-II architecture. The IP cores achieve these high levels of performance and logic density by using Xilinx Smart-IP™ technology.



ug002_c2_068b_100901

Figure 3-130: **Core Generator User Interface**

Smart-IP Technology

Smart-IP technology leverages Xilinx FPGA architectural features, such as look-up tables (LUTs), distributed RAM, segmented routing and floorplanning information, as well as relative location constraints and expert logic mapping to optimize the performance of every core instance in a given Xilinx FPGA design. In the context of Virtex-II cores, Smart-IP technology includes the use of the special high-performance Virtex-II architectural features, such as embedded 18x18 multipliers, block memory, shift register look-up tables (SRL16's), and special wide mux elements.

Smart-IP technology delivers:

- Physical layouts optimized for high performance
- Predictable high performance and efficient resource utilization
- Reduced power requirements through compact design and interconnect minimization
- Performance independent of device size
- Ability to use multiple cores without deterioration of performance
- Reduced compile time over competing architectures

CORE Generator Design Flow

A block diagram of the CORE Generator design flow is shown in Figure 3-131.

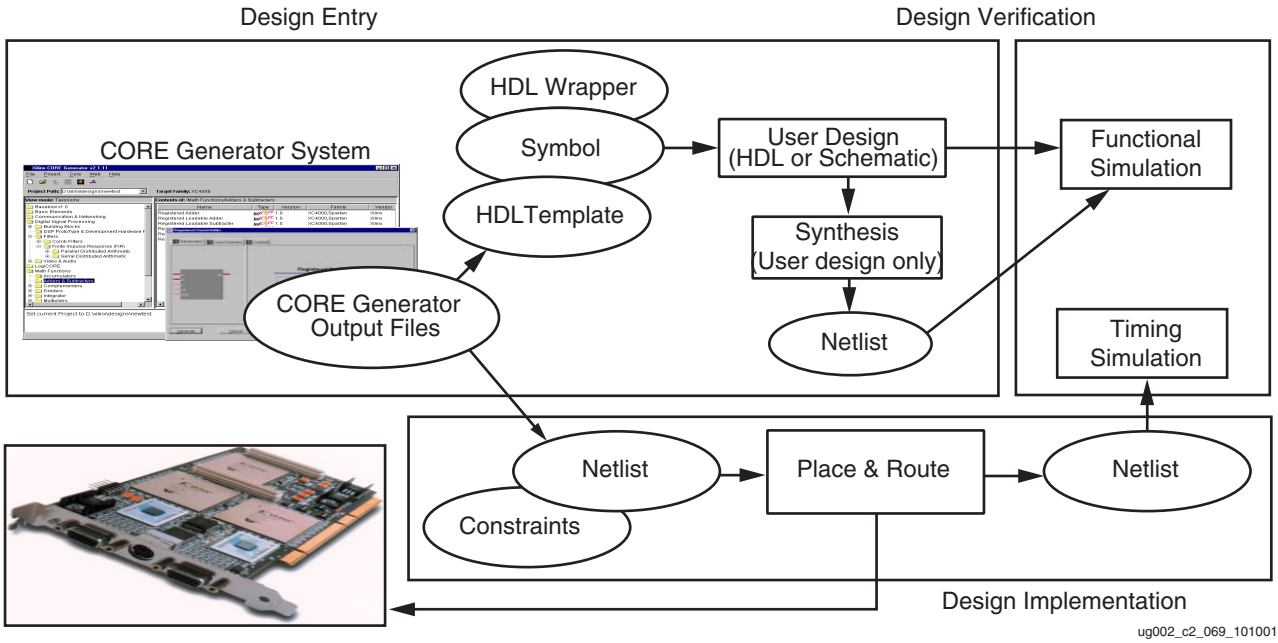


Figure 3-131: **CORE Generator Design Flow**

Note:

1. The outputs produced by the CORE Generator consist of an implementation Netlist and optional schematic symbol, HDL template files, and HDL simulation model wrapper files.

Core Types

Parameterized Cores

The CORE Generator System supplies a wide assortment of parameterized IP cores that can be customized to meet specific Virtex-II design needs and size constraints. See Figure 3-132. For each parameterized core, the CORE Generator System supplies:

- A customized EDIF implementation netlist (.EDN)
- A parameterized Verilog or VHDL behavioral simulation model (.V, .VHD) and corresponding wrapper file (also .V, .VHD)
- Verilog or VHDL templates (.VEO, .VHO)
- An ISE Foundation or Viewlogic® schematic symbol

The EDIF implementation netlist is used by the Xilinx tools to implement the core. The other design files generated depend on the Design Entry settings specified (target CAE vendor, and design flow type -- schematic or HDL). Schematic symbol files are generated when a schematic design flow is specified for the project.

Parameterized HDL simulation models are provided in two separate HDL simulation libraries, one for Verilog functional simulation support, and the other for VHDL functional simulation support. The libraries, which are included as part of the Xilinx installation, are in the following locations:

$XILINX/verilog/src/XilinxCoreLib

$XILINX/vhdl/src/XilinxCoreLib

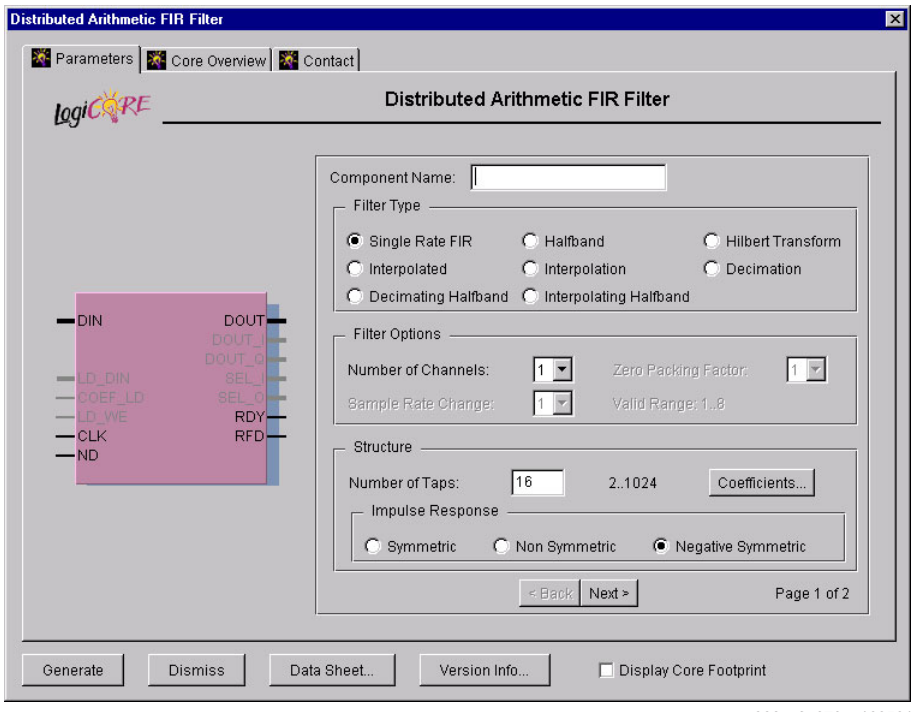


Figure 3-132: **Core Customization Window for a Parameterized Core**

If using a compiled simulator, these libraries must be precompiled before performing a functional simulation of the cores. An analyze_order file describing the required compile order of these models is included with each XilinxCoreLib library, one for Verilog (verilog_analyze_order) and one for VHDL (vhdl_analyze_order).

For an HDL design flow, Verilog and VHDL templates (.VEO and .VHO files) are also provided to facilitate the integration of the core into the design for the purposes of functional simulation, synthesis, and implementation. The Verilog (.V) and VHDL (.VHD) wrapper files are also generated. The wrapper files for a particular core are compiled like normal simulation models. They convey custom parameter values to the corresponding generic, parameterized behavioral model for that core in the XilinxCoreLib library. The custom parameter values are used to tailor the behavior of the customized core.

The following is a sample VHO template:

```
component adder8
    port (
    a: IN std_logic_VECTOR(7 downto 0);
    b: IN std_logic_VECTOR(7 downto 0);
    c: IN std_logic;
    ce: IN std_logic;
    ci: IN std_logic;
    clr: IN std_logic;
    s: OUT std_logic_VECTOR(8 downto 0));
end component;

-- Synplicity black box declaration
attribute black_box : boolean;
attribute black_box of test: component is true;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------

-- The following code must appear in the VHDL architecture
-- body. Substitute your own instance name and net names.

------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG
your_instance_name : adder8
    port map (
    a => a,
    b => b,
    c => c,
    ce => ce,
    ci => ci,
    clr => clr,
    s => s);

-- INST_TAG_END ------ End INSTANTIATION Template ------------
-- You must compile the wrapper file test.vhd when simulating
-- the core, test. When compiling the wrapper file, be sure to
-- reference the XilinxCoreLib VHDL simulation library. For detailed
-- instructions, please refer to the "Core Generator Guide".
```

Fixed Netlist Cores

The other type of Virtex-II core provided by the CORE Generator is the fixed netlist core. These are preset, non-parameterized designs that are shipped with the following:

- A fixed EDIF implementation netlist (as opposed to one that is customized on the fly)
- .VEO and .VHO templates
- Non-parameterized .V and .VHD behavioral simulation models
- Schematic symbol support

Examples include the fixed netlist Xilinx FFTs and most AllianceCORE products.

Since the HDL behavioral models for fixed netlist cores are not parameterized, the corresponding .VEO and .VHO template files are correspondingly simple. They do not need to pass customizing parameter values to a library behavioral model.

Xilinx IP Solutions and the IP Center

The CORE Generator works in conjunction with the Xilinx IP Center on the world wide web to provide the latest IP and software upgrades. To make the most of this resource, Xilinx highly recommends that whenever starting a design, first do a quick search of the Xilinx IP Center (**www.xilinx.com/ipcenter**) to see whether a ready-made core solution is already available.

A complete catalog of Xilinx cores and IP tools resides on the IP Center, including:

- LogiCORE Products
- AllianceCORE Products
- Reference Designs
- XPERTS Partner Consultants
- Design Reuse Tools

When installing the CORE Generator software, the designer gains immediate access to dozens of cores supplied by the LogiCORE Program. In addition, data sheets are available for all AllianceCORE products, and additional, separately licensed, advanced function LogiCORE products are also available. New and updated Virtex-II IP for the CORE Generator can be downloaded from the IP Center and added to the CORE Generator catalog.

LogiCORE Program

LogiCORE products are designed, sold, licensed, and supported by Xilinx. LogiCORE products include a wide selection of generic, parameterized functions, such as muxes, adders, multipliers, and memory cores which are bundled with the Xilinx CORE Generator software at no additional cost to licensed software customers. System-level cores, such as PCI, Reed-Solomon, ADPCM, HDLC, POS-PHY, and Color Space Converters are also available as optional, separately licensed products. Probably, the most common application of the CORE Generator is to use it to quickly generate Virtex-II block and distributed memories. A more detailed listing of available Virtex-II LogiCORE products is available in Table 3-64 and on the Xilinx IP Center website (**www.xilinx.com/ipcenter**).

Types of IP currently offered by the Xilinx LogiCORE program include:

- Basic Elements: logic gates, registers, multiplexers, adders, multipliers
- Communications and Networking: ADPCM modules, HDLC controllers, ATM building blocks, forward error correction modules, and POS-PHY Interfaces
- DSP and Video Image Processing: cores ranging from small building blocks (e.g., Time Skew Buffers) to larger system-level functions (e.g., FIR Filters and FFTs)
- System Logic: accumulators, adders, subtracters, complementers, multipliers, integrators, pipelined delay elements, single and dual-port distributed and block RAM, ROM, and synchronous and asynchronous FIFOs
- Standard Bus Interfaces: PCI 64/66 (64-bit, 66 MHz), 64/33 (64-bit, 33 MHz), and 32/33 (32-bit, 3 3MHz) Interfaces

AllianceCORE Program

The AllianceCORE program is a cooperative effort between Xilinx and third-party IP developers to provide additional system-level IP cores optimized for Xilinx FPGAs. To ensure a high level of quality, AllianceCORE products are implemented and verified in a Xilinx device as part of the certification process.

Xilinx develops relationships with AllianceCORE partners who can complement the Xilinx LogiCORE product offering. Where Xilinx does not offer a LogiCORE for a particular function, Xilinx partners with an AllianceCORE partner to offer that function. A large percentage of Xilinx AllianceCORE partners focus on data and telecommunication applications, as well as processor and processor peripheral designs.

Together, Xilinx and the AllianceCORE partners are able to provide an extensive library of cores to accelerate the design process. AllianceCORE products include customizable cores which can be configured to exact needs, as well as fixed netlist cores targeted toward specific applications. In many cases, partners can provide cores customized to meet the specific design needs if the primary offerings do not fit the requirements. Additionally, source code versions of the cores are often available from the partners at additional cost for those who need maximum flexibility.

The library of Xilinx and AllianceCORE IP cores allows designers to leverage the expertise of experienced designers who are well-versed in optimizing designs for Virtex-II and other Xilinx architectures. This enables designers to obtain high performance and density in the target Virtex-II device with a faster time to market.

Reference Designs

Xilinx offers two types of reference designs; application notes (XAPPs) developed by Xilinx, and reference designs developed through the Xilinx Reference Design Alliance Program. Both types are extremely valuable to customers looking for guidance when designing systems. Reference designs can often be used as starting points for implementing a broad spectrum of functions in Xilinx programmable logic.

Application notes developed by Xilinx usually include supporting design files. They are supplied free of charge, without technical support or warranty. To see currently available reference designs, visit the Xilinx wwebsite.

Reference designs developed through the Xilinx Reference Design Alliance Program are developed, owned, and controlled by the partners in the program. The goal of the program is to form strategic engineering and marketing partnerships with other semiconductor manufacturers and design houses so as to assist in the development of high quality, 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. Visit the Xilinx website to see a list of designs currently available through this program.

XPERTS Program

Xilinx established the XPERTS Program to provide customers with access to a worldwide network of certified design consultants proficient with Xilinx Platform FPGAs, software, and IP core integration. All XPERT members are certified and have extensive expertise and experience with Xilinx technology in various vertical applications, such as communications and networking, DSP, video and image processing, system I/O interfaces, and home networking.

XPERTS partners are an integral part of Xilinx strategy to provide customers with cost-efficient design solutions, while accelerating time to market. For more information on Xilinx XPERTS Program, visit the Xilinx website.

Design Reuse Tools

To facilitate the archiving and sharing of IP created by different individuals and workgroups within a company, Xilinx offers the IP Capture Tool. The IP Capture Tool helps to package design modules created by individual engineers in a standardized format so that they can be cataloged and distributed using the Xilinx CORE Generator. A core can take the form of synthesizable VHDL or Verilog code, or a fixed function netlist. Once it is packaged by the IP Capture Tool and installed into the CORE Generator, the *"captured"* core can be shared with other designers within a company through an internal network. The IP Capture Tool is supplied as a separate utility through the Xilinx IP Center. For more information, visit the Xilinx website.

CORE Generator Summary

The CORE Generator delivers a complete catalog of IP including behavioral models, synthesis templates, and netlists with performance guaranteed by Xilinx Smart-IP technology. It is a repository for LogiCORE products from Xilinx, AllianceCORE products from Xilinx partners, and it supports Design Reuse for internally developed IP. In addition, LogiCORE products are continuously updated to add support for new Xilinx architectures, such as Virtex-II. The most current IP updates are available from the Xilinx IP Center.

Utilizing the CORE Generator library of parameterizable cores, designed by Xilinx for Xilinx FPGAs, the designer can enjoy the advantages of design reuse, including faster time to market and lower cost solutions. For more information, visit the Xilinx IP Center **www.xilinx.com/ipcenter** website.

Virtex-II IP Cores Support

Table 3-64 provides a partial listing of cores available for Virtex-II designs. For a complete catalog of Virtex-II IP, visit the Xilinx IP Center **www.xilinx.com/ipcenter** website.

Table 3-64: **Virtex-II IP Cores Support**

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
Basic Elements							
BUFE-based Multiplexer Slice	Xilinx	LogiCORE				1-256 bits wide	
BUFT-based Multiplexer Slice	Xilinx	LogiCORE				1-256 bits wide	
Binary Counter	Xilinx	LogiCORE				2-256 bits output width	
Binary Decoder	Xilinx	LogiCORE				2-256 bits output width	
Bit Bus Gate	Xilinx	LogiCORE				1-256 bits wide	
Bit Gate	Xilinx	LogiCORE				1-256 bits wide	
Bit Multiplexer	Xilinx	LogiCORE				1-256 bits wide	
Bus Gate	Xilinx	LogiCORE				1-256 bits wide	
Bus Multiplexer	Xilinx	LogiCORE				IO widths up to 256 bits	
Comparator	Xilinx	LogiCORE				1-256 bits wide	
FD-based Parallel Register	Xilinx	LogiCORE				1-256 bits wide	
FD-based Shift Register	Xilinx	LogiCORE				1-64 bits wide	
LD-based Parallel Latch	Xilinx	LogiCORE				1-256 bits wide	
RAM-based Shift Register	Xilinx	LogiCORE				1-256 bits wide, 1024 words deep	
Communication & Networking							
3G FEC Package	Xilinx	LogiCORE				Viterbi Decoder, Turbo Codec, Convolutional Enc	3G Wireless Infrastructure
3GPP Compliant Turbo Convolutional Decoder	Xilinx	LogiCORE	80%	40	XC2V500	3GPP specs, 2 Mbps, BER=10-6 for 1.5dB SNR	3G Wireless Infrastructure
3GPP Compliant Turbo Convolutional Encoder	Xilinx	LogiCORE	65%	60	XC2V250	Compliant w/ 3GPP, puncturing	3G Wireless Infrastructure
3GPP Turbo Decoder	SysOnChip	AllianceCORE	87%	66	XC2V500-5	3GPP/UMTS compliant, IMT-2000, 2Mbps data	Error correction, wireless

Table 3-64: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
8b/10b Decoder	Xilinx	LogiCORE	1 BRAM	100	XC2V1000	Industry std 8b/10b en/decode for serial data transmission	Physical layer of Fiber Channel
8b/10b Encoder	Xilinx	LogiCORE	1 BRAM	100	XC2V1000	Industry std 8b/10b en/decode for serial data transmission	Physical layer of Fiber Channel
ADPCM 1024 Channel	Amphion	AllianceCORE				G.721, 723, 726, 726a, 727, 727a, u-law, a-law	DECT, VOIP, cordless telephony
ADPCM 256 Channel	Amphion	AllianceCORE				G.721, 723, 726, 726a, 727, 727a, u-law, a-law	DECT, VOIP, cordless telephony
ADPCM 512 Channel	Amphion	AllianceCORE					
ADPCM 768 Channel	Amphion	AllianceCORE	89%	50	XC2V500-5	G.721, 723, 726, 726a, 727, 727a, u-law, a-law	DECT, VOIP, cordless telephony
ADPCM Speech Codec, 32 Channel (DO-DI-ADPCM32)	Xilinx	LogiCORE	62%	25	XC2V500	G.726, G.727, 32 duplex channels	DECT, VOIP, Wireless local loop, DSLAM, PBX
ADPCM Speech Codec, 64 Channel (DO-DI-ADPCM64)	Xilinx	LogiCORE	61%	27	XC2V500	G.726, G.727, 64 duplex channels	DECT, VOIP, wireless local loop, DSLAM, PBX
BOOST LITE Bluetooth Baseband Processor	NewLogic	AllianceCORE	73%	33%	XC2V1000-4	Compliant to Bluetooth v1.1, BQB qualified software for L2CAP, LHP, HC1, voice support	Bluetooth applications
BOOST Lite Bluetooth Baseband Processor	NewLogic	AllianceCORE	73%	33%	XC2V1000-4	Compliant to Bluetooth v1.1, BQB qualified software for L2CAP, LHP, HC1, voice support	Bluetooth applications
Convolutional Encoder	Xilinx	LogiCORE	10%	26	XC2V40	k from 3 to 9, puncturing from 2/3 to 12/13	3G base stations, broadcast, wireless LAN, cable modem, xDSL, satellite com, uwave
DVB-RCS Turbo Decoder	iCODING	AllianceCORE	54%	69	XC2V2000-5	DVB-RCS compliant, 9Mbps, data rate, switchable code rates and frame sizes	Error correction, wireless, DVB, Satellite data link
Flexbus 4 Interface Core, 16-Channel (DO-DI-FLX4C16)	Xilinx	LogiCORE	31%	200	XC2V3000 FG676-5		Line card: terabit routers & optical switches
Flexbus 4 Interface Core, 4-Channel (DO-DI-FLX4C4)	Xilinx	LogiCORE	27%	200	XC2V1000 FG456-5		Line card: terabit routers & optical switches
Flexbus 4 Interface Core, 1-Channel (DO-DI-FLX4C1)	Xilinx	LogiCORE	12%	200	XC2V1000 FG456-5		Line card: terabit routers & optical switches
HDLC Controller Core, 32 Channels	Xilinx	LogiCORE	34%	81	XC2V250	32 full duplex, CRC-16/32, 8/16-bit address insertion/deletion	X.25, POS, cable modems, frame relay switches, video confer. over ISDN
HDLC Controller Core, Single Channel	Xilinx	LogiCORE	15%	115	XC2V250	16/32-bit frame seq, 8/16-bit addr insert/delete, flag/zerop insert/detect	X.25, POS, cable modems, frame relay switches, video conf. over ISDN
Interleaver/De-interleaver	Xilinx	LogiCORE	30%	187	XC2V40	Convolutional, width up to 256 bits, 256 branches	Broadcast, wireless LAN, cable modem, xDSL, sat com, uwave nets, digital TV

Table 3-64: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
PE-MACMII Dual Speed 10/100 Mbps Ethernet MAC	Alcatel	AllianceCORE	33%	60	XC2V500-4	802.3 compliant, Supports single & multimode fiber optic devices, M11 interfaces, RMON and Etherstate statistics	Networking, broadband, NIC, SOHO, home networking, storage, routers, switches, printers,
POS-PHY Level 3 Link Layer Interface Core, 48 Channel (DO-DI-POSL3LINK48A)	Xilinx	LogiCORE	33%	104	XC2V6000 FF1152-4		
POS-PHY L3 Link Layer Interface, 16-Ch (DO-DI-POSL3LINK16)	Xilinx	LogiCORE	40%	104	XC2V1000 FG456-4		Line card: terabit routers & optical switches
POS-PHY L3 Link Layer Interface, 4-Ch (DO-DI-POSL3LINK4)	Xilinx	LogiCORE	15%	104	XC2V1000 FG456-4		Line card: terabit routers & optical switches
POS-PHY L3 Link Layer Interface, 2-Ch (DO-DI-POSL3LINK2)	Xilinx	LogiCORE	55%	104	XCV50E-8		Line card: terabit routers & optical switches
POS-PHY L3 Link Layer Interface, Single Channel	Xilinx	LogiCORE	6%	104	XC2V1000 FG456-4		
POS-PHY L4 Multi-Channel Interface (DO-DI-POSL4MC)	Xilinx	LogiCORE	29%	104	XC2V3000 FG676-5		
Reed-Solomon Decoder	Xilinx	LogiCORE	40%	98	XC2V250	Std or custom coding, 3-12 bit symbol width, up to 4095 symbols	Broadcast, wireless LAN, digital TV, cable modem, xDSL, sat com, uwave nets
Reed-Solomon Decoder	TILAB	AllianceCORE	56%	61	XC2V1000-5	parameterizable, RTL available	Error correction, wireless, DSL
Reed-Solomon Encoder	Xilinx	LogiCORE	42%	180	XC2V40	Std or cust coding, 3-12 bit width, up to 4095 symbols with 256 check symb.	Broadcast, wireless LAN, digital TV, cable modem, xDSL, satellite com,uwave nets
SDLC Controller	CAST	AllianceCORE	38%	158	XC2V100-5	Like Intel 8XC152 Global Serial Channel, Serial Comm., HDLC apps, telecom	embedded systems, professional audio, video
SPEEDROUTER Network Processor	IP	AllianceCORE	64%	80 MHz, 2.5 Gbps	XC2V1500-5	Solution requires SPEEDAnalyzer ASIC, 2.5 Gbps fdx wire speed; net processor (NPV)	Networking, edge and access, switches and routers
Turbo Decoder - 3GPP	SysOnChip	AllianceCORE	88%	65	XC2V2000-5	3GPP/UMTS compliant, 2Mbps data rate	Error correction, wireless
Turbo Encoder	TILAB	AllianceCORE	48%	120	XC2V80-5	3GPP/UMTS compliant, upto 4 interleaver laws	Error correction, wireless
TURBO_DEC Turbo Decoder	TILAB	AllianceCORE	99%	65	XC2V2000-5	3GPP/UMTS compliant, >2Mbps data rate	Error correction, wireless
Viterbi Decoder	Xilinx	LogiCORE	80%	100	XC2V250	Puncturing, serial & parallel architecture,	3G base stations, broadcast, wireless LAN, cable modem, xDSL, satellite com, uwave

Table 3-64: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
Viterbi Decoder, IEEE 802-compatible	Xilinx	LogiCORE	70%	147	XC2V250	Constraint length(k)=7, G0=171, G1=133	L/MMDS, cable modem, broadcast equip, wireless LAN, xDSL, sat com, uwave nets
Digital Signal Processing							
1024-Point Complex FFT IFFT for Virtex-II	Xilinx	LogiCORE	62%	41us, 100 MHz	XC2V500	16 bit complex data, 2's comp, forward and inverse transform	
16-Point Complex FFT IFFT for Virtex-II	Xilinx	LogiCORE	37%	123ns, 130 MHz	XC2V500	16 bit complex data, 2's comp, forward and inverse transform	
256-Point Complex FFT IFFT for Virtex-II	Xilinx	LogiCORE	54%	7.7us, 100 MHz	XC2V500	16 bit complex data, 2's comp, forward and inverse transform	
32 Point Complex FFT/IFFT	Xilinx	LogiCORE					
64-Point Complex FFT IFFT for Virtex-II	Xilinx	LogiCORE	38%	1.9us, 100 MHz	XC2V500	16 bit complex data, 2's comp, forward and inverse transform	
Bit Correlator	Xilinx	LogiCORE				4096 taps, serial/parallel input, 4096 bits width	
Cascaded Integrator Comb (CIC)	Xilinx	LogiCORE				32 bits data width, rate change from 8 to 16384	
Direct Digital Synthesizer	Xilinx	LogiCORE				8-65K samples, 32-bits output precision, phase dithering/offset	
Distributed Arithmetic FIR Filter	Xilinx	LogiCORE				32-bit input/coeff width, 1024 taps, 1-8 chan, polyphase, online coeff reload	
GVA-300 Virtex-II DSP Hardware Accelerator	GV	AllianceCORE	NA	NA		2 Virtex-II, Spartan-II FPGAs, 1 CPLD, Matlab I/F	DSP prototyping
LFSR, Linear Feedback Shift Register	Xilinx	LogiCORE				168 input widths, SRL16/register implementation	
Math Functions							
Accumulator	Xilinx	LogiCORE				1-256s bit wide	
Adder Subtracter	Xilinx	LogiCORE				1-256s bit wide	
DFP2INT Floating Point to Integer Converter	Digital	AllianceCORE	39%	66	XC2V250-5	Full IEEE-754 compliance, 4 pipelines, Single precision real format support	DSP, Math, Arithmetic apps
DFPADD Floating Point Adder	Digital	AllianceCORE	39%	66	XC2V250-5	Full IEEE-754 compliance, 4 pipelines, Single precision real format support	DSP, Math, Arithmetic apps
DFPCOMP Floating Point Comparator	Digital	AllianceCORE	16%	91	XC2V80-5	Full IEEE-754 compliance, 4 pipelines, Single precision real format support	DSP, Math, Arithmetic apps.
DFPDIV Floating Point Divider	Digital	AllianceCORE	99%	53	XC2V250-5	Full IEEE-754 compliance, 15 pipelines, Single precision real format support	DSP, Math, Arithmetic apps

Table 3-64: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
DFPMUL Floating Point Multiplier	Digital	AllianceCORE	44%	74	XC2V250-5	Full IEEE-754 compliance, 7 pipelines,32x32 mult, Single precision real format support	DSP, Math, Arithmetic apps.
DFPSQRT Floating Point Square Root	Digital	AllianceCORE	39%	66	XC2V250-5	Full IEEE-754 compliance, 4 pipelines, Single precision real format support	DSP, Math, Arithmetic apps
DINT2FP Integer to Floating Point Converter	Digital	AllianceCORE	37%	73	XC2V250-5	Full IEEE-754 compliance, double word input, 2 pipelines, Single precision real output	DSP, Math, Arithmetic apps
Multiply Accumulator (MAC)	Xilinx	LogiCORE				Input width up to 32 bits, 65-bit accumulator, truncation rounding	
Multiply Generator	Xilinx	LogiCORE				64-bit input data width, constant, reloadable or variable inputs, parallel/sequential implementation	
Pipelined Divider	Xilinx	LogiCORE				32-bit input data width, multiple clock per output	
Sine Cosine Look Up Table	Xilinx	LogiCORE				3-10 bit in, 4-32 bit out, distributed/block ROM	
Twos Complementer	Xilinx	LogiCORE				Input width up to 256 bits	
Memories & Storage Elements							
Asynchronous FIFO	Xilinx	LogiCORE				1-256 bits, 15-65535 words, DRAM or BRAM, independent I/O clock domains	
Content Addressable Memory (CAM)	Xilinx	LogiCORE				1-512 bits, 2-10K words, SRL16	
Distributed Memory	Xilinx	LogiCORE				1-1024 bit, 16-65536 word, RAM/ROM/SRL16, opt output regs and pipelining	
Dual-Port Block Memory	Xilinx	LogiCORE				1-256 bits, 2-13K words	
Single-Port Block Memory	Xilinx	LogiCORE				1-256 bits, 2-128K words	
Synchronous FIFO	Xilinx	LogiCORE				1-256 bits, 16-256 words, distributed/block RAM	
Microprocessors, Controllers & Peripherals							
10/100 Ethernet MAC	Xilinx	LogiCORE				Interfaces through OPB to MicroBlaze	Networking, comm., processor applications
AX1610 16-bit RISC Processor	Loarant	AllianceCORE	12%	91	XC2V500-5	44 opcode, 64-K word data, program, Harvard arch.	Control functions, state mach, coprocessor
C165X MicroController	CAST	AllianceCORE	60%	134	XC2V80-5	Microchip 16C5X PIC like	Embedded systems, telecom
C68000 Microprocessor	CAST	AllianceCORE	90%	32	XC2V500-5	MC68000 Compatible	Embedded systems, pro audio, video
CPU FPGA (Virtex-II) MicroEngine Cards	NMI	AllianceCORE	NA	NA	NA	Hitachi SH-3 CPU	Embedded systems

Table 3-64: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
CZ80CPU Microprocessor	CAST	AllianceCORE	55%	72	XC2V500-5	Zilog Z80 compatible, 8-bit processor	Embedded systems, communications
DDR SDRAM Controller Core	Memec-Core	AllianceCORE	7%	133	XC2V1000-4	DDR SDRAM burst length support for 2,4,8 per access, supports data 16,32, 64, 72.	Digital video, embedded computing , networking
DFPIC125X Fast RISC MicroController	Digital	AllianceCORE	49%	126	XC2V80-5	PIC 12c4x like, 2X faster, 12-bit wide instruction set, 33 instructions	Embedded systems, telecom, audio and video
DFPIC1655X Fast RISC MicroController	Digital	AllianceCORE	79%	140	XC2V80-5	S/W compatible with PIC16C55X, 14-bit instruction set, 35 instructions	Embedded systems, telecom, audio and video
DFPIC165X Fast RISC MicroController	Digital	AllianceCORE	49%	126	XC2V80-5	PIC 12c4x like, 2X faster, 12-bit wide instruction set, 33 instructions	Embedded systems, telecom, audio and video
DI2CM I2C Bus Controller Master	Digital	AllianceCORE	58%	143	XC2V50-5	I2C-like, multi master, fast/std. modes	Embedded systems
DI2CM I2C Bus Controller Slave	Digital	AllianceCORE	28%	157	XC2V50-5	I2C-like, Slave	Embedded
DI2CSB I2C Bus Controller Slave Base	Digital	AllianceCORE	15%	187	XC2V50-5	I2C-like, Slave	Embedded Systems
DR8051 RISC MicroController	Digital	AllianceCORE	68%	73	XC2V250-5	80C31 instruction set, RISC architecture 6.7X faster than standard 8051	Embedded systems, telecom, video
DR8051BASE RISC MicroController	Digital	AllianceCORE	46%	80-90	XC2V250-5	80C31 instruction set, high speed multiplier, RISC architecture 6.7X faster than standard 8051	Embedded systems, telecom, video
DR8052EX RISC MicroController	Digital	AllianceCORE	99%	71	XC2V250-5	80C31 instruction set, high speed mult/div ,RISC 6.7X faster than standard 8051	Embedded systems, telecom, video
e8254 Programmable Interval Timer/Counter	einfochips	AllianceCORE	1%	175	XC2V1000-5	Three 8-bit parallel ports, 24 programmable IO lines, 8-bit bidi data bus	Processor, I/O interface
e8255 Peripheral Interface	einfochips	AllianceCORE	1%	175	XC2V1000-5	Three 8-bit parallel ports, 24 programmable IO lines, 8-bit bidi data bus	Processor, I/O interface
Flip805x-PS Microprocessor	Dolphin	AllianceCORE	39%	38	XC2V1000-5	Avg 8X faster & code compatible v. legacy 8051, verification bus monitor, SFR IF, DSP focused	DSP, Telecom, industrial, high speed control
IIC	Xilinx	LogiCORE				Interfaces through OPB to MicroBlaze	Networking, com, processor applic
LavaCORE Configurable Java Processor Core	Derivation	AllianceCORE	38%	20	XC2V1000-5	32b data/address optional DES	Internet appliance, industrial control
LavaCORE Configurable Java Processor Core	Derivation	AllianceCORE	38%	20	XC2V1000-5	32b data/address optional DES	Internet appliance, industrial control
Lightfoot 32-bit Java Processor Core	Digital	AllianceCORE	33%	40	XC2V1000-5	32bit data, 24 bit address, 3 Stage pipeline, Java/C dev. tools	Internet appliance, industrial control, HAVi multimedia, set top boxes
MicroBlaze Soft RISC Processor	Xilinx	LogiCORE		125		Soft RISC Processor, small footprint	Networking, communications
OPB Arbiter	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications

Table 3-64: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
OPB GPIO	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB Interrupt Controller	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB Memory Interface (Flash, SRAM)	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB Timer/Counter	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB UART (16450, 16550)	Xilinx	LogiCORE		125		Interfaces through OPB to MicroBlaze	Processor applications
OPB UART Lite	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
OPB WDT	Xilinx	LogiCORE		125		Bundled in the MicroBlaze Development Kit	Processor applications
PF3100 PC/104-Plus Reconfigurable Module	Derivation	AllianceCORE	N/A	N/A	XC2V1000 FG256	PC/104 & PC/104+ devlopment board	Internet appliance, industrial control
SPI	Xilinx	LogiCORE				Interfaces through OPB to MicroBlaze	Networking, communications, processor applications
XF-UART Asynchronous Communications Core	Memec-Core	AllianceCORE	15%	50	XCS20-4	UART and baud rate generator	Serial data communication
Standard Bus Interfaces							
PCI-X 64/100 Interface for Virtex-II (DO-DI-PCIX64-VE)	Xilinx	LogiCORE	30%	100	XC2V1000 FG456-5	PCI-X 1.0 comp, 64/32-bit, 66 MHz PCI-X initiator and target IF, PCI 2.2 comp, 64/32-bit, 33 MHz PCI initiator and target IF, 3.3 V PCI-X at 33-66 MHz, 3.3 V PCI at 0-33 MHz	Server, embedded, Gb ethernet, U320 SCSI, Fibre Ch, RAID cntl, graphics
PCI32 Virtex Interface Design Kit (DO-DI-PCI32-DKT)	Xilinx	LogiCORE	6%	66	XC2V1000 FG456-5	Includes PCI32 board, drive development kit, and customer education 3-day training class	
PCI32 Virtex Interface, IP Only (DO-DI-PCI32-IP)	Xilinx	LogiCORE	6%	66	XC2V1000 FG456-5	v2.2 comp, assured PCI timing, 3.3/5-V, 0-waitstate, CPCI hot swap friendly	PC add-in boards, CPCI, embedded
PCI64 & PCI32, IP Only (DO-DI-PCI-AL)	Xilinx	LogiCORE	6 - 7%	66	XC2V1000 FG456-5	v2.2 comp, assured PCI timing, 3.3/5-V, 0-waitstate, CPCI hot swap friendly	PC boards, CPCI, embedded, hiperf video, Gb ethernet
PCI64 Virtex Interface Design Kit (DO-DI-PCI64-DKT)	Xilinx	LogiCORE	7%	66	XC2V1000 FG456-5	v2.2 comp, assured PCI timing, 3.3/5-V, 0-waitstate, CPCI hot swap friendly	PC boards, CPCI, embedded, hiperf video, Gb ethernet
PCI64 Virtex Interface, IP Only (DO-DI-PCI64-IP)	Xilinx	LogiCORE	7%	66	XC2V1000 FG456-5	v2.2 comp, assured PCI timing, 3.3/5-V, 0-waitstate, CPCI hot swap friendly	PC boards, CPCI, embedded, hiperf video, Gb ethernet

Table 3-64: **Virtex-II IP Cores Support *(Continued)***

Function	Vendor Name	IP Type	Implementation Example			Key Features	Application Examples
			Occ	MHz	Device		
RapidIO 8-bit port LP-LVDS Phy Layer (DO-DI-RIO8-PHY)	Xilinx	LogiCORE	24%	250	XC2V1000 FG456-5	RapidIO Interconnect v1.1 compliant, verified with Motorola's RapidIO bus functional model v1.4	Routers, switches, backplane, control plane, data path, embedded sys, high-speed interface to memory and encryption engines, high-end video
USB 1.1 Device Controller	Memec-Core	AllianceCORE	21%	12	XC2V1000-5	Compliant with USB1.1 spec., Supports VCI bus, Performs CRC, Supports 1.5 Mbps & 12 Mbps	Scanners, Printers, Handhelds, Mass Storage
Video & Image Processing							
1-D Discrete Cosine Transform	Xilinx	LogiCORE				8-24 bits for coeff & input, 8-64 pts	
2-D DCT/IDCT Forward/Inverse Discrete Cosine Transform	Xilinx	LogiCORE					image, video phone, color laser printers
FASTJPEG_BW Decoder	BARCO-SILEX	AllianceCORE	67%	73	XC2V1000-4	Conforms to ISO/IEC Baseline 10918-1, Gray-Scale	Video editing, digital camera, scanners
FASTJPEG_C Decoder	BARCO-SILEX	AllianceCORE	78%	56	XC2V1000-4	Conforms to ISO/IEC Baseline 10918-1, color, multi-scan, Gray-Scale	Video editing, digital camera, scanners

XILINX®

Chapter 4

Configuration

Summary

This chapter covers the following topics:

- Introduction
- Configuration Solutions
- Software Support and Data Files
- Serial Programming Modes
- SelectMAP Programming Modes
- JTAG / Boundary Scan Programming Mode
- Configuration Details
- Readback

Introduction

Virtex-II devices are configured by loading application-specific configuration data into internal memory. Configuration is carried out using a subset of the device pins, some of which are dedicated, while others can be reused as general-purpose inputs and outputs after configuration is complete.

Depending on the system design, several configuration modes are selectable via mode pins. The mode pins M2, M1, and M0 are dedicated pins. An additional pin, HSWAP_EN, is used in conjunction with the mode pins to select whether user I/O pins have pull-up resistors during configuration. By default, HSWAP_EN is tied High (internal pull-up resistor), which shuts off pull-up resistors on the user I/O pins during configuration. When HSWAP_EN is tied Low, the pull-up resistors are on and therefore, the user I/Os have pull-up resistors during configuration.

Other dedicated pins are:

- CCLK — configuration clock pin
- DONE — configuration status pin
- TDI, TDO, TMS, TCK — boundary-scan pins
- PROG_B — configuration reset pin

Depending on the configuration mode selected, CCLK can be an output generated by the Virtex-II FPGA or an input accepting externally generated clock data. For correct operation, these pins require a V_{CCAUX} of 3.3V to permit low-voltage transistor-to-transistor logic (LVTTL) operations.

All dual-function configuration pins are contained in banks 4 and 5. Bank 4 contains pins used in serial configuration modes, and banks 4 and 5 contain pins used for SelectMAP modes.

A persist option is available, which can be used to force 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

for user I/O in normal operation. The persist option does not apply to boundary-scan related pins. The persist feature is valuable in applications that employ partial reconfiguration, dynamic reconfiguration, or readback.

The M2, M1, and M0 mode pins are intended to be set at a constant DC voltage level, either through pull-up or pull-down resistors, or tied directly to ground or V_{CCAUX}. The mode pins are not to be toggled (changed) during and after configuration.

Configuration Modes

Virtex-II supports the following configuration modes:

- Master-Serial
- Slave-Serial (default)
- Master SelectMAP
- Slave SelectMAP
- Boundary-Scan (IEEE 1532 and IEEE 1149)

Table 4-1 shows Virtex-II configuration mode pin settings.

Table 4-1: **Virtex-II Configuration Mode Pin Settings**

Configuration Mode(1)	M2	M1	M0	CCLK Direction	Data Width	Serial Dout(2)
Master Serial	0	0	0	Out	1	Yes
Slave Serial	1	1	1	In	1	Yes
Master SelectMAP	0	1	1	Out	8	No
Slave SelectMAP	1	1	0	In	8	No
Boundary Scan	1	0	1	N/A	1	No

Notes:

1. The HSWAP_EN pin controls the pull-ups. Setting M2, M1, and M0 selects the configuration mode, while the HSWAP_EN pin controls whether or not the pull-ups are used.
2. Daisy chaining is possible only in modes where Serial Dout is used. For example, in SelectMAP modes, the first device does NOT support daisy chaining of downstream devices.

Table 4-2 lists the total number of bits required to configure each device:

Table 4-2: **Virtex-II Bitstream Sizes**

Device	Bitstream Size		
	Regular Multiplier(1,3)	Enhanced Multiplier(1,3)	Enhanced Multiplier with `FreezeDCI:Yes` Setting(2,3)
XC2V40	360,096	338,976	340,832
XC2V80	635,296	598,816	601,952
XC2V250	1,697,184	1,593,632	1,598,048
XC2V500	2,761,888	2,560,544	2,566,240
XC2V1000	4,082,592	3,752,736	3,759,712
XC2V1500	5,659,296	5,170,208	5,178,464
XC2V2000	7,492,000	6,812,960	6,822,304
XC2V3000	10,494,368	9,594,656	9,605,472
XC2V4000	15,659,936	14,226,720	14,240,096
XC2V6000	21,849,504	19,759,904	19,775,840
XC2V8000	29,063,072	26,194,208	26,212,704

Notes:

1. A specific code must be used to order devices with enhanced multiplier. For device availability and ordering information, please contact your FAE or Xilinx sales office.
2. The bitgen **`-g FreezeDCI:Yes`** setting generates a larger bitstream, as shown in this column. This is due to additional configuration activity needed to disable the DCI oscillator.
3. Other bitgen settings that can impact bitstream size include:
   `-g DebugBitstream:Yes` (creates a larger bitstream)
   `-g compress:yes` (creates a potentially smaller bitstream)

Configuration Process and Flow

The configuration process involves loading the configuration bitstream into the FPGA using the selected mode. There are four major phases in the configuration process:

- Clearing Configuration Memory
- Initialization
- Loading Configuration Data
- Device Startup

Figure 4-1 illustrates the configuration process flow.

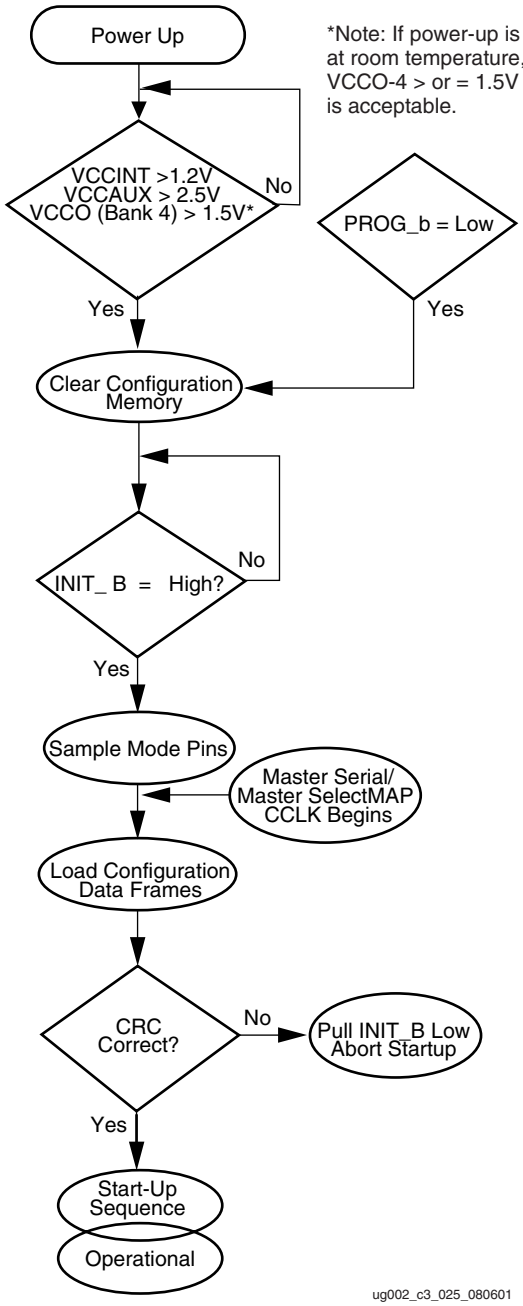


Figure 4-1: **Configuration Process**

Power Up

The V_{CCINT} power pins must be supplied with a 1.5V source. (Refer to the **Virtex-II Data Sheet (DS031)** for DC characteristics.) The IOB voltage input for Bank 4 (V_{CCO_4}) and the auxiliary voltage input (V_{CCAUX}) are also used as a logic input to the Power-On-Reset (POR) circuitry. Even if this bank is not being used, V_{CCO_4} must be connected to a 1.5V or greater source.

Clearing Configuration Memory

In the memory clear phase, non-configuration I/O pins are 3-stated with optional pull-up resistors. The INIT_B and DONE pins are driven Low by the FPGA, and the memory is cleared. After PROG_B transitions High, memory is cleared twice and initialization can begin.

The INIT_B pin transitions High when the clearing of configuration memory is complete. A logic Low on the PROG_B input resets the configuration logic and holds the FPGA in the clear configuration memory state. When PROG_B is released, the FPGA continues to hold INIT_B Low until it has completed clearing all of the configuration memory. The minimum Low pulse time for PROG_B is defined by the $T_{PROGRAM}$ timing parameter. There is no maximum value. The power-up timing of configuration signals is shown in Figure 4-2. For corresponding specifications, see the **Virtex-II Platform FPGA Data Sheet**.

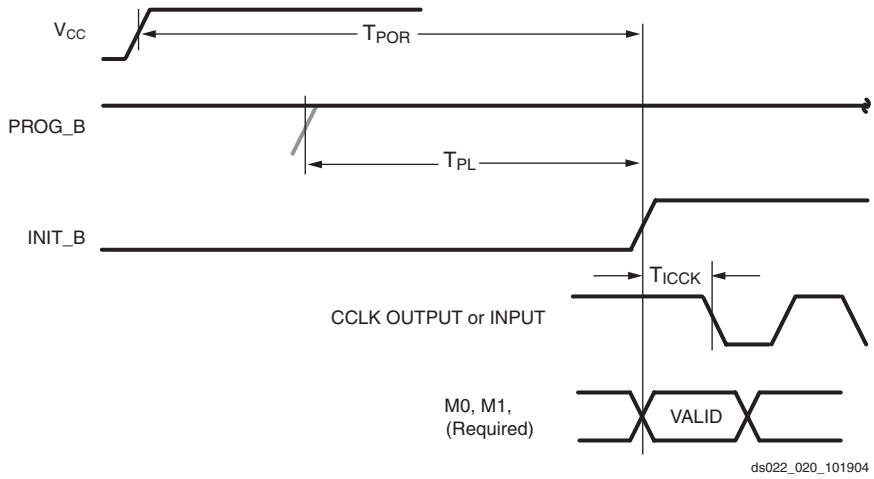


Figure 4-2: **Power-Up Timing Configuration Signals**

Initialization

For the initialization phase, the INIT_B pin is released, the mode pins are sampled, the appropriate pins become active, and the configuration process begins. It is possible to delay configuration by externally holding INIT_B Low.

Delaying Configuration

The INIT_B pin can also be held Low externally to delay configuration of the FPGA. The FPGA samples its mode pins on the rising edge of INIT_B. After INIT_B transitions to High, configuration can begin. No additional time-out or waiting periods are required, but configuration does not need to commence immediately after the transition of INIT_B. The configuration logic does not begin processing data until the synchronization word from the bitstream is loaded.

Loading Configuration Data

Cyclic Redundancy Checking (CRC) is automatically performed (“AutoCRC” — see page 320) at the end of any write to the Frame Data Input Register (FDRI), and again prior to startup. If the CRC checks do not fail, the device start-up phase can begin.

If the CRC check fails, the INIT_B pin is asserted (logic Low) to indicate that a CRC error has occurred. The startup sequence is aborted and configuration fails with DONE=0 and INIT=0.

To reconfigure the device, the PROG_B pin should be asserted to reset the configuration logic. Cycling power on the device also resets the FPGA for configuration. For more information on CRC calculation, refer to "CRC Register (CRC)" on page 319.

The details of loading configuration data in each of the five modes are discussed in the following sections:

- "Serial Programming Modes" on page 283
- "SelectMAP Programming Modes" on page 286
- "Slave Serial Mode" on page 284
- "Slave SelectMAP Mode" on page 289
- "JTAG / Boundary Scan Programming Mode" on page 298

After the last data frame is loaded, the global set/reset (GSR) signal is pulsed (see the "GRESTORE" description in Table 4-24 on page 322). The GSR signal initializes all registers in the device prior to entering the startup sequence, which is the next configuration phase.

Device Startup

Device startup brings the FPGA out of the configuration process and into normal device operation. The Startup Sequence is controlled by an eight-phase state machine that:

- Deasserts GTS, activating I/Os
- Asserts GWE, allowing RAMs and flip-flops to toggle
- Releases the DONE pin
- Asserts the End-Of-Startup signal (EOS) during the final phase. The EOS signal is an internal flag that can be read through the Status Register (see "Status Register (STAT)" on page 324).

Figure 4-3 depicts the default start-up sequence.

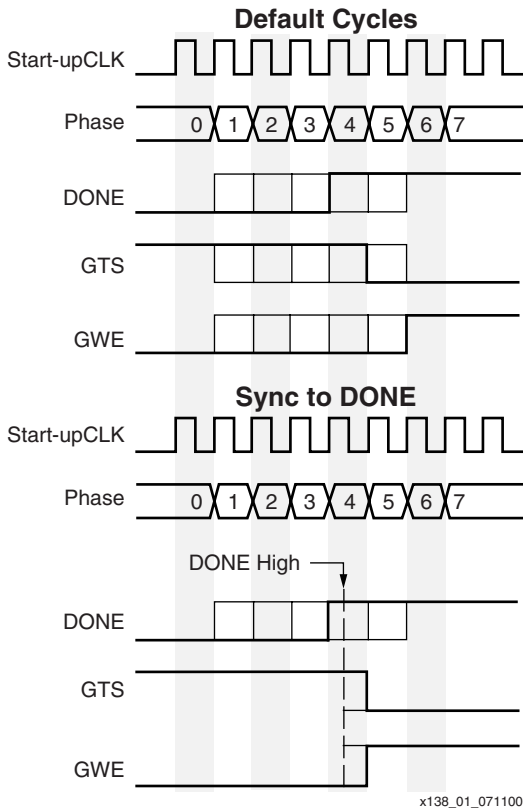


Figure 4-3: **Default Start-Up Sequence**

The order of the start-up events is user-programmable according to the BitGen startup options (see Appendix A: BitGen and PROMGen Switches and Options). Delays can be inserted in the start-up sequence to allow DCMs to lock and DCI to match.

The startup sequencer always waits for the DONE pin on the FPGA to go High during the Done cycle. The DONE pin does not necessarily go High when it is "released" by the device, since the pin may be held Low externally by other devices. The DONE pin can be configured as an open-drain output, in which case the DONE pin is "released" to a high-impedance state. Alternatively, it can be configured as an active driver, in which case the DONE pin is "released" to an active High level. The BitGen DRIVE DONE (bitgen **`-g drivedone:yes`**) setting determines whether the active driver is enabled on the DONE pin. This feature allows several FPGAs with connected DONE pins become active simultaneously.

The DONE pin may be held low for many CCLK cycles after being released by the device, but when the transition from logic 0 to logic 1 begins, it must finish within one CCLK cycle. If many devices are connected to the same DONE signal, the last device may not be able to

drive DONE High within one CCLK cycle. In this case, an external pull-up resistor may be needed on the DONE signal. If the rise time on the DONE signal cannot be reduced to one CCLK period, set the DONE PIPE option (bitgen **`-g donepipe:yes`**) to allow an additional CCLK cycle.

Configuration Pins

Dedicated pins (CCLK, PROG_B, DONE, M2, M1, and M0) powered by V_{CCAUX} are designated for configuration. Dual-function pins (D0/DIN, D1:D7, CS_B, RDWR_B, BUSY/DOUT and INIT_B) powered by V_{CCO} are designated for configuration and other user functions after configuration. Table 4-3 is the list of all dedicated, dual-function, and user pins.

Before and during configuration, all configuration I/O pins are set for LVTTL, 12mA, *fast* slew rate, except for the CCLK pin. It is set for LVTTL, 12mA, *slow* slew rate.

Table 4-3: **Virtex-II Pins and Voltages**

Pin Name	Direction	Supply Voltage (V)	Comments
CCLK	Input/Output	3.3	Pin is supplied by V_{CCAUX}.
PROG_B	Input	3.3	Pin is supplied by V_{CCAUX}.
DONE	Open-drain output	3.3	Pin is supplied by V_{CCAUX}.
M2, M1, M0	Input	3.3	Pins are supplied by V_{CCAUX}.
HSWAP_EN	Input	3.3	Pin is supplied by V_{CCAUX}.
TDI	Input	2.5/3.3	Pin is supplied by V_{CCAUX}.
TMS	Input	2.5/3.3	Pin is supplied by V_{CCAUX}.
TCK	Input	2.5/3.3	Pin is supplied by V_{CCAUX}.
TDO	Output	3.3	Pin is supplied by V_{CCAUX}.
PWRDWN_B	Input	3.3	Pin is supplied by V_{CCAUX}. It has an internal pull-up resistor.
D0/DIN, D1-D7	Input/Output	2.5/3.3	Pins are supplied by V_{CCO}.
CS_B	Input	2.5/3.3	Pin is supplied by V_{CCO}.
RDWR_B	Input	2.5/3.3	Pin is supplied by V_{CCO}.
BUSY/DOUT	Output	2.5/3.3	Pin is supplied by V_{CCO}.
INIT_B	Input/Output	2.5/3.3	Pin is supplied by V_{CCO}.
VRP	Input	N/A	Reference resistor pulled down to GND.
VRN	Input	N/A	Reference resistor pulled up to V_{CCO}.
ALT_VRP	Input	N/A	Reference resistor pulled down to GND.
ALT_VRN	Input	N/A	Reference resistor pulled up to V_{CCO}.
V_{REF}	Input	N/A	Depends on the I/O standard used.
DXN, DXP	N/A	N/A	These pins are the cathode and anode of the temperature diode.
V_{BATT}	Input	3.0	Battery supply for the encryption keys. Connect this pin to either V_{CCAUX} or GND when bitstream encryption is not used.
User I/O	Input/Output	1.5/1.8/ 2.5/3.3	Must be supported by a 3.3V signaling standard (LVTTL, LVCMOS, PCI33, PCI66, PCIX, LVDCI33)

Mixed Voltage Environments

Virtex-II devices have multiple power inputs:

- V_{CCINT} (1.5V) powers the internal circuitry.
- V_{CCAUX} (3.3V) powers critical resources in the FPGA.
- V_{CCO} (1.5V, 1.8V, 2.5V, or 3.3V) powers the IOB circuitry.

SelectI/O-Ultra is separated into eight banks of I/O groups. Each bank can be configured with one of several I/O standards. Refer to the "Design Considerations" chapter for I/O banking rules and available I/O standards. Before and during configuration, all I/O banks are set for the LVTTL standard, which requires an output voltage (V_{CCO}) of 3.3V for normal operation.

All dedicated configuration pins are powered by V_{CCAUX}. All dual-function configuration pins are located within banks 4 and 5. As described under Configuration Process and Flow, the V_{CCO_4} input voltage is used as a logic input to the power-on-reset (POR) circuitry if the V_{CCO_4} voltage level is monitored by the power-on-reset (POR) circuitry. If it drops below the minimum allowed voltage, the POR circuit will reset the entire device, clearing configuration memory and causing DONE to go Low.

Table 4-4 summarizes the configuration V_{CCO} requirements discussed in the following sections.

Table 4-4: **Configuration Modes and V_{CCO} Voltages**

Configuration Mode	Pins Used	V_{CCO_4} (1)	V_{CCO_5} (1)
JTAG	Dedicated pins	1.5V min (2)	not a concern
Serial	Dedicated pins plus DOUT, DIN, and INIT	3.3V	not a concern
SelectMAP	Dedicated pins plus dual-function pins	3.3V	3.3V

Notes:

1. If $V_{CCO\_4/5}$ = 2.5V, the configuration frequency might be as low as half of the typical frequency.
2. V_{CCO_4} is monitored by the POR circuit. See section "Power Up" on page 273.

JTAG Configuration Mode

For JTAG configuration mode, JTAG inputs are independent of V_{CCO} and work between 2.5V and 3.3V TTL levels (V_{IL} max = .8V, V_{IH} min = 2.0V). The JTAG input pins are 3.3V tolerant. The JTAG output (TDO) is an active driver.

Serial Configuration Mode

For serial configuration mode, V_{CCO_4} pins require a 3.3V supply for output configuration pins to operate normally. In serial mode, all of the configuration pins are in bank 4.

SelectMAP Configuration Mode

For SelectMAP configuration mode, V_{CCO_4} and V_{CCO_5} pins require a 3.3V supply for output configuration pins to operate normally. In SelectMAP mode, all of the configuration pins are in banks 4 and 5.

Special V_{CCO} Requirements during Configuration and Readback

Serial and SelectMAP configuration modes require 3.3V on the V_{CCO_4} and V_{CCO_5} inputs, per Table 4-4. If the I/Os in these banks are programmed for a voltage standard requiring some other voltage level, the V_{CCO} level will need to be switched. For example, if the I/Os in bank 4 are programmed for 1.5V HSTL and the device is configured through the Master Serial mode, V_{CCO_4} will need be held at 3.3V during configuration and switched to 1.5V after the DONE pin goes High.

Configuration Solutions

Several configuration solutions are available to support Virtex-II, each targeted to specific application requirements. Guidance and support (application notes, reference designs, and so forth) is also available for designers looking to develop and implement their own configuration solution for Virtex FPGAs.

System ACE CF(CompactFlash) Solution

The *System Advanced Configuration Environment* (System ACE™) solution combines a standard CompactFlash™ Association (CFA) Type I or Type II memory module (CompactFlash or 1" disk drive) with a Xilinx-designed ACE Controller™ configuration control chip. See Figure 4-4.

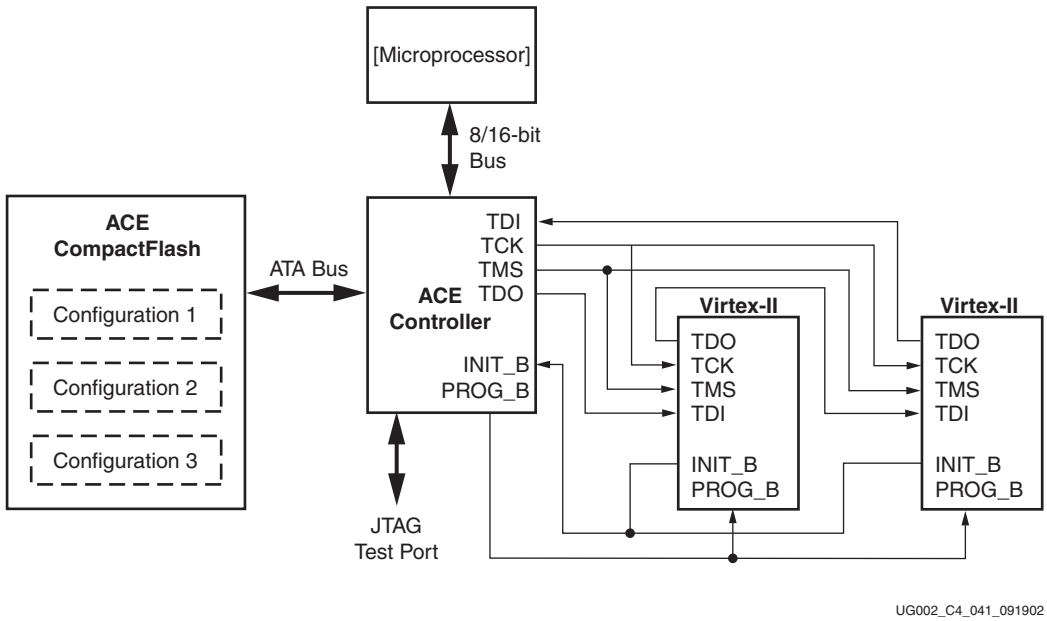


Figure 4-4: **System ACE CompactFlash and Controller**

The CompactFlash card stores an unlimited number of bitstreams, and supports densities up to 2 GB for FAT16 file formats. This card is capable of storing one large bitstream or several smaller bitstreams. If several bitstreams are used, the system can be set up so that individual bitstreams are callable as needed, allowing for dynamic reconfiguration of the Virtex-II device and other Xilinx FPGAs in the JTAG chain.

The ACE Controller drives bits through the FPGA JTAG chain and has three other ports:

- A port for interfacing with a microprocessor or a network
- A port for interfacing with the CompactFlash card
- A port that provides access to the FPGA JTAG chain for FPGA testing or configuration via automatic test equipment or via desktop or third-party programmers

For further information on any System ACE product, visit www.xilinx.com/systemace.

Configuration PROMs

Using Platform Flash PROMs

The Platform Flash family of in-system programmable (ISP) configuration PROMs provides the flexibility of reprogrammability and the efficiency of small-footprint packages, combined with both serial and SelectMAP (parallel) FPGA configurability. The family offers single devices ranging in density from 1 Mb to 32 Mb, and PROMs can be

cascaded to support even larger bitstreams. Data throughput rates of up to 264 Mb/s can be attained, and FPGA reconfiguration can be triggered via a JTAG command.

These PROMs are programmable using Xilinx iMPACT software via cable, HW-130, or standard third-party programmers.

Platform Flash PROMs are available in VO20, VO48, and FS48 packages. See Table 4-5 on page 281 to determine which PROMs go with which Virtex-II FPGAs.

Using XC17V00 PROMs

The XC17V00 family of one-time programmable (OTP) PROMs provides a proven, low-cost, compact, and pre-engineered configuration solution. Ranging from 1 Mb to 16 Mb, this family can also be daisy-chained to support larger bitstreams. This family supports serial configuration of Virtex-II FPGAs; in addition, the XC17V08 and XC17V16 support SelectMAP configuration modes.

The XC17V00 family can be used for stabilized designs that are in a high-volume production flow and/or for designs requiring a low-cost solution. XC17V00 PROMs can be programmed either by using the HW-130 or by using a variety of third-party programmers. The XC17V00 PROMs are available in VO8, SO20, PC20, VQ44, and PC44 packages. Data sheets for PROMs are available at **www.xilinx.com**. See Table 4-5 on page 281 to determine which PROMs go with which Virtex-II FPGAs.

Flash PROMs With a CPLD Configuration Controller

Some designers prefer to leverage existing Flash memory in their system to store the configuration bitstreams. A small CPLD-based configuration controller can provide the mechanism to access the bitstreams in the FLASH and deliver them quickly to Virtex-II devices. The following application notes describe the details for a serial or SelectMAP configuration architecture using FLASH memories and CPLDs:

- **XAPP079**: *Configuring* Xilinx *FPGAs Using an XC9500 CPLD and Parallel PROM* (available on **www.xilinx.com**) describes an architecture that configures a chain of Virtex-II devices using Master-Serial mode. See Figure 4-5 for an example of FPGA configuration using a CPLD and a parallel PROM.

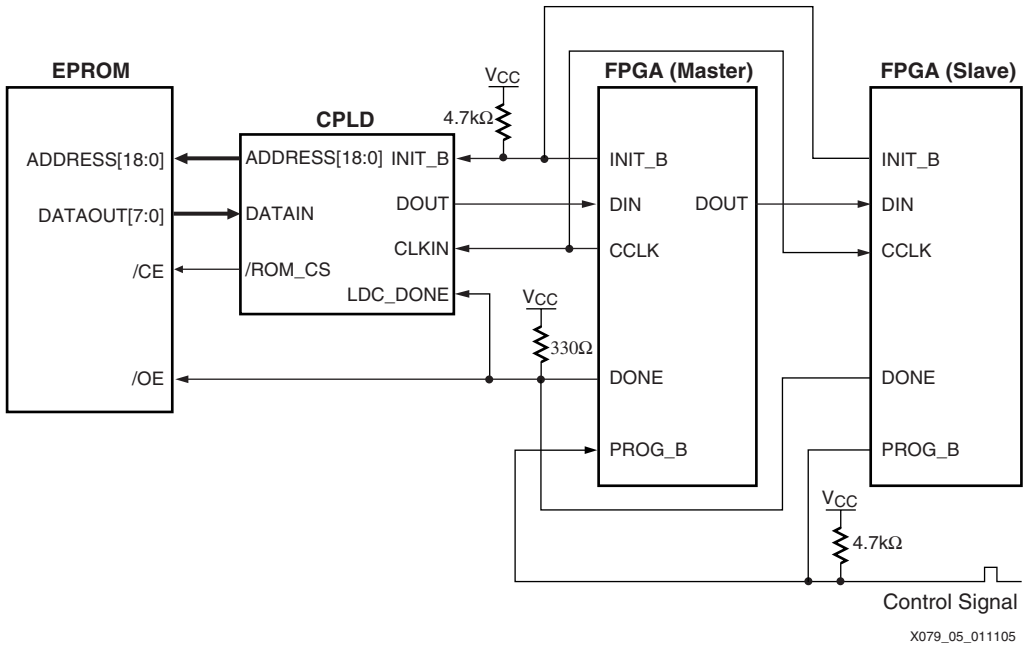


Figure 4-5: **Configuring Virtex-II Using a CPLD and Parallel PROM**

- **XAPP137**: *Configuring Virtex FPGAs From Parallel EPROMs With a CPLD* (available on **www.xilinx.com**) describes an architecture that configures one or more Virtex-II devices using the Slave SelectMAP mode. See Figure 4-6 for an example of FPGA configuration using a CPLD and a parallel EPROM.

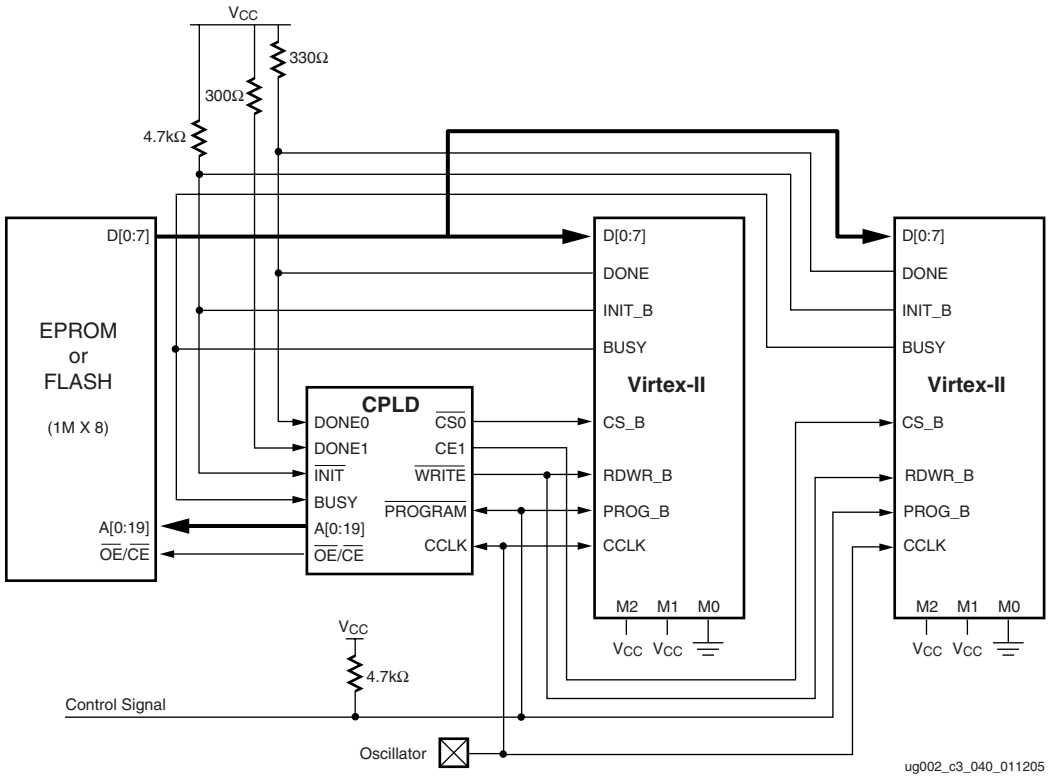


Figure 4-6: **Configuring Virtex-II from Parallel EPROMs**

Embedded Solutions

Using an Embedded Microcontroller

XAPP058, "Xilinx In-System Programming Using an Embedded Microcontroller" (available on **www.xilinx.com**) describes a compact and robust process that (re)configures Virtex-II devices directly from a microprocessor through the JTAG test port of the Virtex-II device. The process additionally supports (re)configuration of XC18V00 ISP PROMs and CPLDs that reside on the JTAG scan chain. Portable, reference C-code is provided with the application note for rapid implementation.

Using IEEE Standard 1532

Systems that implement an IEEE Standard 1532 player can configure Virtex-II devices. Users need a 1532 BSDL file and a 1532 configuration data file. 1532 BSDL files for Xilinx devices and information on the Xilinx J DRIVE 1532 configuration engine are available here:

http://www.support.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=Configuration+Solutions

Choose the **JDrive Engine** entry from the selection menu on the left of the page.

PROM Selection Guide

Use Table 4-5 to determine which PROMs go with which Virtex-II FPGAs.

Table 4-5: **Using Virtex-II Devices with PROMs**

Virtex-II Device	Default Bitstream Size (bits)(1)	PROM Solutions	
		One-Time Programmable (OTP)	In-System Programmable (ISP)
XCV2V40	360,096	17V01	XCF01S
XCV2V80	635,296	17V01	XCF01S
XCV2V250	1,697,184	17V02	XCF02S
XCV2V500	2,761,888	17V04	XCF04S
XCV2V1000	4,082,592	17V04	XCF04S
XCV2V1500	5,659,296	17V08	XCF08P
XCV2V2000	7,492,000	17V08	XCF08P
XCV2V3000	10,494,368	17V16	XCF16P
XCV2V4000	15,659,936	17V16	XCF16P
XCV2V6000	21,849,504	17V16 + 17V08	XCF32P
XCV2V8000	29,063,072	2 x 17V16	XCF32P

Notes:

1. Bitstream sizes for devices with regular multiplier. See Table 4-2 on page 271 for information about devices with enhanced multiplier.

Software Support and Data Files

This section provides information on Xilinx device programming software and configuration-related data files.

iMPACT Software

For programming Virtex-II and other Xilinx devices with a personal computer, Xilinx provides iMPACT software as a part of the ISE software package. A free version of iMPACT software is also available through the WebPACK software suite. More information on WebPACK is available at **http://www.support.xilinx.com**.

Programming Cables

iMPACT software supports several Xilinx programming cables that are compatible with Virtex-II devices, including the Parallel Cable III, Parallel Cable IV, and Platform Cable USB. For more information on these cables, or to order programming cables online, visit **http://www.support.xilinx.com**.

Boundary Scan Interconnect Testing for Virtex-II Devices

Virtex-II devices support the EXTEST, INTEST, and SAMPLE/PRELOAD instructions required for Boundary Scan interconnect tests. Xilinx does not provide direct support for Boundary Scan software, although several third party suppliers offer Boundary Scan test equipment that is compatible with Virtex-II devices. A list of third-parties offering Boundary Scan test equipment is available online at **http://www.support.xilinx.com**.

Customers seeking to perform interconnect tests on a Virtex-II device using third-party boundary scan tester must have a Boundary Scan Description Language (BSDL) file for the Virtex-II device. BSDL files for all Xilinx devices are provided with the ISE software installation, and are also available online.

In-System Programming Data Files

Many third party JTAG configuration solutions and in-system configuration solutions require an SVF (Serial Vector Format) or STAPL (Standard Test and Programming Language) file. The SVF and STAPL file formats are used to convey Boundary Scan instructions in a generic format. Customers requiring an SVF or STAPL file can use iMPACT software to generate these files. For more information on SVF, STAPL, and In-System Programming (ISP), see the following resources online:

- The iMPACT *Software Manual* is included on the software manuals web page.
- In-System Programming details are contained in Xilinx Application Note 058.

Serial Programming Modes

In the serial programming modes, the FPGA is configured by loading one bit per CCLK cycle. In Master Serial mode, the FPGA drives the CCLK pin. In Slave Serial mode, the FPGAs CCLK pin is driven by an external source. In both serial configuration modes, the MSB of each data byte is always written to the DIN pin first.

Master Serial Mode

The Master Serial mode is designed so the FPGA can be configured from a Serial PROM, Figure 4-7. The speed of the CCLK is selectable by BitGen options; see Appendix A: BitGen and PROMGen Switches and Options. Be sure to select a CCLK speed supported by the PROM.

Figure 4-7 shows a Master Serial FPGA configuring from a PROM.

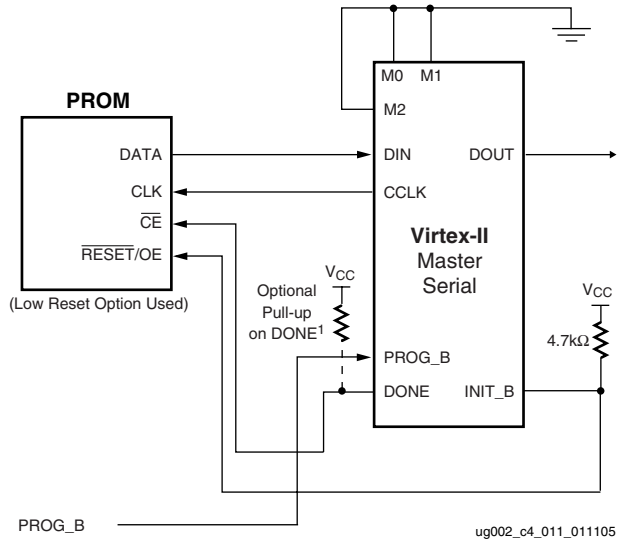


Figure 4-7: **Master Serial Mode Circuit Diagram**

Notes:

1. If the Virtex-II device has not selected the DriveDONE option, then an external pull-up resistor of 330Ω should be added to the DONE pin. This pull-up resistor is not needed if DriveDONE = Yes.

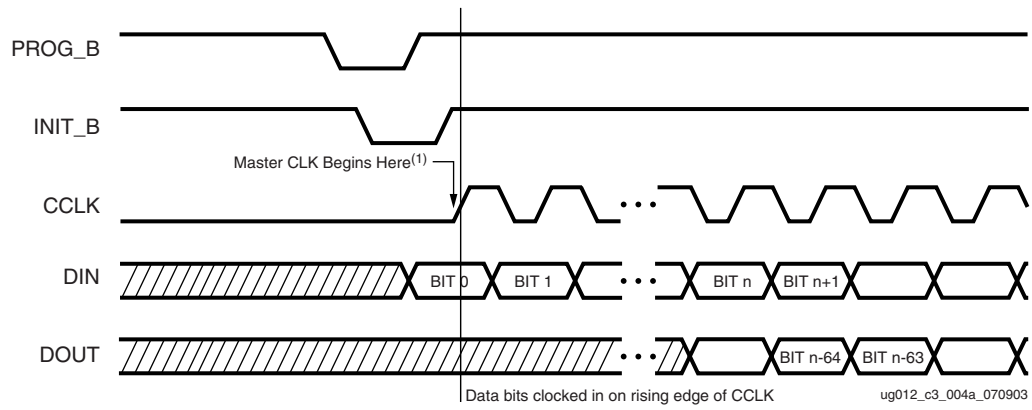


Figure 4-8: **Master Serial Configuration Clocking Sequence**

Notes:

1. For Master configurations, CCLK does not transition until after initialization, as indicated by arrow.
2. DOUT transitions on the falling edge of CCLK.

Slave Serial Mode

In serial configuration mode, the FPGA is configured by loading one bit per CCLK cycle. In Slave Serial mode, the FPGAs CCLK pin is driven by an external source. In both serial configuration modes, the MSB of each data byte is always written to the DIN pin first.

The Slave Serial configuration mode allows for FPGAs to be configured from other logic devices, such as microprocessors, or in a daisy-chain fashion. Figure 4-9 shows a Master Serial FPGA configuring from a PROM with a Slave Serial FPGA in a daisy-chain with the Master.

Daisy-Chain Configuration

Virtex-II devices can be daisy-chained with the Virtex, Spartan-II, Virtex-E, Spartan-IIE, Spartan-3, Virtex-II Pro, and Virtex-4 families. There are three important design considerations when designing a mixed-serial daisy chain:

1. Many older devices cannot accept as fast a CCLK frequency as (for example) a Virtex-4 device can generate. Select a configuration CCLK speed supported by *all* devices in the chain.
2. Newer devices should be grouped at the beginning of the serial daisy chain, with older devices located at the end of the chain.
3. There is a limit to the number of configuration bits that a device can pass through its DOUT pin. As shown in Table 4-6, this limit varies for different families. The sum of the bitstream lengths for all downstream devices must not exceed the number in this table for each family.

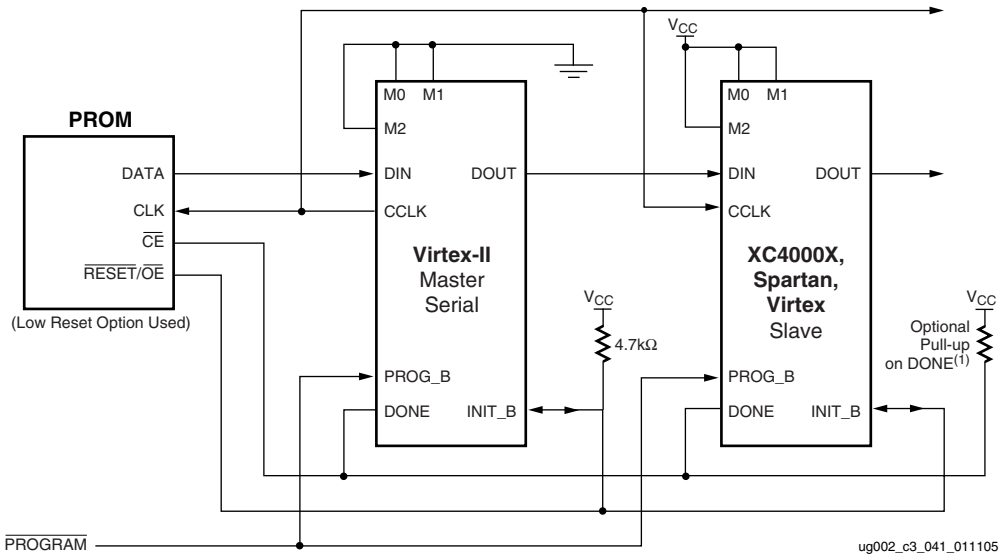


Figure 4-9: **Master/Slave Serial Mode Daisy-Chain Circuit Diagram**

Notes:

1. If none of the devices have been selected to DriveDONE, then an external pull-up resistor of 330Ω should be added to the common DONE line. This pull-up resistor is not needed if DriveDONE = Yes. If used, DriveDONE should be enabled only for the last device in the configuration chain.

Table 4-6: **Maximum Number of Configuration Bits, Various Device Families**

Architecture	Maximum DOUT Bits
Virtex-4	32 x(2^{27} – 1) = 4,294,967,264
Virtex-II Pro, Virtex-II	32 x(2^{27} – 1) = 4,294,967,264
Spartan-3	32 x (2^{27} – 1) = 4,294,967,264
Virtex, Virtex-E, Spartan-II, Spartan-IIE	32 x (2^{20} – 1) = 33,554,216

The separate bitstreams for the FPGAs in a daisy-chain must be combined into a single PROM file, by using either iMPACT software or the PROMGen utility (see Appendix A: BitGen and PROMGen Switches and Options). Separate .bit files cannot be simply concatenated together to form a daisy-chain bitstream.

The first device in the chain is the first to be configured. No data is passed onto the DOUT pin until all the data frames, start-up command, and CRC check have been loaded. CRC checks only include the data for the current device, not for any others in the chain. After finishing the first stream, data for the next device is loaded. The data for the downstream device appears on DOUT typically about 80 CCLK cycles after being loaded into DIN. This is due to internal packet processing. Each daisy-chained bitstream carries its own synchronization word. Nothing of the first bitstream is passed to the next device in the chain other than the daisy-chained configuration data.

The DONE_cycle must be set before GTS, or during the same cycle to guarantee each Virtex-II device to move to the operation state when all the DONE pins have been released. When daisy-chaining multiple devices, either set the last device in the chain to DriveDONE, or add external pull-up resistors to counteract the combined capacitive loading on DONE. If non-Virtex devices are included in the daisy-chain, it is important to set their bitstreams to SyncToDONE with BitGen options. For more information on Virtex BitGen options, see Appendix A: BitGen and PROMGen Switches and Options.

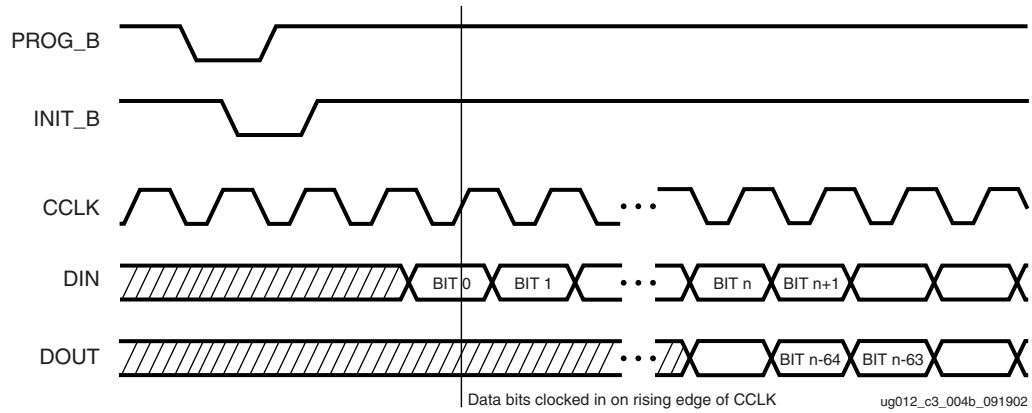


Figure 4-10: **Serial Configuration Clocking Sequence**

Notes:
1. For Slave configurations, a free running CCLK can be used, as shown in Figure 4-10.
2. DOUT transitions on the falling edge of CCLK.

SelectMAP Programming Modes

The SelectMAP™ interface provides an 8-bit bidirectional data bus interface to the Virtex-II Pro configuration logic. In Master SelectMAP mode the CCLK signal is an output from the FPGA. In Slave SelectMAP mode the CCLK signal is an input. Slave SelectMAP mode allows for both configuration and readback, while only configuration is possible in Master SelectMAP mode.

Table 4-7 describes the SelectMAP configuration interface pins.

Table 4-7: **SelectMAP Configuration Interface Pins**

Pin Name	Type	Dedicated or Dual-Function	Description
D[0:7]	Bidirectional	Dual-Function	The D[0:7] pins function as a bi-directional data bus for the SelectMAP interface, with D0 serving as the MSB. The D[0:7] bus direction is controlled by the RDWR_B signal. When the SelectMAP interface is enabled (CS_B=0) and the port is set for write control (RDWR_B=0), the D[0:7] signals are registered by rising CCLK edges. When the SelectMAP interface is disabled (CS_B=1), the D[0:7] pins are placed in a high-Z state. After configuration, the D[0:7] pins become user I/O unless the bitgen **`-g persist:yes`** setting is used.
RDWR_B	Input	Dual-Function	The RDWR_B signal determines whether the SelectMAP interface is set for read or write control. 0 = write control, D[0:7] are inputs 1 = read control, D[0:7] are outputs The RDWR_B pin is registered on rising CCLK edges. If a new RDWR_B value is registered by the device while the SelectMAP interface is enabled (CS_B=0), an ABORT will occur (see "SelectMAP ABORT Sequence and ABORT Recovery" on page 294). After configuration, the RDWR_B pin becomes user I/O unless the bitgen **`-g persist:yes`** setting is used.
CS_B	Input	Dual-Function	Active-Low chip select to enable the SelectMAP data bus. 0 = SelectMAP data bus enabled 1 = SelectMAP data bus disabled. When the CS_B signal is deasserted, the SelectMAP interface is asynchronously disabled and all SelectMAP outputs are placed in a high-Z state. After configuration, the CS_B pin becomes user I/O unless the bitgen **`-g persist:yes`** setting is used.
BUSY	Three-State Output	Dual-Function	The BUSY pin is an active-High three-state output. 0 = BUSY deasserted 1 = BUSY asserted The BUSY pin has two functions: 1. During configuration, the BUSY pin indicates that data are being loaded too quickly. Values on the D[0:7] bus are ignored while BUSY is active. 2. During readback, BUSY indicates when readback data is valid. When the SelectMAP interface is disabled (CS_B=1), the BUSY pin is placed in a high-Z state. After configuration, the BUSY pin becomes user I/O unless the bitgen **`-g persist:yes`** setting is used.
M[2:0]	Input	Dedicated	MODE pins. Specify configuration mode.

Table 4-7: **SelectMAP Configuration Interface Pins *(Continued)***

Pin Name	Type	Dedicated or Dual-Function	Description
CCLK	Input or Output	Dedicated	Configuration clock for all modes other than JTAG. Input for Slave SelectMAP mode; output for Master SelectMAP mode.
PROG_B	Input	Dedicated	Active-Low asynchronous full-chip reset. Has permanent weak pull-up.
INIT_B	Bi-directional Open-Drain	Dual-Function	Before MODE pins are sampled, INIT_B is an input that can be held Low to delay configuration. After MODE pins are sampled, INIT_B is an open-drain, active Low output indicating whether a CRC error occurred during configuration: 0 = CRC error 1 = No CRC error After configuration, the INIT_B pin becomes user I/O unless the bitgen **-g persist:yes** setting is used.
DONE	Input or Output	Dedicated	Active High signal indicating configuration is complete: 0 = FPGA not configured 1 = FPGA configured Pin has optional internal pull-up.

Daisy-chaining is not possible for SelectMAP mode, however multiple devices can be connected on the same SelectMAP bus as shown in Figure 4-13 on page 290. To connect devices on a SelectMAP bus, the DATA pins (D0:D7), CCLK, RDWR_B, BUSY, PROG_B, DONE, and INIT_B are connected in common between all devices. The CS_B (Chip Select) inputs are kept separate so that each device can be accessed individually. External control logic is required to arbitrate between devices by asserting and deasserting the CS_B signals as necessary.

Alternatively, it is possible to simultaneously configure multiple devices using the same bitstream, provided that readback is not required. In this arrangement the CS_B pins are connected together, along with the remaining configuration pins, so that the SelectMAP ports on all devices are enabled simultaneously.

Master SelectMAP Mode

The Master SelectMAP mode is used to connect a single Virtex-II FPGA to one or more configuration PROMs as shown in Figure 4-11.

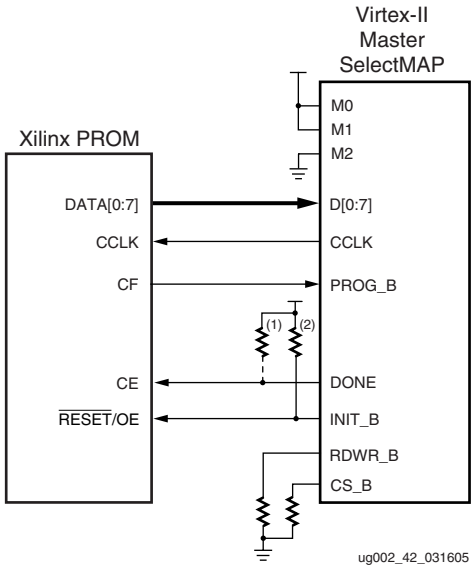


Figure 4-11: **Single-Device Master SelectMAP Configuration**

Notes:

1. The DONE pin is by default an open-drain output requiring an external pull-up resistor. In this arrangement, the active DONE driver can be enabled, eliminating the need for an external pull-up resistor.
2. The INIT_B pin is a bidirectional, open-drain pin. An external pull-up is required.
3. The BitGen startup clock setting must be set for CCLK for SelectMAP configuration.
4. The PROM in this diagram represents one or more Xilinx serial PROMs. Multiple serial PROMs can be cascaded to increase the overall configuration storage capacity.
5. The .bit file must be reformatted into a PROM file before it can be stored on the serial PROM. Refer to Appendix A: BitGen and PROMGen Switches and Options.
6. On XC17V00 devices, the reset polarity is programmable. $\overline{\text{RESET}}$ should be set for active Low when using an XC17V00 device in this setup.
7. The Xilinx PROM must be set for parallel mode. (This mode is not available for all devices.)
8. When configuring a Virtex-II device in SelectMAP mode from a Xilinx configuration PROM, the RDWR_B and CS_B signals can be tied Low (see section Master SelectMAP Data Loading).
9. The BUSY signal does not need to be monitored for this setup and can be left unconnected (see section Master SelectMAP Data Loading).

The following pins are involved in Master SelectMAP configuration mode:

DATA Pins (D[0:7])

The D0 through D7 pins function as a bidirectional data bus in the SelectMAP mode. Configuration data is written to the bus, and readback data is read from the bus. The bus direction is controlled by the RDWR_B signal. see “Configuration Details” on page 314. The D0 pin is considered the MSB of each byte.

RDWR_B

When asserted Low, the RDWR_B signal indicates that data is being written to the data bus. When High, the RDWR_B signal indicates that data is being read from the data bus.

CS_B

The Chip Select input (CS_B) enables the SelectMAP data bus. To write or read data onto or from the bus, the CS_B signal must be asserted Low. When CS_B is High, Virtex-II devices do not drive onto or read from the bus.

CCLK

The CCLK pin is a clock output in the Master SelectMAP interface. It synchronizes all loading and reading of the data bus for configuration and readback. The CCLK pin is driven by the FPGA.

Master SelectMAP Data Loading

To load data in the Master SelectMAP mode, a data byte is loaded on every rising CCLK edge as shown in Figure 4-12. If BUSY is being monitored during configuration (for example, when configuring with an encrypted bitstream) and it is High, the current byte must be reloaded when BUSY is Low.

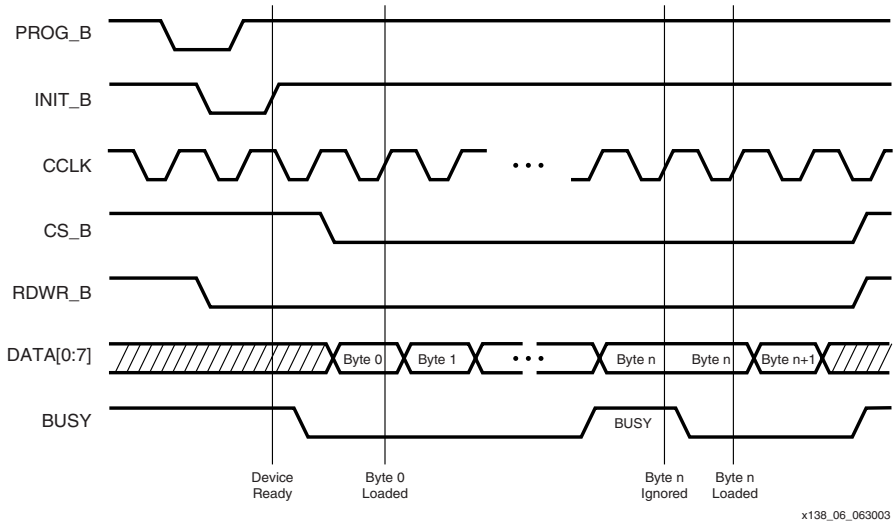


Figure 4-12: **Data Loading in SelectMAP**

The first byte can be loaded on the first rising CCLK edge that INIT_B is High, and when both CS_B and RDWR_B are asserted Low. CS_B and RDWR_B can be asserted anytime before or after INIT_B has gone High. However, the SelectMAP interface is not active until after INIT_B has gone High. If RDWR_B toggles while CS_B is asserted, an ABORT occurs.

Extra processing time is needed when the bitstream is encrypted. Because of this, BUSY handshaking is required for encrypted bitstreams at a lower speed than for non-encrypted bitstreams. The maximum speed at which a Virtex-II device can be configured via SelectMAP without BUSY handshaking is 50 MHz.

Slave SelectMAP Mode

One or more Virtex-II Pro devices in Slave SelectMAP mode can be connected on a common SelectMAP bus (Figure 4-13). In a SelectMAP bus, the DATA pins (D[0:7]), CCLK, RDWR_B, BUSY, PROG_B, DONE, and INIT_B share a common connection between all of the devices. To allow each device to be accessed individually, the CS_B (Chip Select) inputs must not be tied together. External control of the CS_B signal is required and is usually provided by a microprocessor or CPLD.

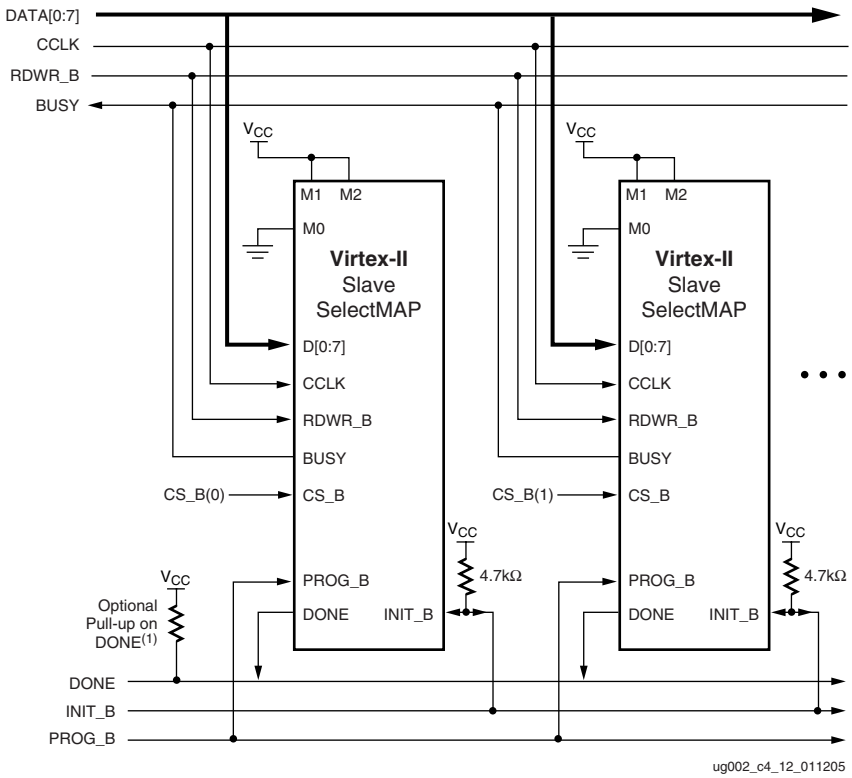


Figure 4-13: **Slave SelectMAP Mode Circuit Diagram**

Notes:

1. If none of the Virtex-II devices have been selected to DriveDONE, add an external 330Ω pull-up resistor to the common DONE line. This pull-up resistor is not needed if DriveDONE = Yes. If used, DriveDONE should be selected only for the last device in the configuration chain.

If Readback is going to be performed on the device after configuration, the RDWR_B and BUSY signals must be handled appropriately. (For details, refer to Chapter 8, "Readback and Configuration Verification.")

Although Figure 4-13 does not show a control module for the SelectMAP interface, the SelectMAP interface is typically driven by a processor, microcontroller, FPGA or CPLD.

Controlling SelectMAP Data Flow

The rules for sending configuration data to a Virtex-II Pro device via the SelectMAP interface follow:

- All SelectMAP inputs are ignored until the INIT_B pin goes high after power-up. The RDWR_B and CS_B signals can be asserted prior to or during power-up without triggering an ABORT.
- After INIT_B has gone high, the RDWR_B signal should be set before asserting CS_B. Registering a new value on RDWR_B while CS_B is enabled will trigger an ABORT. If necessary, the RDWR_B signal can be toggled without triggering an ABORT if:
  - CS_B is deasserted, or
  - if CS_B is asserted BUT no positive CCLK edge arrives

  Refer to "Toggling between Read and Write Control" on page 293 for details.
- When the SelectMAP port is set for write control (RDWR_B=0), the device samples the D0:7 inputs on each rising CCLK edge.
- The BUSY signal must be monitored during readback and when bitstream encryption is used during configuration.

There are several ways to control configuration data loading using Slave SelectMAP mode:

1. **Continuous Data Loading (Free-Running CCLK).**

   After the INIT_B signal goes High, the device registers the D[0:7] inputs on every rising CCLK edge while the SelectMAP interface is enabled (CS_B=0) and set for write control (RDWR_B=0). Configuration data are loaded continuously, without pauses. This method is depicted in Figure 4-14.

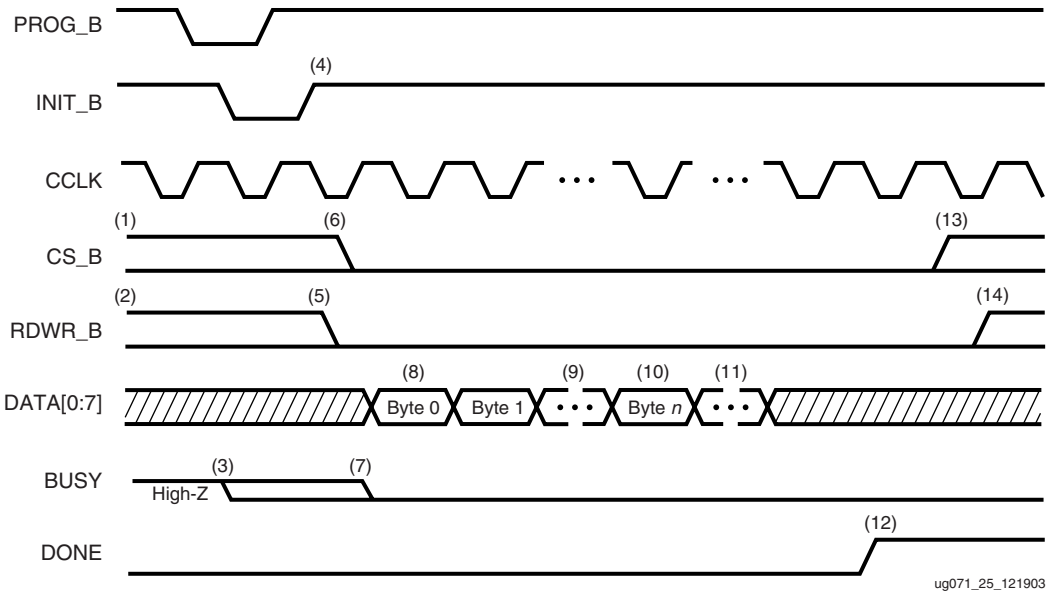


Figure 4-14: **Continuous Data Loading for SelectMAP Modes**

Notes:

1. CS_B signal can be tied Low if there is only one device on the SelectMAP bus. If CS_B is not tied Low, it can be asserted at any time.
2. RDWR_B can be tied Low if readback is not needed. RDWR_B should not be toggled after CS_B has been asserted, as this triggers an ABORT. (See section SelectMAP ABORT Sequence and ABORT Recovery.)
3. If CS_B is tied Low, BUSY is driven Low before INIT_B toggles High.
4. The MODE pins are sampled when INIT_B goes High.
5. RDWR_B should be asserted before CS_B to avoid causing an ABORT.
6. CS_B is asserted, enabling the SelectMAP interface.
7. BUSY remains in high-Z state until CS_B is asserted.
8. The first byte is loaded on the first rising CCLK edge after CS_B is asserted.
9. The configuration bitstream is loaded one byte per rising CCLK edge.
10. After the last byte is loaded the device enters the startup sequence.
11. The startup sequence lasts a minimum of eight CCLK cycles.
12. The DONE pin goes High during the startup sequence. Additional CCLKs could be required to complete the startup sequence.
13. After configuration has finished, the CS_B signal can be deasserted.
14. After the CS_B signal is deasserted, RDWR_B can be deasserted.

2. SelectMAP Data Loading Controlled by the CS_B signal (Non-Contiguous Data Strobe).

In some cases configuration will need to be paused while the next byte of configuration data is presented on the D[0:7] pins. One way of pausing SelectMAP configuration is to disable the SelectMAP interface (CS_B=0). When the SelectMAP interface is disabled, the D[0:7] and BUSY pins are placed in a high-Z state, and all inputs are ignored. The D[0:7] signals are sampled on the next rising CCLK edge after the SelectMAP interface is enabled. See Figure 4-15.

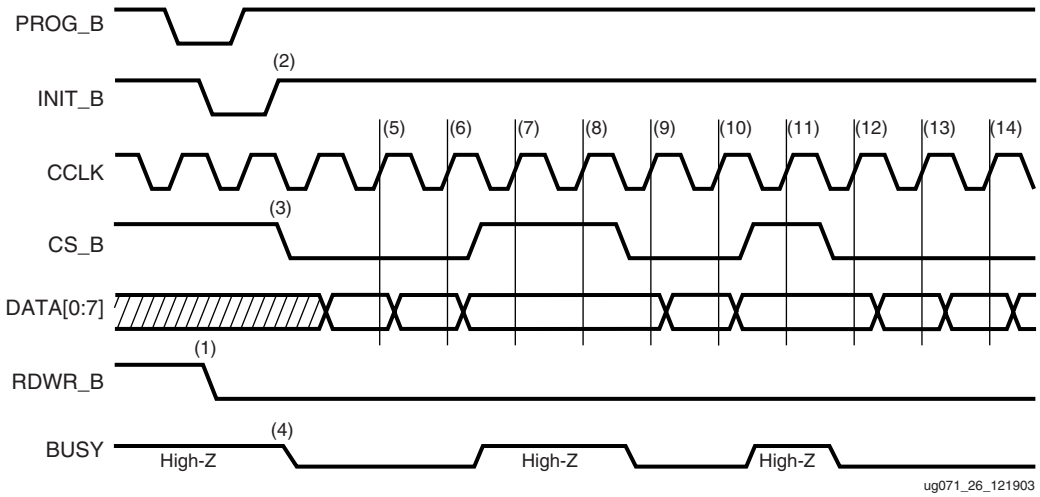


Figure 4-15: **SelectMAP Data Loading Controlled by the CS_B Signal**

Notes:

1. RDWR_B is driven Low by the user, setting the D[0:7] pins as inputs for configuration. RDWR_B can be tied Low if readback is not needed. RDWR_B should not be toggled after CS_B has been asserted, as this triggers an ABORT. (See section SelectMAP ABORT Sequence and ABORT Recovery.)
2. Device is ready for configuration after INIT_B toggles High.
3. The user asserts CS_B Low, enabling SelectMAP data bus. CS_B signal can be tied Low if there is only one device on the SelectMAP bus. If CS_B is not tied Low, it can be asserted at any time.
4. BUSY goes Low shortly after CS_B is asserted. If CS_B is tied Low, BUSY is driven Low before INIT_B toggles High.
5. Byte loaded on rising CCLK edge.
6. Byte loaded on rising CCLK edge.
7. The user deasserts CS_B; byte ignored.
8. The user deasserts CS_B; byte ignored.
9. Byte loaded on rising CCLK edge.
10. Byte loaded on rising CCLK edge.
11. The user deasserts CS_B; byte ignored.
12. Byte loaded on rising CCLK edge.
13. Byte loaded on rising CCLK edge.
14. Byte loaded on rising CCLK edge.

3. **Data Loading Controlled by Gating CCLK (Controlled CCLK).**

   An alternate way to control SelectMAP data loading is to gate (halt) the CCLK signal. If needed, the RDWR_B signal can be toggled during this time, as long as it is returned to its prior value before the next rising CCLK edge. If the RDWR_B signal is driven high (set for read control) while the SelectMAP interface is enabled (CS_B=0), the D[0:7] pins will become outputs, and will be driven actively by the device. The D[0:7] signals will retain their state while CCLK is paused. See Figure 4-16.

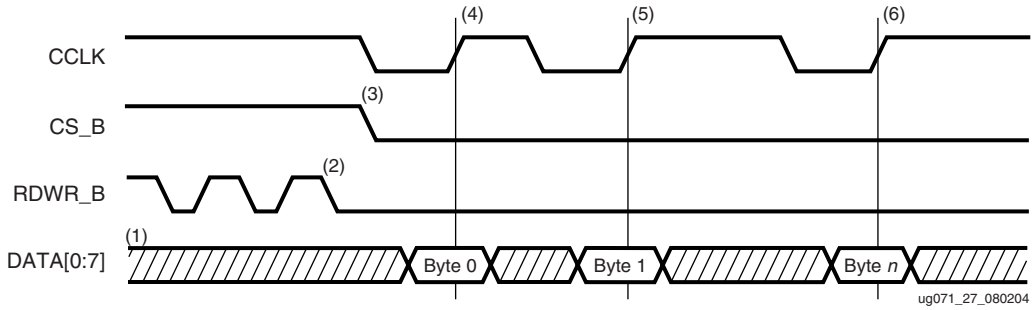


Figure 4-16: **Data Loading Controlled by gating CCLK**

Notes:

1. D[0:7] pins are in High-Z state while CS_B is deasserted.
2. RDWR_B has no effect on the device while CS_B is deasserted.
3. CS_B is asserted by the user. Device begins loading configuration data on rising CCLK edges.
4. Byte loaded on rising CCLK edge.
5. Byte loaded on rising CCLK edge.
6. Byte loaded on rising CCLK edge.

Toggling between Read and Write Control

This section explains how to change the SelectMAP interface from write control to read control and vise-versa. This is only needed for readback operations, during which the SelectMAP port must be set for write control to send commands to the device, then toggled to read control to retrieve configuration data from the device (see "Readback" on page 342). For designs that do not require readback, the RDWR_B signal can be tied Low.

Write and read control for the SelectMAP interface is determined by the RDWR_B input: the SelectMAP data pins (D0:7) are inputs when the interface is set for Write control (RDWR_B = 0); they are outputs when the interface is set for Read control (RDWR_B = 1).

The procedure for changing the SelectMAP interface from Write to Read Control, or vice-versa, is:

1. **Deassert CS_B (set CS_B = 1).**

   After deasserting CS_B, he device reacts by asynchronously disabling the SelectMAP interface, placing the D[0:7] pins and the BUSY pin in a high-Z state.

2. **Toggle RDWR_B**

   RDWR_B = 0 : Write control
   RDWR_B = 1 : Read control

3. **Assert CS_B (set CS_B = 0)**

   After asserting CS_B, the BUSY signal and the D[0:7] pins will become active on the next rising CCLK edge, with the BUSY signal asserted. After a few CCLK cycles (the exact number is variable), the device will deassert the BUSY signal.

When switching to write control (writing data to the FPGA), the device begins sampling data on the D[0:7] pins after it deasserts the BUSY signal. Similarly, when switching to read control, the device presents valid data on the D[0:7] pins after it deasserts the BUSY signal.

This procedure is illustrated in Figure 4-17.

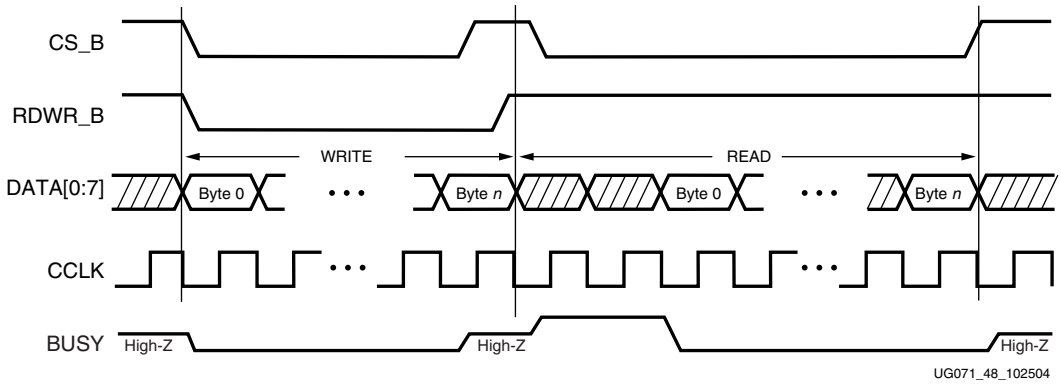


Figure 4-17: **Changing the SelectMAP Port from Write to Read Control**

SelectMAP ABORT Sequence and ABORT Recovery

An ABORT is an interruption in the SelectMAP configuration or readback sequence that occurs when the user improperly toggles the RDWR_B signal. In some cases, this might be done intentionally, because the device will provide a status word that can be useful in debugging configuration problems.

Triggering an ABORT

An ABORT occurs when the device registers a new RDWR_B state while CS_B remains asserted and a rising CCLK edge occurs.

ABORT During Configuration

An ABORT is triggered during configuration as follows:

1. Configuration sequence begins normally.
2. User pulls the RDWR_B pin high while CS_B is still asserted.
3. The device sees a rising edge on CCLK.
4. If CS_B remains asserted and RDWR_B remains high (SelectMAP interface is set for read control), the device drives BUSY high and it drives the ABORT Status word on the D[0:7] outputs (see "ABORT Status Word" on page 295).
5. The ABORT ends after four CCLK cycles, regardless of whether or not CS_B is asserted.

This sequence is illustrated by the timing diagram in Figure 4-18.

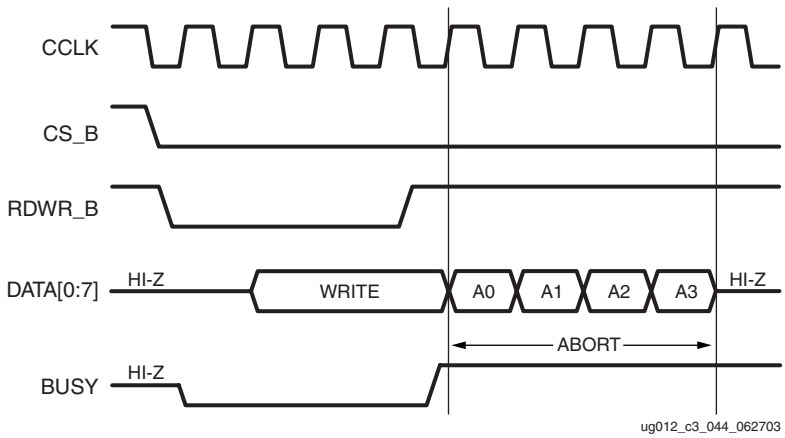


Figure 4-18: **Configuration ABORT Sequence**

ABORT During Readback

An ABORT is triggered during readback as follows:

1. Readback sequence begins normally.
2. User pulls the RDWR_B pin low while CS_B is still asserted.
3. The device sees a rising edge on CCLK.
4. If CS_B remains asserted and RDWR_B remains low (SelectMAP interface is set for write control), the device places BUSY and the D[0:7] pins in a high-Z state.
5. The ABORT ends after four CCLK cycles, regardless of whether or not CS_B is asserted. If the RDWR_B signal is toggled to read control (RDWR_B=0) before or during the ABORT, the current status byte will appear on the D[0:7] pins.

This sequence is illustrated by the timing diagram in Figure 4-19.

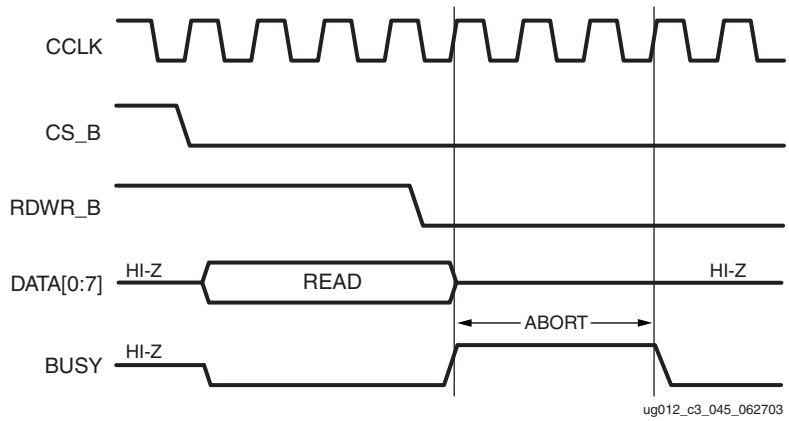


Figure 4-19: **Readback ABORT Sequence**

Note that ABORTs during readback are not followed by a status word if the RDWR_B signal remains set for write control.

ABORT Status Word

During the configuration ABORT sequence, the device drives a status word onto the D[0:7] pins. The key for that status word is as follows:

Table 4-8: **ABORT Status Word**

Bit Number	Status Bit Name	Meaning
D7	CFGERR_B	Configuration error (active Low) 0 = A configuration error has occurred. 1 = No configuration error.
D6	DALIGN	Sync word received (active High) 0 = No sync word received. 1 = Sync word received by interface logic.
D5	RIP	Readback in progress (active High) 0 = No readback in progress. 1 = A readback is in progress.
D4	IN_ABORT_B	ABORT in progress (active Low) 0 = ABORT is in progress. 1 = No ABORT in progress.
D3-D0	1111	

The ABORT sequence lasts for four CCLK cycles. The status byte reflects data alignment, ABORT status, and whether readback is in progress. An example sequence is given in Table 4-9.

Table 4-9: **ABORT Sequence Example**

Byte Number	D[0:7]	DALIGN	IN_ABORT_B
0	11111011	1	1
1	11110011	1	0
2	11110001	0	0
3	11111001	0	1

After the last cycle, the synchronization word can be reloaded to establish data alignment.

ABORT Recovery

There are two procedures for recovering from an ABORT—one simple, the other more complex. These procedures apply to both read- and write-ABORT recovery.

ABORT Recovery Procedure 1

Pulse the PROG_B pin at any time during or after the ABORT sequence. Configuration must start over completely.

ABORT Recovery Procedure 2

This procedure is more complicated, but it does not involve clearing configuration memory. This allows configuration or readback to resume from where it was interrupted by the ABORT sequence. To take advantage of this capability, however, users must have fine control over, and a detailed understanding of, the configuration process. Use of this procedure will require a thorough knowledge of the material in the Configuration Details and Readback sections.

1. Wait until the ABORT sequence has finished
2. Toggle the SelectMAP interface to Write control
3. Write the Synchronization Word
4. Write a value equal to the (number of words per frame – 1) to the FLR
5. Write the RCFG command
6. Set the FAR to address `0xFFFFFFFF` (out of bounds)
7. Send the FDRO read header to the device. The read length is a number of words equal to the frame length.
8. Write four NO-OP commands
9. Toggle the SelectMAP interface to Read control
10. Read 1 frame from the device
11. Toggle the SelectMAP interface to Write control

Steps to Resume Readback:

12. Send RCFG command
13. Set the FAR to the desired address
14. Send the FDRO read header
15. etc…

Steps to Resume Configuration:

12. Send the WCFG command
13. Set the FAR to the desired address
14. Send the FDRI read header
15. etc…

Internal Configuration Access Port (ICAP)

Virtex-II devices feature an Internal Configuration Access Port (ICAP) that provides a configuration interface to the FPGA fabric. This allows the user design to control configuration and readback operations. The ICAP interface uses the same protocol as the SelectMAP interface in Slave mode, although it features separate data input and output ports, whereas the D[0:7] pins on the SelectMAP interface are bidirectional (see Table 4-7 on page 286). To use ICAP, the ICAP_VIRTEX2 primitive must be instantiated in the user design. The ICAP_VIRTEX2 primitive is shown in Figure 4-20.

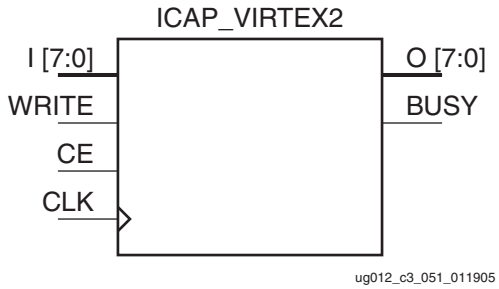


Figure 4-20: **ICAP_VIRTEX2 Primitive**

The ports on the ICAP primitive correspond to ports on the SelectMAP interface as shown in Table 4-10. The ICAP ports behave exactly as do the equivalent SelectMAP ports.

Table 4-10: **ICAP Ports and Equivalent SelectMAP Ports**

ICAP Port		Equivalent SelectMAP Port
Name	Direction	
BUSY	Output	BUSY
O[0:7]	Output	D[0:7] when SMAP port is set for read control
CE	Input	CS_B
CLK	Input	CCLK
I[0:7]	Input	D[0:7] when SMAP port is set for write control
WRITE	Input	RDWR_B

The following restrictions govern the use of ICAP:

- ICAP and SelectMAP cannot be used together after configuration.
  - After the DONE pin goes High, the ICAP port will be disabled if the PERSIST bit is set.
  - A design that instantiates the ICAP port can be configured via SelectMAP, provided that the PERSIST bit is not set.
- Care must be taken when using the JTAG interface on a design that also accesses the ICAP port.

- The JTAG CFG_IN, CFG_OUT, JSTART, and JSHUTDOWN instructions must not be sent to the device while configuration or readback operations are being performed through the ICAP port.
- After finishing a configuration or readback operation on the JTAG interface, the configuration logic should be desynchronized before performing any operations using the ICAP port. Conversely, after finishing an operation on the ICAP port, the configuration logic should be desynchronized before performing any operations using the JTAG interface.

JTAG / Boundary Scan Programming Mode

Introduction

Virtex-II devices support the new IEEE 1532 standard for In-System Configuration (ISC), based on the IEEE 1149.1 standard. The IEEE 1149.1 Test Access Port and Boundary-Scan Architecture is commonly referred to as JTAG. JTAG is an acronym for the Joint Test Action Group, the technical subcommittee initially responsible for developing the standard. This standard provides a means to assure the integrity of individual components and the interconnections between them at the board level. With increasingly dense multi-layer PC boards, and more sophisticated surface mounting techniques, boundary-scan testing is becoming widely used as an important debugging standard.

Devices containing boundary-scan logic can send data out on I/O pins in order to test connections between devices at the board level. The circuitry can also be used to send signals internally to test the device specific behavior. These tests are commonly used to detect opens and shorts at both the board and device level.

In addition to testing, boundary-scan offers the flexibility for a device to have its own set of user-defined instructions. The added common vendor-specific instructions, such as configure and verify, have increased the popularity of boundary-scan testing and functionality.

Boundary-Scan for Virtex-II Devices Using IEEE Standard 1149.1

The Virtex-II family is fully compliant with the IEEE Standard 1149.1 Test Access Port and Boundary-Scan Architecture. The architecture includes all mandatory elements defined in the IEEE 1149.1 Standard. These elements include the Test Access Port (TAP), the TAP controller, the instruction register, the instruction decoder, the boundary-scan register, and the bypass register. The Virtex-II family also supports some optional instructions; the 32-bit identification register, and a configuration register in full compliance with the standard. Outlined in the following sections are the details of the JTAG architecture for Virtex-II devices.

Test Access Port

The Virtex-II TAP contains four mandatory dedicated pins as specified by the protocol (Table 4-11).

Table 4-11: **Virtex-II TAP Controller Pins**

Pin	Description
TDI	Test Data In
TDO	Test Data Out
TMS	Test Mode Select
TCK	Test Clock

There are three input pins and one output pin to control the 1149.1 boundary-scan TAP controller. There are optional control pins, such as $\overline{\text{TRST}}$ (Test Reset) and enable pins,

which might be found on devices from other manufacturers. It is important to be aware of these optional signals when interfacing Xilinx devices with parts from different vendors, because they might need to be driven.

The TAP controller is a 16-state state machine shown in Figure 4-21. The four mandatory TAP pins are outlined below.

- TMS - This pin determines the sequence of states through the TAP controller on the rising edge of TCK. TMS has an internal resistive pull-up to provide a logic High if the pin is not driven.
- TCK - This pin is the JTAG test clock. It sequences the TAP controller and the JTAG registers in the Virtex-II devices.
- TDI - This pin is the serial input to all JTAG instruction and data registers. The state of the TAP controller and the current instruction held in the instruction register determine which register is fed by the TDI pin for a specific operation. TDI has an internal resistive pull-up to provide a logic High to the system if the pin is not driven. TDI is applied into the JTAG registers on the rising edge of TCK.
- TDO - This pin is the serial output for all JTAG instruction and data registers. The state of the TAP controller and the current instruction held in the instruction register determine which register (instruction or data) feeds TDO for a specific operation. TDO changes state on the falling edge of TCK and is only active during the shifting of instructions or data through the device. This pin is 3-stated at all other times.

  NOTE: As specified by the IEEE Standard, the TMS and TDI pins all have internal pull-up resistors by default. These internal pull-up resistors of 50-150 kΩ are active, regardless of the mode selected. User can select a pull-up, pull-down, or float option in the implementation tools.

For JTAG configuration mode, JTAG inputs are independent of V_{CCO} and work between 2.5V and 3.3V TTL levels (V_{IL} max = .8V, VIH min = 2.0V). The JTAG input pins are 3.3V tolerant. The JTAG output (TDO) is an active driver.

TAP Controller

Figure 4-21 diagrams a 16-state finite state machine. The four TAP pins control how data is scanned into the various registers. The state of the TMS pin at the rising edge of TCK determines the sequence of state transitions. There are two main sequences, one for shifting data into the data register and the other for shifting an instruction into the instruction register.

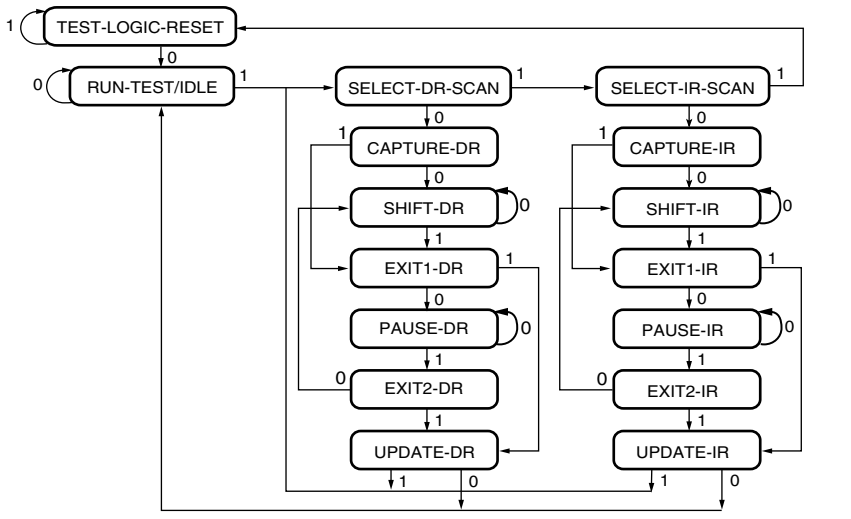


Figure 4-21: **State Diagram for the TAP Controller**

Boundary-Scan Instruction Set

To determine the operation to be invoked, an instruction is loaded into the Instruction Register (IR). The Instruction Register is 6 bits long in Virtex-II devices to support the new IEEE Standard 1532 for In-System Configurable (ISC) devices. Table 4-12 lists the available instructions for Virtex-II devices.

Table 4-12: **Virtex-II Boundary Scan Instructions**

Boundary Scan Command	Binary Code (5:0)	Description
EXTEST	000000	Enables boundary-scan EXTEST operation
SAMPLE	000001	Enables boundary-scan SAMPLE operation
USER1	000010	Access user-defined register 1
USER2	000011	Access user-defined register 2
CFG_OUT	000100	Access the configuration bus for readback
CFG_IN	000101	Access the configuration bus for configuration
INTEST	000111	Enables boundary-scan INTEST operation
USERCODE	001000	Enables shifting out user code
IDCODE	001001	Enables shifting out of ID code
HIGHZ	001010	3-states output pins while enabling the bypass register
JSTART	001100	Clocks the start-up sequence when StartClk is TCK
JSHUTDOWN	001101	Clocks the shutdown sequence
BYPASS	111111	Enables BYPASS
JPROG_B	001011	Equivalent to and has the same affect as PROG_B
RESERVED	All other codes	Xilinx reserved instructions

The mandatory IEEE 1149.1 commands are supported in Virtex-II devices, as well as several Xilinx vendor-specific commands. Virtex-II devices have a powerful command set. The EXTEST, INTEST, SAMPLE/PRELOAD, BYPASS, IDCODE, USERCODE, and HIGHZ instructions are all included. The TAP also supports two internal user-defined registers (USER1 and USER2) and configuration/readback of the device. The Virtex-II boundary-scan operations are independent of mode selection. The boundary-scan mode in Virtex-II devices overrides other mode selections. For this reason, boundary-scan instructions using the boundary-scan register (SAMPLE/PRELOAD, INTEST, EXTEST) must not be performed during configuration. All instructions except USER1 and USER2 are available before a Virtex-II device is configured. After configuration, all instructions are available.

JSTART and JSHUTDOWN are instructions specific to the Virtex-II architecture and configuration flow. As described in Table 4-12, the JSTART and JSHUTDOWN instructions clock the startup sequence when the appropriate BitGen option is selected. The instruction does not work correctly without the correct BitGen option selected.

```
bitgen -g startupclk:jtagclk designName.ncd
```

For details on the standard boundary-scan instructions EXTEST, INTEST, and BYPASS, refer to the IEEE Standard. The user-defined registers (USER1/USER2) are described in "USER1, USER2 Registers" on page 304.

Boundary-Scan Architecture

Virtex-II device registers include all registers required by the IEEE 1149.1 Standard. In addition to the standard registers, the family contains optional registers for simplified testing and verification (Table 4-13).

Table 4-13: **Virtex-II JTAG Registers**

Register Name	Register Length	Description
Instruction register	6 bits	Holds current instruction OPCODE and captures internal device status.
Boundary scan register	3 bits per I/O	Controls and observes input, output, and output enable.
Bypass register	1 bit	Device bypass.
Identification register	32 bits	Captures device ID.
JTAG configuration register	64 bits	Allows access to the configuration bus when using the CFG_IN or CFG_OUT instructions.
USERCODE register	32 bits	Captures user-programmable code

Boundary-Scan Register

The test primary data register is the boundary-scan register. Boundary-scan operation is independent of individual IOB configurations. Each IOB, bonded or un-bonded, starts as bidirectional with 3-state control. Later, it can be configured to be an input, output, or 3-state only. Therefore, three data register bits are provided per IOB (Figure 4-22).

When conducting a data register (DR) operation, the DR captures data in a parallel fashion during the CAPTURE-DR state. The data is then shifted out and replaced by new data during the SHIFT-DR state. For each bit of the DR, an update latch is used to hold the input data stable during the next SHIFT-DR state. The data is then latched during the UPDATE-DR state when TCK is Low.

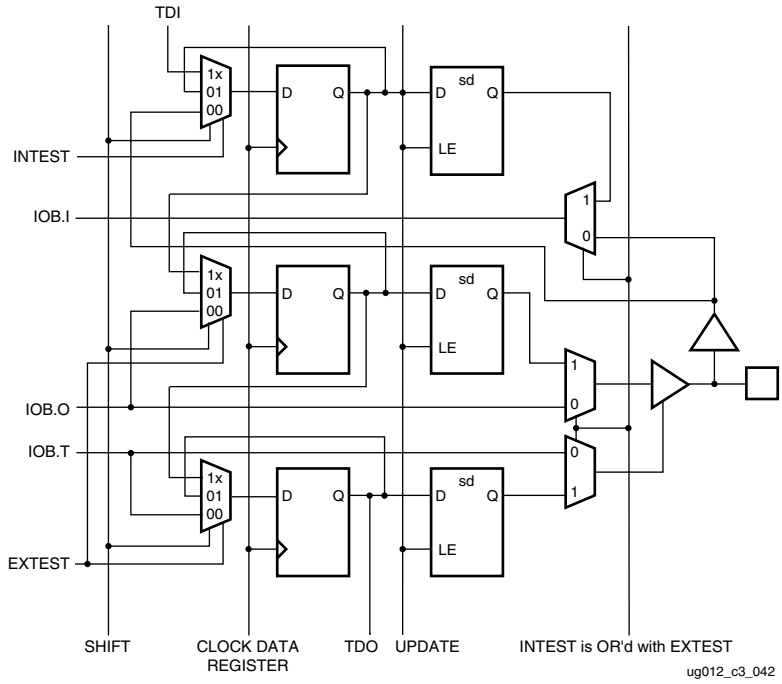


Figure 4-22: **Virtex Series Boundary Scan Logic**

The update latch is opened each time the TAP Controller enters the UPDATE-DR state. Care is necessary when exercising an INTEST or EXTEST to ensure that the proper data has been latched before exercising the command. This is typically accomplished by using the SAMPLE/PRELOAD instruction.

Consider internal pull-up and pull-down resistors when developing test vectors for testing opens and shorts. The boundary-scan mode determines if the IOB has a pull-up resistor. Figure 4-22 is a representation of Virtex-II Boundary-Scan Architecture.

Bit Sequence

The order in each non-TAP IOB is described in this section. The input is first, then the output, and finally the 3-state IOB control. The 3-state IOB control is closest to the TDO. The input-only pins contribute only the input bit to the boundary-scan I/O data register. The bit sequence of the device is obtainable from the "Boundary-Scan Description Language Files" (BSDL files) for the Virtex family. These files can be obtained from the Xilinx software download area. The bit sequence is independent of the design. It always has the same bit order and the same number of bits.

Bypass Register

The other standard data register is the single flip-flop BYPASS register. It passes data serially from the TDI pin to the TDO pin during a bypass instruction. This register is initialized to zero when the TAP controller is in the CAPTURE-DR state.

Instruction Register

The instruction register is a 6-bit register that loads the OPCODE necessary for the Virtex-II boundary-scan instruction set. This register loads the current OPCODE and captures internal device status.

Configuration Register (Boundary-Scan)

The configuration register is a 64-bit register. This register allows access to the configuration bus and readback operations.

Identification Register

Virtex devices have a 32-bit identification register, commonly referred to as the IDCODE register. This register is based upon IEEE Standard 1149.1 and allows easy identification of the part being tested or programmed via boundary scan.

Virtex-II Identification Register

The Virtex-II JTAG ID Code register has the following format.

```
3322 2222222 211111111 110000000000
1098 7654321 098765432 109876543210   <-- bit positions(00 to 31)
vvvv:fffffff:aaaaaaaaa:ccccccccccc1
```

where

```
v is the revision code and
f is the 7-bit family code = 0001000 0x08
a is the number of array rows in the part expressed in 9 bits.

                XC2V40 =   8 =   0x08
                XC2V80 =  16 =   0x010
               XC2V250 =  24 =   0x018
               XC2V500 =  32 =   0x020
```

```
XC2V1000 =   40 =  0x028
XC2V1500 =   48 =  0x030
XC2V2000 =   56 =  0x038
XC2V3000 =   64 =  0x040
XC2V4000 =   80 =  0x050
XC2V6000 =   96 =  0x060
XC2V8000 =  112 =  0x070
```

`c is the company code = 00001001001 = 0x049*`

*Since the last bit of the JTAG IDCODE is always one, the last three hex digits appear as `0x093`.

	vvvv	ffff	fff	a	aaaa	aaaa	cccc	cccc	cccc
		0001	000	0	0001	1000	0000	1001	0011
XC2V250	v	1	0		1	8	0	9	3
XC2V500	v	1	0		2	0	0	9	3

ID Codes assigned to Virtex-II FPGAs are shown in Table 4-14.

Table 4-14: **Virtex-II Device ID Codes**

FPGA	IDCODE
XC2V40	<v>01008093
XC2V80	<v>01010093
XC2V250	<v>01018093
XC2V500	<v>01020093
XC2V000	<v>01028093
XC2V1500	<v>01030093
XC2V2000	<v>01038093
XC2V3000	<v>01040093
XC2V4000	<v>01050093
XC2V6000	<v>01060093
XC2V8000	<v>01070093

Notes:
1. The <v> in the IDCODE stands for the device's revision code (in hex).

USERCODE Register

USERCODE is supported in the Virtex family as well. This register allows a user to specify a design-specific identification code. The USERCODE can be programmed into the device and read back for verification at a later time. The USERCODE is embedded into the bitstream during bitstream generation (BitGen **`-g UserID`** option) and is valid only after configuration.

USER1, USER2 Registers

The USER1 and USER2 registers are only valid after configuration. These two registers must be defined by the user within the design. These registers can be accessed after they are defined by the TAP pins.

The BSCAN_VIRTEX2 library macro is required when creating these registers. This symbol is only required for driving internal scan chains (USER1 and USER2). The BSCAN_VIRTEX2 macro provides two user pins (SEL1 and SEL2) for determining usage of USER1 or USER2 instructions respectively. For these instructions, two corresponding pins (TDO1 and TDO2) allow user scan data to be shifted out of TDO. In addition, there are individual clock pins (DRCK1 and DRCK2) for each user register. There is a common input pin (TDI) and shared output pins that represent the state of the TAP controller (RESET, SHIFT, and UPDATE). Unlike earlier FPGA families that required the BSCAN macro to dedicate TAP pins for boundary scan, Virtex-II TAP pins are dedicated and do not require the BSCAN_VIRTEX2 macro for normal boundary-scan instructions or operations.

Note that these are user-defined registers. The example (Figure 4-23) is one of many implementations. For HDL, the BSCAN_VIRTEX2 macro needs to be instantiated in the design.

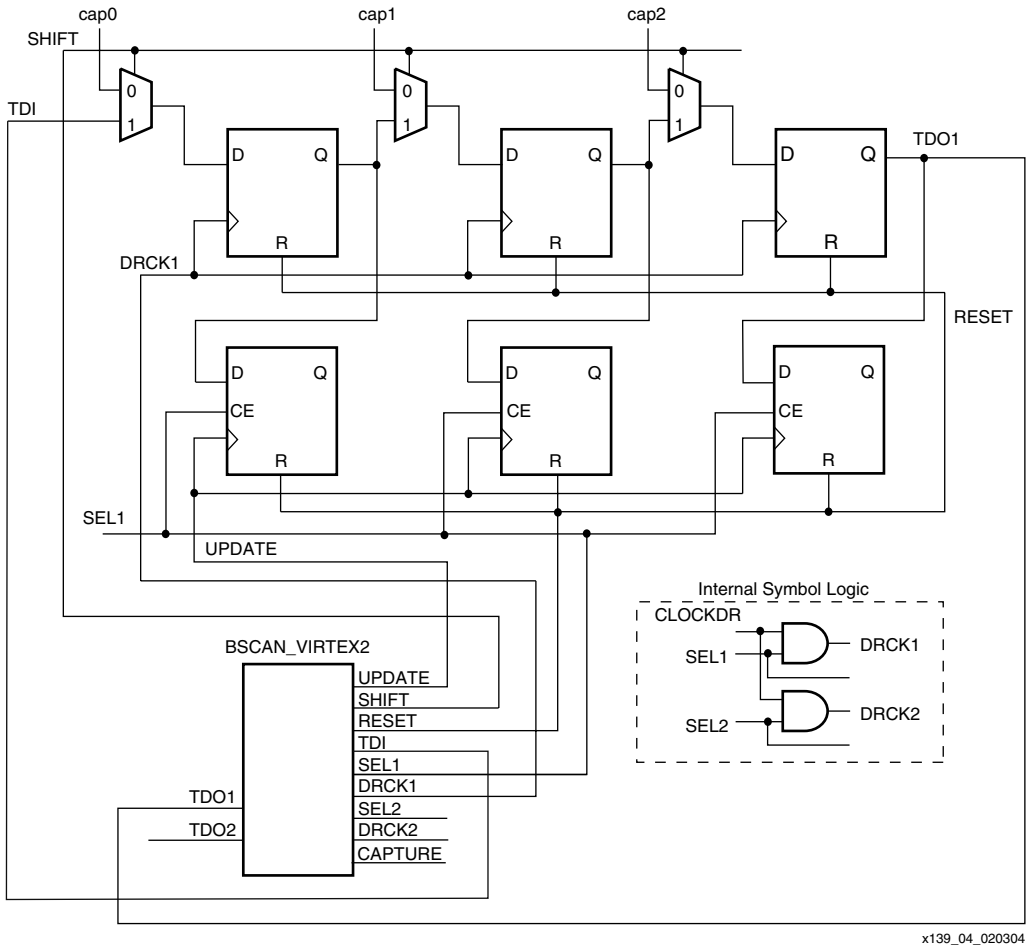


Figure 4-23: **BSCAN_VIRTEX2 (Example Usage)**

Using Boundary Scan in Virtex-II Devices

Some of the most commonly requested timing parameters are illustrated in Figure 4-24. For corresponding specifications, see the **Virtex-II Platform FPGA Data Sheet**.

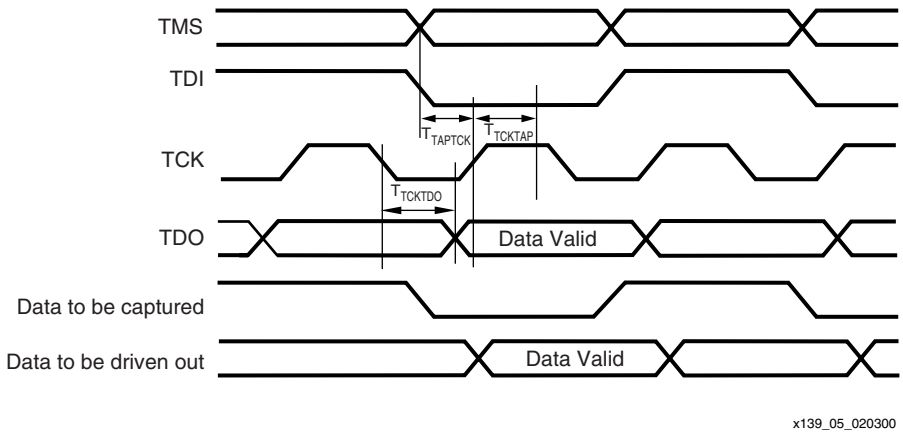


Figure 4-24: **Virtex-II Boundary Scan Port Timing Waveforms**

For further information on the Startup sequence, bitstream, and internal configuration registers referenced here, refer to "Readback" on page 342.

Configuring Through Boundary-Scan

One of the most common boundary-scan vendor-specific instructions is the Configure instruction. An individual Virtex-II device is configured via JTAG on power-up using TAP. If the Virtex-II device is configured on power-up, it is advisable to tie the mode pins to the boundary-scan configuration mode settings; 101 (M2 = 1, M1 = 0, M0 = 1).

Configuration flow for Virtex-II device configuration with JTAG is shown in Figure 4-25 on page 306. The sections that follow describe how the Virtex-II device can be configured as a single device via boundary-scan or as part of a multiple-device scan chain.

A configured device can be reconfigured by toggling the TAP and entering a CFG_IN instruction after pulsing the PROG_B pin or issuing the shut-down sequence. (Refer to "Power Up" on page 273). For additional details on power-up or the start-up sequence in Virtex-II devices, see "Device Startup" on page 275.

Customers who wish to implement the Virtex-II JTAG configuration algorithm are encouraged to use the SVF-based flow provided in Xilinx Application Note 058 (available on **www.xilinx.com**).

Single Device Configuration

Configure a Virtex-II part as a single device via boundary-scan operations as follows. Ensure that the bitstream is generated with the JTAG clock option.

```
bitgen -g startupclk:jtagclk <designName>.ncd
```

Also, when using iMPACT software, verify that the most current version is being used.

Table 4-15 on page 307 describes the TAP controller commands required to configure a Virtex-II device. Refer to Figure 4-21 on page 299 for TAP controller states. These TAP controller commands are issued automatically if configuring the part with the iMPACT software.

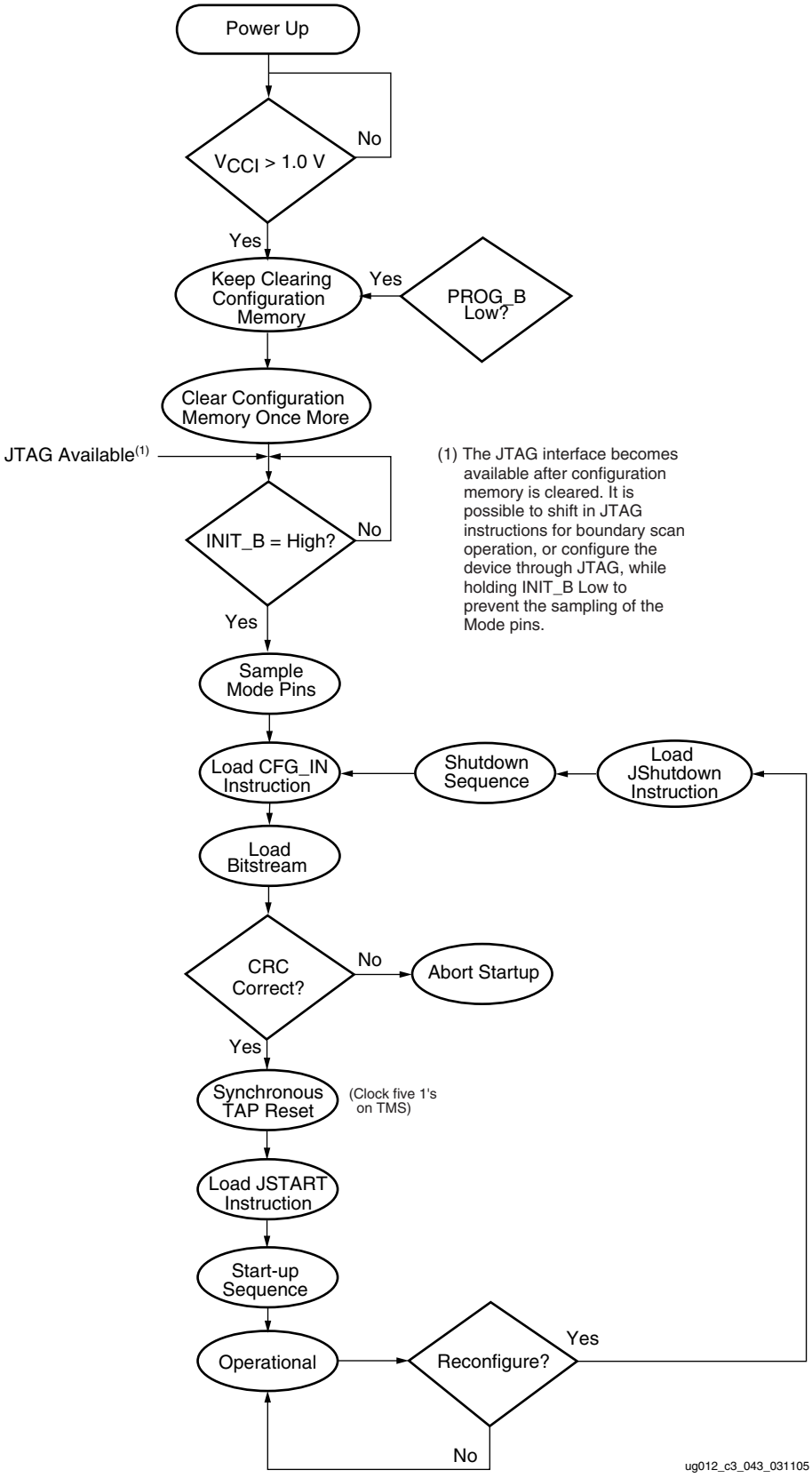


Figure 4-25: **Device Configuration Flow Diagram**

Table 4-15: **Single Device Configuration Sequence**

TAP Controller Step Description		Set & Hold		# of Clocks
		TDI	TMS	TCK
1.	On power-up, place a logic “one” on the TMS and clock the TCK five times. This ensures starting in the TLR (Test-Logic-Reset) state.	X	1	5
2.	Move into the RTI state.	X	0	1
3.	Move into the SELECT-IR state.	X	1	2
4.	Enter the SHIFT-IR state.	X	0	2
5.	Start loading the CFG_IN instruction, LSB first:	00101	0	5
6.	Load the MSB of CFG_IN instruction when exiting the SHIFT-IR state, as defined in the IEEE standard.	0	1	1
7.	Enter the SELECT-DR state.	X	1	2
8.	Enter the SHIFT-DR state.	X	0	2
9.	Shift in the Virtex-II bitstream. Bit_n (MSB) is the first bit in the bitstream(1).	$bit_1 \ldots bit_n$	0	(bits in bitstream)-1
10.	Shift in the last bit of the bitstream. Bit_0 (LSB) shifts on the transition to EXIT1-DR.	bit_0	1	1
11.	Enter UPDATE-DR state.	X	1	1
12.	Reset TAP by clocking five 1’s on TMS	X	1	5
13.	Enter the SELECT-IR state.	X	1	2
14.	Move to the SHIFT-IR state.	X	0	2
15.	Start loading the JSTART instruction. The JSTART instruction initializes the startup sequence.	01100	0	5
16.	Load the last bit of the JSTART instruction.	0	1	1
17.	Move to the UPDATE-IR state.	X	1	1
18.	Move to RTI and clock the STARTUP sequence by applying a minimum of 12 clock cycles to the TCK.	X	0	≥12
19.	Move to the TLR state. The device is now functional.	X	1	3

Notes:

1. In the Configuration Register, data is shifted in from the right (TDI) most significant bit, to the left, least significant bit (D0 ... D7).

Multiple Device Configuration

It is possible to configure multiple Virtex-II devices in a chain (Figure 4-26). The devices in the JTAG chain are configured one at a time. The multiple device configuration steps can be applied to any size chain. Ensure the bitstream is generated with the JTAG clock option:

```
bitgen -g startupclk:jtagclk <designName>.ncd
```

Refer to the State Diagram in Figure 4-21 on page 299 for the following TAP controller steps.

1. On power-up, place a logic 1 on the TMS and clock the TCK 5 times. This ensures starting in the TLR (Test-Logic-Reset) state.
2. Load the CFG_IN instruction into the target device (and BYPASS in all other devices). Go through RTI (RUN-TEST/IDLE).
3. Load in the configuration bitstream per step 7 through step 11 in Table 4-15.

4. Repeat step 2 and step 3 for each device.
5. Reset all TAPs by clocking five 1's on TMS.
6. Load the JSTART command into all devices.
7. Go to RTI and clock TCK 12 times.

All devices are active at this point.

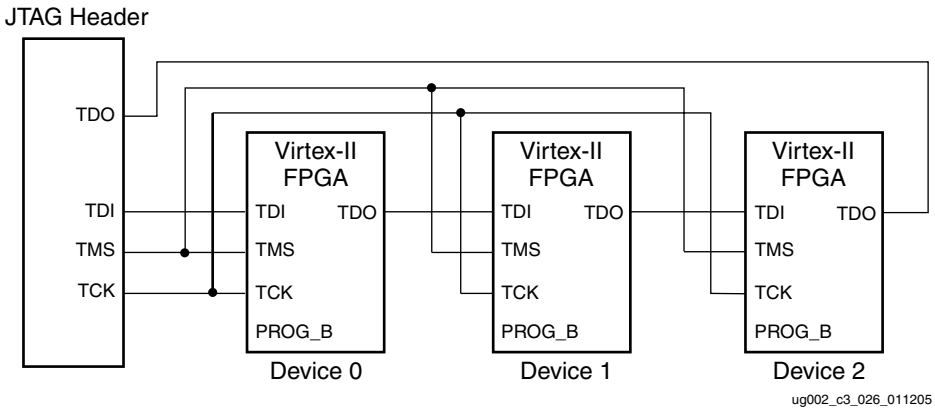


Figure 4-26: **Boundary Scan Chain of Devices**

Notes:

1. PROG_B pin should be deasserted during JTAG operation.

Reconfiguring Through Boundary Scan

The ability of Virtex-II devices to perform partial reconfiguration is the reason that the configuration memory is not cleared when reconfiguring the device. When reconfiguring a chain of devices, refer to step 3 in Table 4-15. There are two methods to reconfigure Virtex-II devices without possible internal contention. The first method is to pulse the PROG_B pin which resets the internal configuration memory. The alternate method is to perform a shutdown sequence, placing the device in a safe state. The following shutdown sequence includes using internal registers. (For details on internal registers, refer to "Readback" on page 342.)

1. Load the CFG_IN instruction.
2. In SHIFT-DR state, load the synchronization word followed by the Reset CRC Register (RCRC) command.

```
1111 1111 1111 1111 1111 1111 1111 1111 → Pad word
1010 1010 1001 1001 0101 0101 0110 0110 → Synchronization word
0011 0000 0000 0000 1000 0000 0000 0001 → Header: Write to CMD register
0000 0000 0000 0000 0000 0000 0000 0111 → RCRC command
0000 0000 0000 0000 0000 0000 0000 0000 → flush pipe
0000 0000 0000 0000 0000 0000 0000 0000 → flush pipe
```

3. Load JSHUTDOWN.
4. Go to RTI and clock TCK at least 12 times to clock the shutdown sequence.
5. Proceed to SHIFT-IR state and load the CFG_IN instruction again.
6. Go to SHIFT-DR state and load the configuration bits. Make sure the configuration bits contain AGHIGH command, which asserts the global signal GHIGH_B. This prevents contention while writing configuration data.

```
0011 0000 0000 0000 1000 0000 0000 0001 → Header: Write to CMD
0000 0000 0000 0000 0000 0000 0000 1000 → AGHIGH command asserts GHIGH_B
```

7. When all configuration bits have been loaded, reset the TAP by clocking five 1's on TMS.

8. Go to SHIFT-IR state and load the JSTART instruction.
9. Go to RTI and clock TCK at least 12 times to clock the startup sequence.
10. Go to TLR state to complete the reconfiguration process.
11. Reset the DCMs after reconfiguration through JTAG.

Debugging Configuration

To verify successful configuration, there are several options. Some of the most helpful verification steps include using TAP pins and the readback command. Using the Virtex-II TAP controller and status pins is discussed first.

When using TAP controller pins, TDO is driven only in the SHIFT-DR and SHIFT-IR state. If the output of the TDO can be changed via an external pull-up resistor, the TAP is not in the SHIFT-IR or SHIFT-DR state. If the TAP can be controlled precisely, use this to test the application.

In JTAG configuration, the status pin (DONE) functions the same as in the other configuration modes. The DONE pin can be monitored to determine if a bitstream has been completely loaded into the device. If DONE is Low, the entire bitstream has not been sent or the start-up sequence is not finished. If DONE is High, the entire bitstream has been received correctly. The INIT_B pin functions similar to a normal INIT_B but does not indicate a configuration error in boundary-scan configuration.

In addition to external pin monitoring, an internal test can be conducted. The second method includes the following steps to capture the internal device status register contents:

1. Move the TAP to TLR state.
2. Go to SHIFT-IR state and load in the CFG_IN instruction.
3. Go to SHIFT-DR state and shift in the following 64-bit pattern with the MSB (left-most bit), shifted in first.

```
1111 1111 1111 1111 1111 1111 1111 1111  → Pad word
1010 1010 1001 1001 0101 0101 0110 0110  → Synchronization word
0010 1000 0000 0000 1110 0000 0000 0010  → Read STAT Register(1)
0000 0000 0000 0000 0000 0000 0000 0000  → flush pipe
0000 0000 0000 0000 0000 0000 0000 0000  → flush pipe
0000 0000 0000 0000 0000 0000 0000 0000  → flush pipe
```

Notes:

1. Since an odd number of words (1) is sent to the device for this JTAG read operation, it must end with exactly three pad words. This is because all JTAG read command sequences must consist of an even number of words (see "Packet Processor" on page 318 for details).

4. After shifting in the pattern, load the CFG_OUT instruction in the SHIFT-IR state.
5. Move to SHIFT-DR state and clock TCK 32 times while reading TDO. The data seen on TDO is the content of the status register. The last bit out is a one if a CRC error occurred. If successful, it should read as follows.

```
0000 0000 0000 0000 0001 1MMM 1110 11101(1,2)
```

Notes:

1. *MMM* is the mode pins value.
2. Assuming that the device is in normal operation mode.
3. Refer to "Status Register (STAT)" on page 324 for a detailed explanation of the bits in the Status Register.

6. Go to SHIFT-IR state and load the CFG_IN instruction again.
7. Move to SHIFT-DR state and shift in the following bit pattern:

```
0011 0000 0000 0000 1000 0000 0000 0001 → Header: Write to CMD register
0000 0000 0000 0000 0000 0000 0000 0111  → RCRC command
0000 0000 0000 0000 0000 0000 0000 0000  → flush pipe
0000 0000 0000 0000 0000 0000 0000 0000  → flush pipe
```

8. Put the TAP in TLR state when finished.

The device status register also gives the status of the DONE and INIT_B signals. For information on the status register, refer to Figure 4-38 on page 324.

Boundary-Scan for Virtex-II Devices Using IEEE Standard 1532

ISC Modal States

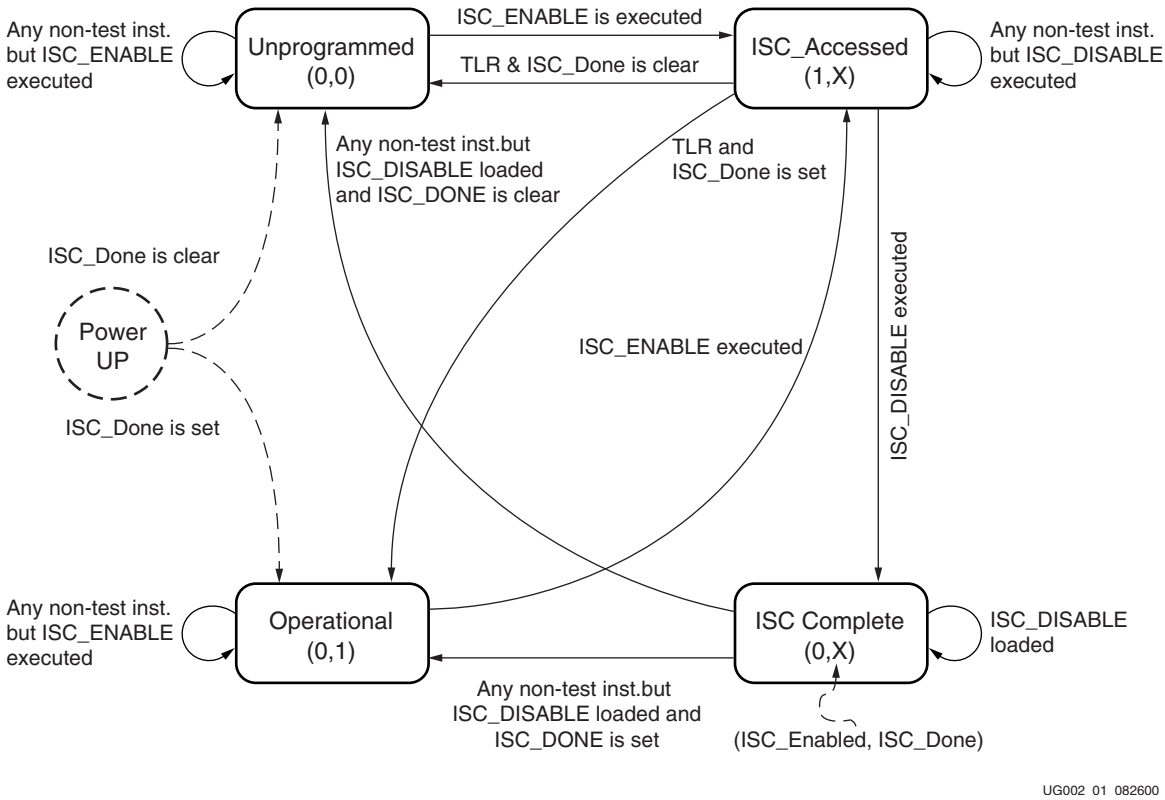


Figure 4-27: **ISC Modal States**

Once the device is powered up, it goes to an Unprogrammed state. The I/Os are all either 3-stated or pulled up. When ISC_ENABLE is successfully executed, the ISC_Enabled signal is asserted, and the device moves to ISC_Accessed state. When the device moves to ISC_Accessed state from Operational state, the shutdown sequence is executed. The I/Os are all either 3-stated or pulled up.

The StartUp sequence is executed when in the ISC_Accessed state. At the end of the StartUp Sequence, ISC_Enabled is cleared and the device moves to ISC_Complete. The minimum clock cycle requirement is the number of clock cycles required to complete the StartUp sequence. At the completion of the minimum required clock cycles, ISC_Enabled is deasserted.

Whether the StartUp sequence is successful or not is determined by CRC or configuration error status from the configuration processor. If the startup is completed, ISC_Done is asserted; otherwise, ISC_Done stays Low. The I/Os are either 3-stated or pulled up.

When ISC_Done is set in ISC_Complete state, the device moves to the Operational state. Otherwise, if ISC_Done is clear, the device moves to an Unprogrammed state. However, if the TAP controller goes to TLR state while the device is in ISC_Accessed state and if ISC_Done is set, then the device moves to the Operational state. However, the I/O is not active yet because the Startup sequence has not been performed. The Startup sequence has to be performed in the Operational state to bring the I/O active.

Clocking Startup and Shutdown Sequence (JTAG Version)

There are three clock sources for Startup and Shutdown sequence, CCLK, UserCLK, and JTAGCLK. Clock selection is set by BitGen. The Startup sequence is executed in ISC_Accessed state. When it is clocked by JTAGCLK, the Startup sequence receives the JTAGCLK in TAP Run/Test Idle state while ISC_DISABLE is the current JTAG instruction. The number of clock cycles in Run/Test Idle state for successful completion of ISC_DISABLE is determined by the number of clock cycles needed to complete the Startup sequence.

When UserCLK or CCLK is used to clock the Startup sequence, the user should know how many JTAGCLK cycles should be spent in Run/Test Idle to successfully complete the Startup sequence.

The Shutdown sequence is executed when the device transitions from an Operational to ISC_Accessed state. Shutdown is done while executing the ISC_ENABLE instruction. When the Shutdown sequence is clocked using JTAGCLK, the clock is supplied in the Run/Test Idle state of the ISC_ENABLE instruction. The number of clock cycles in Run/Test Idle is determined by the number of clock cycles needed to complete the Shutdown sequence.

When the Shutdown sequence is clocked by CCLK or UserCLK, the user is responsible for knowing how many JTAGCLK cycles in Run/Test Idle are needed to complete the Shutdown sequence.

Notes:

1. It has been decided that when configuring the device through JTAG, the startup and shutdown clock should come from TCK, regardless of the selection in BitGen.
2. In IEEE 1532 configuration mode, Startup and Shutdown clock source is always TCK.

Configuration Flows Using JTAG

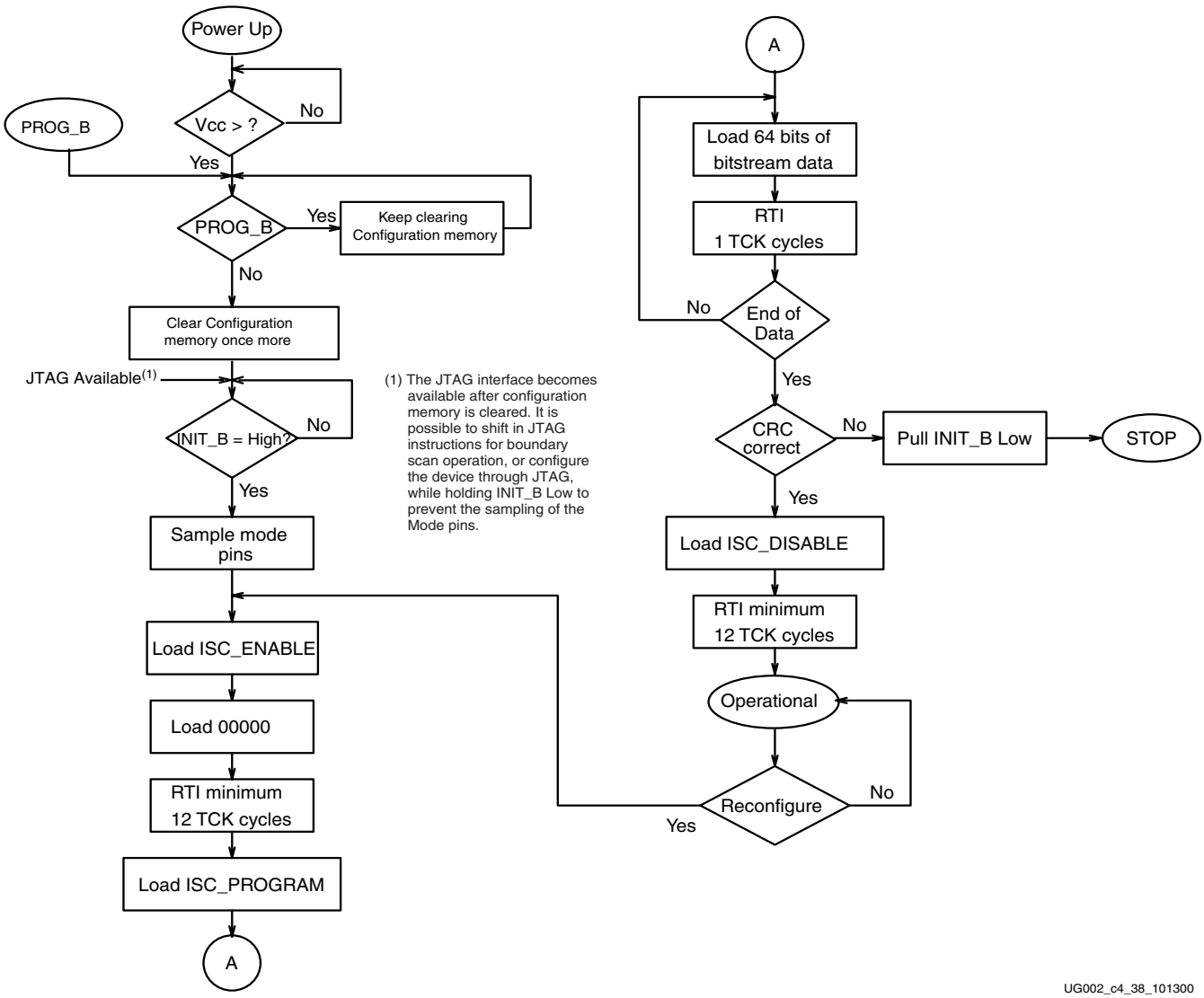


Figure 4-28: **IEEE 1532 Configuration Flow**

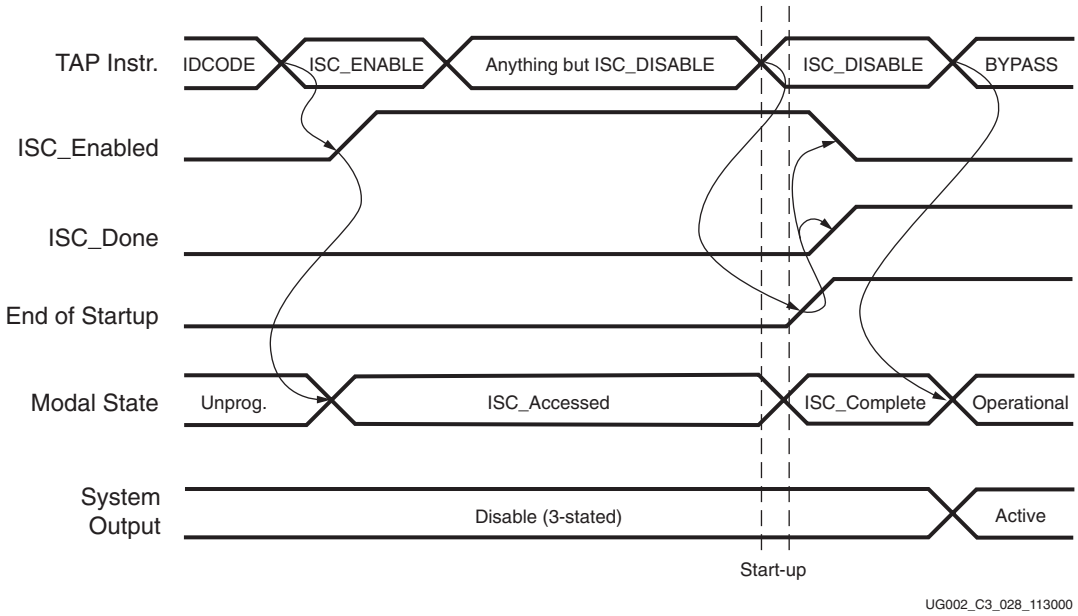


Figure 4-29: **Signal Diagram for Successful First Time ISC Configuration**

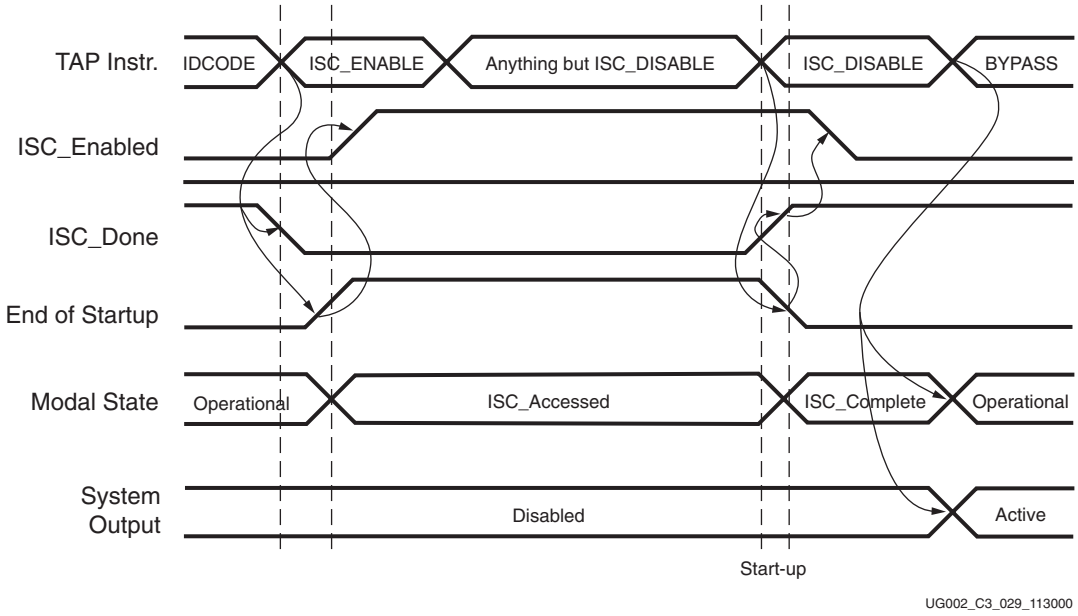


Figure 4-30: **Signal Diagram for Successful ISC Partial and Full Reconfiguration**

Configuration Details

All user programmable features inside a Virtex-II device are controlled by memory cells that are volatile and must be configured on power-up. These memory cells are known as the configuration memory, and define the LUT equations, signal routing, IOB voltage standards, and all other aspects of the user design.

To program configuration memory, instructions for the configuration control logic and data for the configuration memory are provided in the form of a bitstream, which is delivered to the device through one of the JTAG, SelectMAP, or Serial configuration interfaces. The composition of the bitstream is largely independent of the configuration method; however, certain operations such as readback can only be performed via the SelectMAP and JTAG interfaces.

Configuration Memory: Columns and Frames

Virtex-II configuration memory is arranged in vertical frames that are one bit wide and stretch from the top edge of the device to the bottom. These frames are the smallest addressable segments of the Virtex-II configuration memory space; therefore, all operations must act on whole configuration frames. Configuration memory frames do not directly map to any single piece of hardware; rather, they configure a narrow vertical slice of many physical resources.

The length of a Virtex-II frame depends on the size of the device, as shown in Table 4-16.

Table 4-16: **Virtex-II Frame Count, Frame Length, and Bitstream Size**

Device	Frames	Frame Length (32-bit Words)	Configuration Bits(1)	Default Bitstream Size(2)
XC2V40	404	26	336,128	338,976
XC2V80	404	46	594,688	598,816
XC2V250	752	66	1,588,224	1,593,632
XC2V500	928	86	2,553,856	2,560,544
XC2V1000	1104	106	3,744,768	4,082,592
XC2V1500	1280	126	5,160,960	5,170,208
XC2V2000	1456	146	6,802,432	6,812,960
XC2V3000	1804	166	9,582,848	10,494,368
XC2V4000	2156	206	14,212,352	15,659,936
XC2V6000	2508	246	19,742,976	21,849,504
XC2V8000	2860	286	26,174,720	26,194,208

Notes:

1. Number of configuration bits = (Number of Frames) x (Frame Length)
2. Default bitstream size = (Configuration bits) + Overhead. Stepping 1 bitstream with bitgen `-g FreezeDCI:No` is assumed. Other bitgen options can affect bitstream size; see Table 4-2 on page 271.

Virtex-II devices do not "expect" a bitstream of any particular size. The length of the bitstream can vary according to device Stepping Level and BitGen settings.

Configuration frames are grouped into six column types that correspond roughly to physical device resources. All Virtex-II devices have the same configuration column types: IOB, IOI, CLB, GCLK, BlockRAM, and BlockRAM Interconnect. Figure 4-31 shows the relationship between physical device resources and configuration memory.

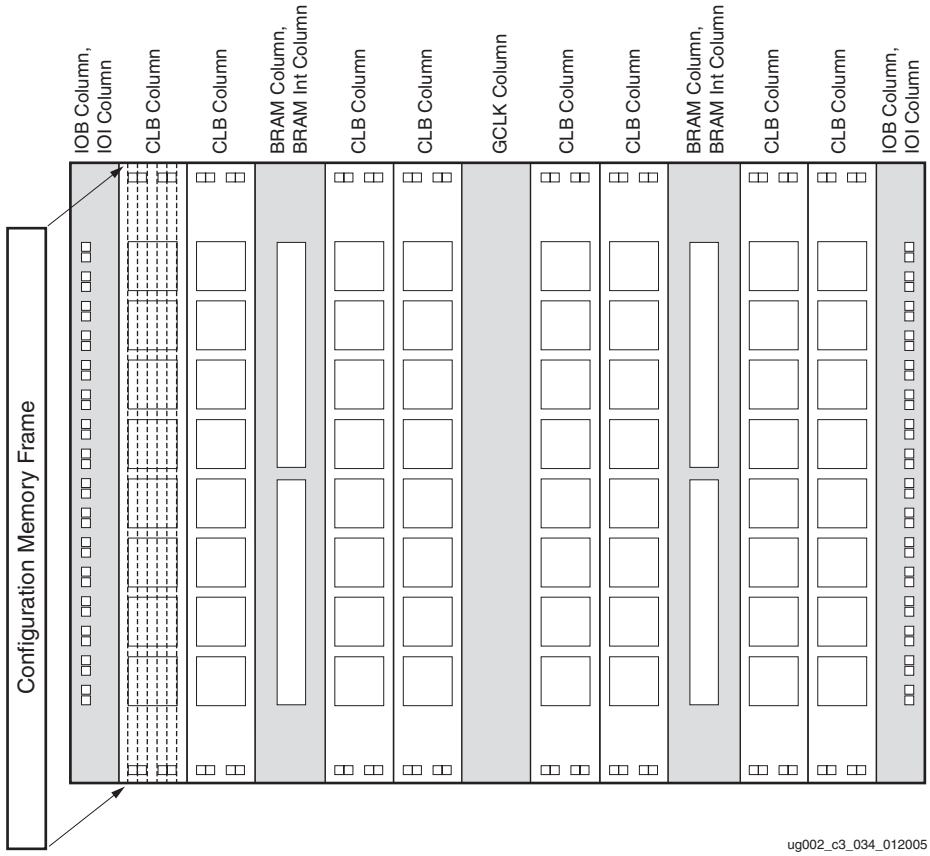


Figure 4-31: **Virtex-II Device with Configuration Columns and Frames Overlaid**

All Virtex-II devices have the same number of IOB, IOI, and GCLK columns. However,the number of CLB, BRAM, and BRAM Interconnect columns varies by device size. The number of frames per column type remains constant for all devices. Table 4-17 shows the number of columns per device and the number of frames per column.

Table 4-17: **Virtex-II Column Types and Frame Counts**

Column Type: →	IOB		IOI		CLB		BRAM		BRAM Interconnect		GCLK	
Device: ↓	Columns per Device:	Frames per Column:	Columns per Device:	Frames per Column:	Columns per Device:	Frames per Column:	Columns per Device:	Frames: per Column	Columns per Device:	Frames per Column:	Columns per Device:	Frames per Column:
XC2V40	2	4	2	22	8	22	2	64	2	22	1	4
XC2V80	2	4	2	22	8	22	2	64	2	22	1	4
XC2V250	2	4	2	22	16	22	4	64	4	22	1	4
XC2V500	2	4	2	22	24	22	4	64	4	22	1	4
XC2V1000	2	4	2	22	32	22	4	64	4	22	1	4
XC2V1500	2	4	2	22	40	22	4	64	4	22	1	4
XC2V2000	2	4	2	22	48	22	4	64	4	22	1	4
XC2V3000	2	4	2	22	56	22	6	64	6	22	1	4
XC2V4000	2	4	2	22	72	22	6	64	6	22	1	4
XC2V6000	2	4	2	22	88	22	6	64	6	22	1	4
XC2V8000	2	4	2	22	104	22	6	64	6	22	1	4

IOB Columns

IOB columns configure the voltage standard for the I/Os on the left and right edges of the device. IOBs on the top and bottom edges of the device are configured by the CLB Columns with which they are vertically aligned. There are two IOB columns per device.

IOI Columns

IOI columns configure the IOB registers, multiplexers, and 3-state buffers in the IOBs on the left and right edges of the device. IOBs on the top and bottom edges of the device are configured by the CLB columns with which they are vertically aligned. There are two IOI columns per device.

CLB Columns

The CLB columns program the configurable logic blocks, routing, and most interconnect. IOBs on the top and bottom edges of the device are also programmed by CLB configuration columns. The number of CLB configuration columns matches the number of physical CLB columns in the device.

BlockRAM Columns

BlockRAM configuration columns program only the BlockRAM user memory space. BlockRAMs are accessed by the configuration control logic in the same way that user designs access BlockRAMs: via the address and data pins that are available on the BlockRAM primitive. Consequently, the user design cannot access BlockRAM while BlockRAM columns are being addressed by the configuration logic. For this reason, active reconfiguration and readback on BlockRAMs should not be attempted. The number of BlockRAM configuration columns matches the number of physical BlockRAM columns in the device.

BlockRAM Interconnect Columns

BlockRAM Interconnect columns program all other BlockRAM and multiplier features, including aspect ratios. The number of BlockRAM Interconnect configuration columns matches the number of physical BlockRAM columns in the device.

GCLK Column

The global clock column configures most global clock resources, including clock buffers and DCMs. There is one global clock column per device.

Configuration Memory Addressing

Each configuration frame has a unique 32-bit address that is composed of a block address (BA), a major address (MJA), a minor address (MNA), and a byte number, shown in Figure 4-32. The major address identifies a specific column within a block, and the minor address identifies a specific frame within a column.

The byte number is used only by the Virtex-II configuration logic (the frame length counter, described later). Users cannot address a byte within a frame, and must always set the nine least significant bits of the frame address to '0'.

					BA		MJA								MNA								Byte Number								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0

Figure 4-32: **Virtex-II Frame Address Composition**

Virtex-II configuration memory is divided into three independently addressable blocks:

- Block Address 0 (BA 00) contains all GCLK, IOB, IOI, and CLB configuration columns
- Block Address 1 (BA 01) contains all BRAM columns
- Block Address 2 (BA 10) contains all BRAM Interconnect columns

Figure 4-33 illustrates the Virtex-II configuration memory map. The MJA for each block type is a function of the device size, and is determined by the number of CLB columns (n) and the number of BlockRAM columns (m) in a device. During configuration, frames are programmed in order of increasing block, major, and minor address (i.e., from left to right in Figure 4-33).

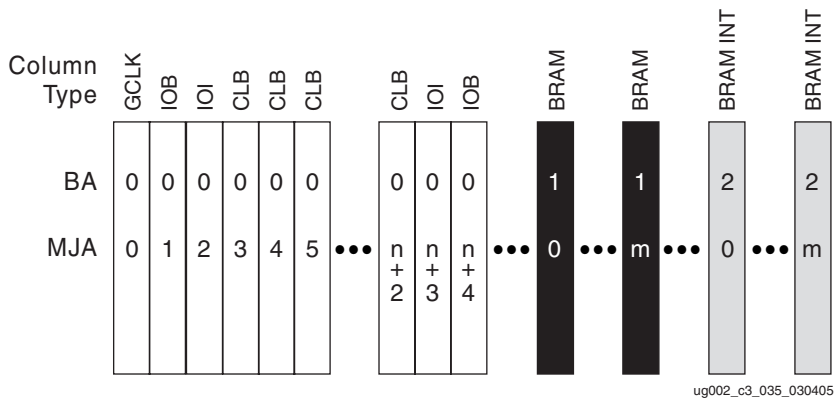


Figure 4-33: **Column-Level (MJA) Configuration Memory Map**

Notes:

1. n = number of device CLB columns; m = number of device BRAM/BRAM Int columns. Refer to Table 4-17 on page 315 for these numbers.
2. Virtex-II Pro configuration control logic allows access to nonexistent memory locations; therefore, attempts to write to invalid addresses have no effect, and attempts to read from invalid addresses yield a static value. The configuration logic does not issue an error of any kind when attempting to perform operations on invalid memory addresses.

Bitstream Packets

The Virtex-II bitstream is composed of a 32-bit synchronization word (0xAA995566) and numerous data packets. The purpose of the synchronization word is to align the device configuration logic with the beginning of the first packet in the bitstream. Each packet targets a specific configuration register to set configuration options, program configuration memory, or toggle internal signals.

Bitstream data packets consist of a 32-bit header and a body of variable length. There are two packet types:

- Type 1, used for smaller packets (up to $2^{11} - 1$ words)
- Type 2, used for larger packets (up to $2^{27} - 1$ words)

Figure 4-34 and Figure 4-35 describe the Type 1 and Type 2 packet headers.

Type			W R	R D	Register Address													RSVD		Word Count										
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	x	x	x	x	x	x	x	x	x	x	x

Figure 4-34: **Type 1 Packet Header**

Type			W R	R D	Word Count																									
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 4-35: **Type 2 Packet Header**

Both packet types specify read or write access and the size of the packet in words. However, only Type 1 packet headers can specify a register address. The register set by an earlier Type 1 packet header remains active when a Type 2 packet header is processed; therefore, all Type 2 packets appear as a part of a "Type 1, Type 2" combination. Table 4-18 gives an example of a Type 1, Type 2 packet combination. Note that the register address is entirely different than the frame address described earlier.

Table 4-18: **Type 1, Type 2 Packet Combination**

Configuration Data	Explanation
0x30004000	Type 1 Packet Header: Write 0 words to FDRI
0x50002922	Type 2 Packet Header: Write 10530 words to the active register (FDRI)
0x--------	10530 data words

Configuration Control Logic

The Virtex-II configuration logic consists of a packet processor, a set of registers, and global signals that are controlled by the configuration registers. The packet processor controls the flow of data from the configuration interface (SelectMAP, JTAG, or Serial) to the appropriate register, while the registers control all other aspects of configuration. This section describes how bitstream packets are processed by the configuration control logic.

Packet Processor

After power-on, the packet processor waits for the synchronization word to appear on the configuration interface before interpreting any data. Once the device is synchronized, the packet processor waits for a valid packet header; when one is received it drives incoming data into the targeted configuration register until the word count specified by the packet header reaches zero. At the end of a packet the processor waits for a new packet header.

All packet headers pass through a 64-bit buffer before reaching the packet processor; the packet processor itself interprets all commands in 32-bit words. To flush the last command in a sequence from the packet buffer, a sequence of configuration commands must end with least four 32-bit NOOP commands. Typically, this is found at the end of a bitstream, as shown in Table 4-19. JTAG read operations must consist of an even number of words in the command sequence.

This means that read command sequences must end with exactly two or exactly three NOOPs, depending on the number of words sent prior to the NOOP commands. Do not include the synchronization word or anything prior to it in this count, and do not use any number of NOOPs other than two or three to finish sending a JTAG read command sequence. See "Configuration Register Read Procedure (JTAG)" on page 346 and "Configuration Memory Read Procedure (1149.1 JTAG)" on page 348 for examples.

Table 4-19: **NOOP Commands at the End of the Virtex-II Bitstream**

Configuration Data	Explanation
30000001	Type 1 write 1 words to CRC
00009A32	CRC = 0x00009A32
30008001	Type 1 write 1 words to CMD
0000000D	DESYNCH command
20000000	Type 1 NOOP word 0
20000000	Type 1 NOOP word 1

Configuration Registers

All configuration operations are executed by reading or writing to the configuration registers. Table 4-20 summarizes these registers; a detailed explanation of each register follows.

Table 4-20: **Virtex-II Configuration Registers**

Register Name	Read	Write	Address	Description
CRC	Y	Y	00000	CRC register
FAR	Y	Y	00001	Frame address register
FDRI	N	Y	00010	Frame data input register (write configuration data)
FDRO	Y	N	00011	Frame data output register (readback configuration data)
CMD	Y	Y	00100	Command register
CTL	Y	Y	00101	Control register
MASK	Y	Y	00110	Masking register for CTL
STAT	Y	N	00111	Status register
LOUT	N	Y	01000	Legacy output register (DOUT for daisy chain)
COR	Y	Y	01001	Configuration option register
MFWR	N	Y	01010	Multiple frame write register
FLR	Y	Y	01011	Frame length register
KEY	N	Y	01100	Initial key address register
CBC	N	Y	01101	Initial CBC value register
IDCODE	Y	Y	01110	Device ID register

CRC Register (CRC)

Configuration data integrity is assured by 16-bit Cyclic Redundancy Checks (CRCs) that must be performed prior to device startup. As configuration data are written to any register other than LOUT, a CRC value is calculated using the register data and address bits. The CRC check is performed after a write to the CRC register by comparing the calculated value to the register value. If the two values differ, the configuration logic goes into the ERROR mode, indicated by driving the INIT_B pin Low and writing to the CRC_ERROR bit in the STATUS register.

CRC checks are performed in two ways:

- Explicit writes to the CRC register

- The "AutoCRC" feature

Explicit CRC checks are performed by writing a pre-calculated CRC value into the CRC register in a distinct configuration packet. Table 4-21 shows the command and data sequence for an explicit CRC check.

Table 4-21: **Example Packet forVirtex-II CRC Check**

Configuration Data	Explanation
30000001	Type 1 write 1 words to CRC
00009A32	CRC = 0x00009A32

AutoCRC is an implicit CRC write that occurs at the end of a write to the Frame Data Input Register (FDRI). The last 32-bit word in a write packet to the FDRI register is automatically interpreted as a CRC value. The AutoCRC word is not included in the packet word count. Note that if the entire bitstream is written in a single Type 2 packet, only one AutoCRC check occurs. Table 4-22 shows an AutoCRC check.

Table 4-22: **Example Packets for Frame Write with AutoCRC**

Configuration Data	Explanation
30008001	Type 1 Packet Header: Write 1 word to CMD
00000001	Packet Data: WCFG command
30002001	Type 1 Packet Header: Write 1 word to FAR
02000000	Packet Data: Frame Address = 0x020000000
3000401A	Type 1 Packet Header: Write 26 words to FDRI
00000000	Packet Data: word 1
. . .	
00000000	Packet Data: word 26
0000A53B	AutoCRC word

CRC checking is disabled by setting the **`-g CRC:Disable`** option in BitGen. This option sets the CRC_DISABLE bit in the Configuration Options Register (COR) to '1'. When CRC checking is disabled, AutoCRC and explicit CRC checks still appear, although they use the default CRC value of (0x0000DEFC), shown in Table 4-23.

Table 4-23: **Example Packets for CRC/AutoCRC Checks with Checking Disabled**

Configuration Data	Explanation
30004000	Type 1 write 0 words to FDRI
50002922	Type 2 write 10530 words to FDRI
00100000	Packet data write FDRI word 0
. . .	
00000000	Packet data write FDRI word 10529
0000DEFC	Auto CRC word
. . .	
30000001	Type 1 write 1 words to CRC
0000DEFC	CRC = 0x0000DEFC

Frame Address Register (FAR)

Configuration frames are addressed by the value in the Frame Address Register (FAR). Writes to the FDRI and reads from the FDRO act on the address specified in the FAR. Each time the FAR is updated with a new value, the command in the Command Register (CMD) is executed.

The FAR is automatically incremented during writes to FDRI and reads from FDRO when a number of words equal to the value in the FLR have been shifted in to FDRI or out of FDRO. Note that only the starting frame address is written before an FDRI write ("Configuration" on page 331). The frame address composition is given in Figure 4-32 on page 316.

Frame Data Input Register (FDRI) and Frame Data Output Register (FDRO)

Configuration data frames are written to memory by shifting frame data into the Frame Data Input Register. Data written to the FDRI is placed in the configuration memory address specified by the FAR. Prior to writing to the FDRI, the CMD register must be loaded with the WCFG command and the correct Device ID must have been written to the IDCODE register.

The write operation is pipelined through the frame buffer such that the first frame of data is written to the configuration memory while the second frame is being shifted through the frame buffer. For this reason, the FDRI must be flushed with one frame of pad data at the end of a write to configuration memory so that the last data frame is loaded. Therefore, the smallest FDRI write operation requires two frames of data: the frame to be loaded into configuration memory, followed by a frame of pad data to flush the first out of the FDRI (see Figure 4-36). The configuration process is explained in detail in the Configuration section.

After a number of bits equal to the value in the FLR have been shifted into the FDRI, frame data in the FDRI are loaded in parallel into the frame specified by the value in the FAR. After a write to the FDRI has completed, the Virtex-II device automatically interprets the next 32-bit word as an AutoCRC value (see CRC).

Configuration data frames are read from memory through the Frame Data Output Register. Prior to reading from the FDRO, the CMD register must be loaded with the RCFG command. The length of the read from the FDRO is specified by the FDRO read packet header.

The read operation is pipelined through the frame buffer such that the first frame is driven to the configuration interface as the second frame is being read from memory (Figure 4-36). For this reason, the first frame read out of the device is not valid data; readback data begins with the second word read from the device. Because of the frame buffer, the smallest FDRO read requires two frames of data: the first frame to be clear the frame buffer, followed by the frame of configuration data. The readback process is explained in detail in the Readback section.

Reads from the FDRO register do not generate a CRC value, nor is an AutoCRC performed after reading from the FDRO.

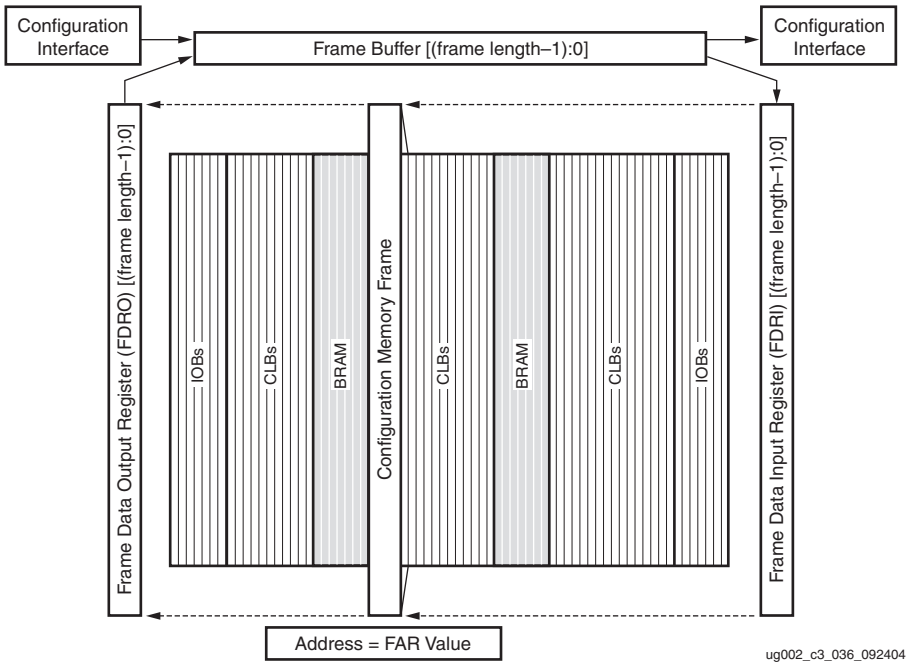


Figure 4-36: **Configuration Frame Data Path**

Command Register (CMD)

The Command Register is used to instruct the configuration control logic to strobe global signals and perform other configuration functions. Commands are executed immediately after being written to the CMD register. Additionally, whenever there is a write to the FAR, the last command written to the CMD register will be executed again. Table 4-24 gives the Command Register commands and codes.

Table 4-24: **Command Register Codes**

Command	Code	Description
WCFG	0001	Write Configuration Data. Used prior to writing configuration data to the FDRI.
MFWR	0010	Multiple Frame Write Register. Used to perform a write of a single frame to multiple frame addresses.
LFRM	0011	Last Frame. Deasserts the GHIGH_B signal, activating all interconnect. The GHIGH_B signal is asserted with the AGHIGH command.
RCFG	0100	Read Configuration Data. Used prior to reading configuration data from the FDRO.
START	0101	Begin Startup Sequence. Initiates the startup sequence. The startup sequence begins after a successful CRC check and a DESYNC command are performed.
RCAP	0110	Reset Capture. Resets the CAPTURE signal after performing readback-capture in single-shot mode.
RCRC	0111	Reset CRC. Resets the CRC register
AGHIGH	1000	Assert GHIGH_B Signal. Places all interconnect in a high-Z state to prevent contention when writing new configuration data. This command is only used during shutdown reconfiguration and readback. Interconnect is reactivated with the LFRM command.

Table 4-24: **Command Register Codes**

Command	Code	Description
SWITCH	1001	Switch CCLK Frequency. Updates the frequency of the Master CCLK to the value specified by the OSCFSEL bits in the COR.
GRESTORE	1010	Pulse the GRESTORE Signal. Sets/resets (depending on user configuration) IOB and CLB flip-flops. See "Readback Capture" on page 356.
SHUTDOWN	1011	Begin Shutdown Sequence. Initiates the shutdown sequence, disabling the device when finished. Shutdown activates on the next successful CRC check or RCRC instruction (typically an RCRC instruction).
GCAPTURE	1100	Pulse GCAPTURE. Loads the capture cells with the current register states.
DESYNCH	1101	Reset DALIGN Signal. Used at the end of configuration to desynchronize the device. After desynchronization, all values on the configuration data pins are ignored.

Control Register (CTL)

The Control Register is used to set the configuration security level, the persist setting, and to toggle the Global Three-State signal. Writes to the Control Register are masked by the value in the MASK Register, which allows the GTS_USR_B signal to be toggled without re-specifying the Security and Persist bits. The fields are illustrated in Figure 4-37 and defined in Table 4-25.

																								Reserved	SBITS		PERSIST	Reserved	Reserved	GTS_USR_B
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	0	0	x

Figure 4-37: **Control Register**

Table 4-25: **Control Register Description**

Name	Bit Index	Description
Reserved	Various	Reserved CTL register bits for testing. Always leave these bits set to 0.

Table 4-25: **Control Register Description**

Name	Bit Index	Description
SBITS	5:4	Security Level: 0: Read/Write OK (default) 1: Readback disabled 2,3: Readback disabled, writing disabled except to the CRC register.
PERSIST	3	The configuration interface defined by M2:M0 remains after configuration. Typically used only with the SelectMAP interface to allow reconfiguration and readback. 0: No (default) 1: Yes
GTS_USER_B	0	Active Low signal to place I/Os into a high-Z state. 0: I/Os placed in high-impedance state 1: I/Os active

Mask Register (MASK)

The Mask Register masks writes into the Control Register. A '1' in any bit within the MASK register allows the corresponding bit position to be written in the Control Register. The default value of the MASK Register is all '0's.

Status Register (STAT)

The Status Register indicate the value of numerous global signals. The register can be read via the SelectMAP or JTAG interfaces. Figure 4-38 gives the name of each bit position in the STAT register; a detailed explanation of each bit position is given in Table 4-26.

																BAD_KEY_SEQ	DEC_ERROR	ID_ERROR	DONE	INIT	MODE			GHIGH_B	GWE	GTS_CFG_B	IN_ERROR	DCI_MATCH	DCM_LOCK	PART_SECURED	CRC_ERROR
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 4-38: **Status Register**

Table 4-26: **Status Register Description**

Name	Bit Index	Description
BAD_KEY_SEQ	15	Encryption keys are stored with a tag indicating their position among the other keys. If the keys are sent in the wrong order, it will be indicated by this status bit. 0: No decryptor key sequence error 1: Decryptor keys were not used in the correct sequence
DEC_ERROR	14	Write FDRI issued before or after decrypt operation. 0: No decryptor error 1: Decryptor error. Assert PROG to recover
ID_ERROR	13	Attempt to write to FDRI without successful IDCODE check. 0: No ID_ERROR 1: ID_ERROR. Assert PROG to recover
DONE	12	The DONE bit in the STATUS register reflects the actual level on the DONE pin. If the DONE signal is released by the device but held Low externally, this bit remains Low. 0: DONE pin = logic 0 1: DONE pin = logic 1
INIT	11	The INIT bit in the STATUS register reflects the actual level on the INIT pin. 0: INIT pin = logic 0 1: INIT pin = logic 1
MODE	10:8	Status of the MODE pins (M2:M0).
GHIGH_B	7	Status of the GHIGH_B signal. 0: GHIGH_B is asserted 1: GHIGH_B is deasserted
GWE	6	Status of the Global Write Enable signal. 0: All FFs and BRAMs are write disabled 1: FFs and BRAMs are write enabled
GTS_CFG_B	5	Status of the GTS_CFG_B signal. 0: All I/Os in high-Z state 1: I/Os are not placed in High-Z by this signal Note that there are other ways of forcing I/Os into a high-Z state, including GTS_USR_B, as well as through various JTAG instructions.
IN_ERROR	4	Indicates that input data is being clocked in too fast. This can happen if the CCLK frequency exceeds $F_{CC\_SELECTMAP}$ in SelectMAP mode. 0: No legacy input error 1: Legacy input error

Table 4-26: **Status Register Description**

Name	Bit Index	Description
DCI_MATCH	3	This signal reflects the logical AND of all MATCH signals. There is one MATCH signal per I/O bank. If no DCI I/Os are instantiated in a given bank,the bank's MATCH signal = 1. 0: DCI not matched 1: DCI is matched
DCM_LOCK	2	This is a logical AND function of all DCM LOCKED signals. Unused DCM LOCKED signals =1. 0: DCMs not locked 1: DCMs are locked
PART_SECURED	1	Status of the Triple-DES decryptor. 0: Decryptor security not set 1: Decryptor security set
CRC_ERROR	0	Status of the CRC. 0: No CRC error 1: CRC error

Legacy Output Register (LOUT)

The Legacy Output Register is used to write data to the DOUT pin when configuring downstream devices in a serial configuration daisy-chain. The LOUT register is not available for SelectMAP and JTAG configuration modes.

Configuration Options Register (COR)

The Configuration Options Register is used to set certain configuration options for the device. The name of each bit position in the COR is given in Figure 4-39 and described in Table 4-27.

RESERVED	RESERVED	CRC_BYPASS	RESERVED		SHUT_RST_DCM	DONE_PIPE	DRIVE_DONE	SINGLE	OSCFSEL						SSCLKSRC		DONE_CYCLE			MATCH_CYCLE			LOCK_CYCLE			GTS_CYCLE			GWE_CYCLE		
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	x	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 4-39: **Configuration Options Register**

Table 4-27: **Configuration Options Register Description**

Name	Bit Index	Description
CRC_BYPASS	29	0: CRC enabled 1: CRC disabled
SHUT_RST_DCM	26	When this bit is set, all DCMs will be reset if the AGHIGH command is issued. 0: DCMs cannot be reset through the configuration logic 1: DCMs will reset during shutdown readback or reconfiguration
DONE_PIPE	25	Adds a pipeline stage for the DONEIN signal. The DONEIN signal reflects the logic level on the DONE pin, not whether the device has released the DONE pin. This pipeline stage can be useful if the rise time on the DONE pin is slow, which could otherwise cause configuration to fail. 0: No pipeline stage for DONEIN 1: Add pipeline stage for DONEIN
DRIVE_DONE	24	By default the DONE pin is an open-drain output, meaning that the pin actively drives a logic '0' before configuration, then goes into a high-Z state when it is "released" during Startup. This setting requires an external pull-up resistor on the DONE pin, and is typically applied to Slave devices in a serial daisy-chain. Alternately, the DONE pin can be configured as an actively-driven output, meaning that the pin actively drives a logic '1' when it is released. 0: DONE pin is an open-drain output (released = high-Z) 1: DONE pin is an actively driven output (released = logic '1')
SINGLE	23	The SINGLE bit determines whether the CAPTURE_VIRTEX2 primitive operates in one-shot mode or in continuous sample mode. This bit is set when the "one-shot" attribute is attached to the CAPTURE_VIRTEX2 primitive (see "Readback Capture" on page 356). 0: SINGLE mode disabled. In this mode, capture values are loaded on every clock cycle while CAP is asserted. 1: SINGLE mode enabled. In this mode, capture values are loaded only once, regardless of how many clock cycles the CAP signal is asserted for.
OSCFSEL	22:17	Selects the CCLK frequency for Master configuration modes (see Table 4-28).
SSCLKSRC	16:15	Startup sequence clock source: 00: CCLK 01: UserClk (per connection on the CAPTURE_VIRTEX2 block) 1x: JTAGClk

Table 4-27: **Configuration Options Register Description *(Continued)***

Name	Bit Index	Description
DONE_CYCLE	14:12	Startup cycle to release the DONE pin: 000: Startup cycle 1 001: Startup cycle 2 010: Startup cycle 3 011: Startup cycle 4 100: Startup cycle 5 101: Startup cycle 6
MATCH_CYCLE	11:9	Startup cycle to stall in until DCI matches: 000: Startup cycle 1 001: Startup cycle 2 010: Startup cycle 3 011: Startup cycle 4 100: Startup cycle 5 101: Startup cycle 6 111: No Wait
LOCK_CYCLE	8:6	Startup cycle to stall in until DCMs lock: 000: Startup cycle 1 001: Startup cycle 2 010: Startup cycle 3 011: Startup cycle 4 100: Startup cycle 5 101: Startup cycle 6 111: No Wait
GTS_CYCLE	5:3	Startup cycle to deassert the Global Three-State (GTS) signal: 000: Startup cycle 1 001: Startup cycle 2 010: Startup cycle 3 011: Startup cycle 4 100: Startup cycle 5 101: Startup cycle 6
GWE_CYCLE	2:0	Startup phase to deassert the Global Write Enable (GWE) signal: 000: Startup cycle 1 001: Startup cycle 2 010: Startup cycle 3 011: Startup cycle 4 100: Startup cycle 5 101: Startup cycle 6

Table 4-28: **OSCFSEL-Specified Master CCLK Settings**

CCLK (MHz)	OSCFSEL (binary)	CCLK (MHz)	OSCFSEL (binary)	CCLK (MHz)	OSCFSEL (binary)
4	000010	13	001010	41	100111
5	010001	15	001101	45	110011
7	000100	20	010111	51	101010
8	000101	26	011010	55	110100
9	000110	30	011101	60	101101
10	000111	34	110010		

Multiple Frame Write Register (MFWR)

The Multiple Frame Write Register is used when the bitstream compression option is enabled in BitGen (**`-g Compress:yes`**). When more than one frame has identical data, the frame data can be loaded once into the Multiple Frame Write Register then copied into multiple memory address locations. In some cases this can decrease the size of the bitstream considerably.

To write multiple frames with the same data, BitGen constructs a bitstream with the following commands:

1. Write WCFG command to CMD register
2. Write desired frame to FDRI
3. Write to FAR register with the first desired address
4. Write MFWR command to CMD register
5. Write two pad words to the MFWR register
6. Write to FAR register with the second desired address
7. Write two pad words to MFWR
8. Repeat step 6 and step 7 until the last desired address

Table 4-29: **Example Packets for Bitstream Compression Using MFWR**

Step	Configuration Data (hex)	Explanation
1	30002001	Type 1 write 1 words to FAR
	00000000	FAR Address = 0x00000000
2	30008001	Type 1 write 1 words to CMD
	00000001	WCFG command
3	30004034	Type 1 write 52 words to FDRI
	00000080	packet data write FDRI word 0
	. . .	
	00000000	packet data write FDRI word 51
	0000A11D	Auto CRC word
4	30002001	Type 1 write 1 words to FAR
	00000200	FAR Address = 0x00000200

Table 4-29: **Example Packets for Bitstream Compression Using MFWR *(Continued)***

Step	Configuration Data (hex)	Explanation
5	30008001	Type 1 write 1 words to CMD
	00000002	MFWR command
6	30014002	Type 1 write 2 words to MFWR
	00000000	packet data write MFWR word 0
	00000000	packet data write MFWR word 1
7	30002001	Type 1 write 1 words to FAR
	00000400	FAR Address = 0x00000400
8	30014002	Type 1 write 2 words to MFWR
	00000000	packet data write MFWR word 0
	00000000	packet data write MFWR word 1

Since configuration frames are arranged vertically, designs that span the fewest possible configuration frames achieve greater compression. Identical frames are usually found in Block RAM columns that are initialized with identical values or in unused CLB or Block RAM columns. To estimate the amount of compression achieved:

- Identify the number of identical CLB or BRAM columns. CLB columns are only identical if they are unused and have no routing passing through them. BRAM columns are identical if the same initialization values are used for multiple BRAM columns.
- Calculate the number of configuration frames for the column type of interest:

  # config frames = (# frames per column) x (# of columns)
- The number of saved configuration frames is given by:

  #Saved configuration frames = (#identical frames – 1).
- A rough estimate of the number of saved configuration bits is therefore given by:

  #Saved configuration bits = (# of saved configuration frames) x (# of bits per frame)

Frame Length Register (FLR)

Virtex-II frame lengths are not hard-coded into the device because the same configuration logic is used for all devices. The frame length must be programmed into the device before FDRI or FDRO operations can be performed.

The Frame Length Register (FLR) stores the device frame length, and must be written to before any FDRI or FDRO operation can be performed.

Note that the value written to the FLR is always one fewer than the actual number of words per frame. For example, the actual frame length of the XC2V40 is 26 words, however the value 0x00000019, or 25, is written to the FLR.

Initial Key Address Register (KEY)

The Initial Key Address Register (KEY) indicates the number of DES keys that are used for bitstream encryption. This is a write-only register; there is no way to read this register through any configuration interface or user logic resources.

Cipher Block Chaining Register (CBC)

The Cipher Block Chaining Register stores the DES encryption keys. This is a write-only register; there is no way to read the keys out of the device through any configuration interface or user logic resources.

Device ID (IDCODE)

The Device ID (IDCODE) Register contains the device-specific identification code. This register is separate from the JTAG IDCODE register although it uses the same 32-bit identification code. The purpose of this register is to ensure that the bitstream has been created for the correct device.

After writing to the IDCODE register the value is automatically compared; if it does not match the expected IDCODE value the ID_ERROR signal is asserted, halting configuration. This prevents a Virtex-II bitstream that is targeted for one array size from successfully programming a Virtex-II of a different array size—for example, an XC2V40 bitstream is prevented from programming an XC2V80 device by this mechanism. A successful write to the Device ID register is required prior to writing frame data. Table 4-30 lists Virtex-II IDCODEs.

Table 4-30: **Virtex-II IDCODEs**

Device	IDCODE	Device	IDCODE
XC2V40	0x01008093	XC2V2000	0x01038093
XC2V80	0x01010093	XC2V3000	0x01040093
XC2V250	0x01018093	XC2V4000	0x01050093
XC2V500	0x01020093	XC2V6000	0x01060093
XC2V1000	0x01028093	XC2V8000	0x01070093
XC2V1500	0x01030093		

Configuration

Configuration can begin after the device is powered and initialization has finished, as indicated by the INIT pin being released. After initialization, the packet processor ignores all data presented on the configuration interface until it receives the synchronization word; after synchronization the packet processor waits for a valid packet header to begin the configuration process.

Default Initial Configuration Process

Initial configuration using a "default" bitstream (a bitstream generated using the default BitGen settings) begins by pulsing the PROG_B pin for SelectMAP and Serial configuration modes or by issuing the JPROG_B instruction for JTAG configuration mode. Configuration proceeds as follows:

Default Initial Configuration

1. **Device Synchronization.** Virtex-II devices are synchronized by sending the synchronization word (0xAA995566) to the configuration logic. The synchronization word tells the device to begin parsing packet data.
2. **Reset CRC Register.** The CRC register is reset to ensure a correct CRC calculation. See "CRC Register (CRC)" on page 319 for details.
3. **Write to Frame Length Register.** Virtex-II devices are not hard-coded with the correct frame length. Prior to configuraiton, the FLR must be written with a value equal to:

   (number of words per frame) – 1

   See "Frame Length Register (FLR)" on page 330 for details.
4. **Write to Configuration Options Register** (startup sequence options, etc.). See"Configuration Options Register (COR)" on page 326 for details on which configuration options are programmed into the COR.

5. **Write to Device ID Register.** A successful write to the Device ID register is required before frame data can be written. See "Device ID (IDCODE)" on page 331 for details.
6. **Write to MASK register** (depends on user settings). The SBITS and PERSIST bits in the CTL register are masked prior to configuration so the GTS_USR_B signal can be pulsed as needed after configuration.
7. **Write SWITCH command to CMD register** (needed for Master configuration modes only). The SWITCH command changes the CCLK output frequency according to the BitGen **`-g configrate`** configuration setting. This command has no effect during slave and JTAG configuration modes.
8. **Write starting Frame Address to FAR** (`0x00000000`). The FAR is set with the starting frame address for initial configuration. Only the starting frame address needs to be written during configuration; the FAR is automatically incremented after every frame is written to FDRI.
9. **Write WCFG command to CMD register.** The WCFG command allows frame data to be written to configuration memory.
10. **Write configuration frame data to FDRI, including 1 frame of pad data.** One frame of pad data must follow the frame data write to flush the frame buffer, as explained in "Frame Data Input Register (FDRI) and Frame Data Output Register (FDRO)" on page 321. For full configuration, the length of the FDRI write is given by:

    #FDRI Words = (#frames + 1 pad frame) x (frame length in words)

    After the last frame of data is sent to the device, a frame of pad data is sent in the same FDRI write to push the last data frame from the Frame Buffer to the FDRI register. This pad frame is included in the packet header word count. The data content of the pad frame does not matter.

    After the FDRI write finishes, the next 32-bit word is interpreted by the device as an AutoCRC value. The AutoCRC value is not included in the packet word count.
11. **Write the GRESTORE command to the CMD register.** GRESTORE command updates all slice and IOB registers with their initial values.
12. **Write the Last Frame (LFRM) command to the CMD register.** The LFRM command deasserts the internal GHIGH signal, activating all interconnect. The GHIGH signal is first activated during initialization (i.e., after the PROG pin is pulsed), and remains activated until the LFRM command is issued.

Include step 13 through step 15 if **`-g FreezeDCI:Yes`** functionality is needed. Otherwise, skip to step 16.

13. **Write WCFG command to the CMD register**. If the **`-g FreezeDCI:Yes`** option is set in BitGen (this is the default setting), a second write to FDRI appears in the bitstream (step 13 through step 15). This FDRI write begins with the WCFG command. If **`-g FreezeDCI:No`** is set, step 13 through step 15 does not occur in the bitstream.
14. **Set the FAR to the first IOI Column.**
15. **Write 2 frames to FDRI (frame to configure plus pad frame).** This frame reconfigures the DCI oscillator to turn it off.
16. **Send START command to bring device into startup sequence.** The START command begins the startup sequence for SelectMAP and Serial configuration modes, using the CCLK or USRCLK signal to clock the sequence. For JTAG the JTAG JSTART instruction must be issued so that the JTAG TCK clocks the startup sequence.
17. **Write to Control Register.** See "Control Register (CTL)" on page 323 for more information.

18. **Perform final CRC check.** A CRC value is always created, even if CRC checking is disabled in BitGen. When CRC checking is disabled, the default CRC value of `0x0000DEFC` is written to the CRC register.
19. **Send DESYNC command.** The DESYNC command "desynchronizes" the configuration logic. Further configuration data are ignored until a new synchronization word is received.
20. **Write at least 64 bits of pad data (NOOP commands) to flush packet buffer.** The packet buffer must be flushed with 64 bits of packet data. For this sequence of commands, the pad bits push the DESYNC command into the packet processor.
21. **Continue sending CCLK or TCK pulses until DONE goes High.** Depending on the startup options and external conditions (for example, if the DONE pin is held Low externally), additional CCLK pulses might be needed to bring the device out of the startup sequence. Refer to "Configuring Through Boundary-Scan" on page 305 for further details on using this procedure with the JTAG configuration interface.

Each of these steps is performed by a single configuration packet except for synchronization (step 1) and the large FDRI write (step 10), which is performed by a Type 1, Type 2 packet combination. Table 4-31 shows how the default configuration steps are written in an XC2V40 bitstream.

Table 4-31: **Default XC2V40 Bitstream for Initial Configuration**

Step	Configuration Data (hex)	Explanation
1	AA995566	Sync word
2	30008001	Type 1 write 1 words to CMD
	00000007	RCRC command
3	30016001	Type 1 write 1 words to FLR
	00000019	Frame Length = 25 words (actual XC2V40 frame length – 1 = 25 words = 0x19)
4	30012001	Type 1 write 1 words to COR
	XXXXXXXX	COR settings (dependent on user options)
5	3001C001	Type 1 write 1 words to ID
	01008093	Device ID= 0x01008093 (XC2V40)
6	3000C001	Type 1 write 1 words to MASK
	XXXXXXXX	CTL Register MASK (dependent on user options)
7	30008001	Type 1 write 1 words to CMD
	00000009	SWITCH command
8	30002001	Type 1 write 1 words to FAR
	00000000	FAR Address= 0x00000000
9	30008001	Type 1 write 1 words to CMD
	00000001	WCFG command

Table 4-31: **Default XC2V40 Bitstream for Initial Configuration *(Continued)***

Step	Configuration Data (hex)	Explanation
10	30004000	Type 1 write 0 words to FDRI
	50002922	Type 2 write 10530 words to FDRI: (404 frames + 1) x (26 words per frame) = 10530 words
	00100000	Packet data write FDRI word 0
	. . .	
	00000000	Packet data write FDRI word 10529
	XXXXXXXX	Auto CRC word (design dependent)
11	30008001	Type 1 write 1 words to CMD
	0000000A	GRESTORE command
12	30008001	Type 1 write 1 words to CMD
	00000003	LFRM command
13(1)	30008001	Type 1 write 1 words to CMD
	00000001	WCFG command
14(1)	30002001	Type 1 write 1 words to FAR
	00042400	FAR Address = 0x00042400
15(1)	30004034	Type 1 write 52 words to FDRI
	00000000	Packet data write FDRI word 0
	. . .	
	00000000	Packet data write FDRI word 51
	XXXXXXXX	Auto CRC word (design dependent)
16	30008001	Type 1 write 1 words to CMD
	00000005	START command
17	3000A001	Type 1 write 1 words to CTL
	XXXXXXXX	CTL settings (dependent on user options)
18	30000001	Type 1 write 1 words to CRC
	XXXXXXXX	CRC (design dependent)
19	30008001	Type 1 write 1 words to CMD
	0000000D	DESYNCH command
20	20000000	Type 1 NOOP word 0
	20000000	Type 1 NOOP word 1
21	XXXXXXXX	Continue sending CCLK or TCK pulses until DONE goes High.

Notes:

1. Include step 13 through step 15 in the bitstream only if the **`"-g FreezeDCI:Yes"`** option is set in BitGen.

Active and Shutdown Reconfiguration

The term *reconfiguration* in this context describes the process of reprogramming a device without pulsing the PROG pin or cycling power. Reconfiguration can involve the whole

device or just a portion of it, and can occur while the device is in the Shutdown state (*shutdown reconfiguration*), or while the device continues to operate (*active reconfiguration*).

Since the user design continues to operate during active reconfiguration, this style of reconfiguration must be used with caution. Internal contention can occur if a device is overwritten with a new design through active reconfiguration, potentially damaging the device. Furthermore, all SRL16, Distributed RAM, and BlockRAM contents are reinitialized by a full bitstream, clearing any data written into these memory elements by the user design.

To bring the device into the Shutdown state through the SelectMAP interface, the configuration logic is synchronized and the SHUTDOWN command is written to the CMD register (Table 4-32). To bring the device into the Shutdown state through the JTAG interface, the JTAG JSHUTDOWN command is issued and the TAP controller is brought to the RTI state for 12 TCK cycles.

The Shutdown sequence runs the startup sequence in reverse:

- Deassert EOS
- Pull the DONE pin Low
- Assert GTS, placing all I/Os in a high-Z state (except for persistent conrfiguration pins)
- De-assert GWE, preventing RAMs and flip-flops from changing state

Neither the start-up sequence nor the shutdown sequence can finish if the CRC_ERROR signal is asserted. To ensure the shutdown sequence completes successfully, it should always be preceded by the RCRC (Reset CRC) command.

To bring a device out of the shutdown state, the device must be brought through the startup sequence.

Table 4-32: **SelectMAP Shutdown Sequence**

Configuration Data	Explanation
30008001	Type 1 write 1 words to CMD
0000000B	SHUTDOWN command

Table 4-33: **JTAG Shutdown Sequence**

Description	Set and Hold		# of Clocks
	TDI	TMS	TCK
Clock five 1s on TMS to bring the device to the TLR state	X	1	5
Move into the RTI state	X	0	1
Move into the Select-IR state	X	1	2
Move into the Shift-IR State	X	0	2
Shift the first 5 bits of the JSHUTDOWN instruction, LSB first	01101	0	5
Shift the MSB of the JSHUTDOWN instruction while exiting SHIFT-IR	0	1	1
Move into the Update-IR State	X	1	1
Move into the RTI state; remain there for 12 TCK cycles	X	0	12

The procedure for shutdown reconfiguration is identical to the "default" configuration sequence described above, except that the SHUTDOWN command is issued to the device immediately after synchronization.

Shutdown Reconfiguration Sequence

1. Device Synchronization.
2. Write SHUTDOWN command to CMD register.
3. Reset CRC Register.
4. Write to Frame Length Register (actual frame length – 1).
5. Write to Configuration Options Register (startup sequence options, etc.)
6. Write to Device ID Register.
7. Write to MASK register (depends on user settings).
8. Write SWITCH command to CMD register (needed for Master configuration modes only).
9. Write starting Frame Address to FAR (`0x00000000`).
10. Write WCFG command to CMD register.
11. Write configuration frame data to FDRI, including 1 frame of pad data.
12. Write the GRESTORE command to the CMD register.
13. Write Last Frame (LFRM) command.
14. Write 1 frame of NOOP commands.
15. Write WCFG command to CMD register (include step 15 through step 17 if "**`-g freezedci:yes`**" functionality is needed).
16. Set FAR to frame address.
17. Write 2 frames to FDRI (frame to configure plus pad frame).
18. Write START command to CMD register to begin startup sequence.
19. Perform final CRC check.
20. Write DESYNC command to CMD register.
21. Write at least 64 bits of pad data (NOOP commands) to flush packet buffer.
22. Continue sending CCLK pulses until DONE goes High.

Partial Reconfiguration

Virtex-II devices allow *partial reconfiguration*—rewriting a subset of configuration frames, either while user design is suspended ("Shutdown" partial reconfiguration) or while the user design is operating ("Active" partial reconfiguration). There are two reasons to perform partial reconfiguration:

- To change design behavior without fully reconfiguring a device
- To correct memory upsets in high-radiation environments

This document addresses the needs of commercial customers wishing to change design behavior. Aerospace customers seeking more information on the use of partial reconfiguration for correcting configuration memory upsets in high-radiation environments should contact their Xilinx FAE or Sales Representative, or open a technical support Webcase at **http://www.support.xilinx.com**.

Since Xilinx does not document the function of the configuration bits within every frame, users typically combine the information in the Partial Reconfiguration Process section with

one of the software flows described in the Partial Reconfiguration Applications and Software Flows section below.

Partial Reconfiguration Process

This section explains the mechanics of partially reconfiguring a single frame of configuration data. Similar to full reconfiguration, partial reconfiguration can be performed while the user design continues to operate or after placing the design in a shutdown state. Partial reconfiguration is only possible through the SelectMAP and JTAG interfaces. For SelectMAP partial reconfiguration, the "persist" setting must be enabled to retain the function of the SelectMAP pins after configuration.

Shutdown Reconfiguration Sequence

1. Device Synchronization.
2. Write SHUTDOWN command to CMD register.
3. Reset CRC Register.
4. Write to Frame Length Register (actual frame length – 1).
5. Write to Configuration Options Register (optional).
6. Write to Device ID Register.
7. Write to MASK register (depends on user settings).
8. Write Frame Address to FAR.
9. Write WCFG command to CMD register.
10. Write frame data to FDRI (packet size = # frames + 1 pad frame).
11. Write the default AutoCRC value of (`0x0000DEFC`).
12. Write the GRESTORE command to the CMD register.
13. Send Last Frame (LFRM) command.
14. Send 1 frame of NOOP commands.
15. Send WCFG command.
16. Set FAR to last frame address.
17. Write 1 pad frame to FDRI.
18. Send START command to bring device into startup sequence.
19. Perform final CRC check.
20. Send DESYNC command.
21. Send at least 64 bits of pad data to flush packet buffer.
22. Continue sending CCLKs pulses until DONE goes High.

Active Partial Reconfiguration Sequence

1. Synchronize device.
2. Reset CRC.
3. Write IDCODE register.
4. Write Frame Length to FLR (actual frame length – 1).
5. Write WCFG command to CMD register.
6. Write frame address to FAR.

7. Write 212 words to FDRI (1 frame + 1) x (106 words per frame) = 212 words.
8. Write default AutoCRC value of `0x0000DEFC`.
9. Reset CRC.
10. Write default CRC value (`0x0000DEFC`).
11. Write DESYNCH command.
12. Write two NOOP words (`0x20000000`) to flush packet buffer.

Consecutive frames of configuration frame data can be written to FDRI in a single packet (step 10 in the Shutdown Partial Reconfiguration process; step 7 in the Active Reconfiguration process), since the FAR automatically increments when each frame is written to configuration memory. In some cases, it is necessary to write to non-consecutive frames; in such instances, the new frame address must be explicitly written to the FAR and steps 8-11 repeated. As always, one frame of pad data must be included at the end of every FDRI packet, and the word following each FDRI packet is interpreted as an AutoCRC.

The precautions that apply to full active reconfiguration (see "Active and Shutdown Reconfiguration" on page 334) also apply to partial active reconfiguration. Table 4-34 shows the configuration packets needed to actively reconfigure a single frame on an XC2V40 device.

Table 4-34: **Active Partial Reconfiguration for a Single XC2V40 Frame**

Step	Configuration Data	Explanation
1	AA995566	Sync Word
2	30008001	Write to CMD
	00000007	RCRC
3	3001C001	Write to IDCODE register
	01008093	Device ID
4	30016001	Write to FLR
	00000019	Frame Length
5	30008001	Write to CMD
	00000001	WCFG
6	30002001	Write to FAR
	XXXXXXXX	Frame address
7	30004034	Write 52 words to FDRI
	. . .	
8	0000DEFC	Default AUTOCRC value
9	30008001	Write to CMD
	00000007	RCRC
10	30000001	Type 1 write 1 words to CRC
	0000DEFC	CRC = 0x0000DEFC (Default value)

Table 4-34: **Active Partial Reconfiguration for a Single XC2V40 Frame *(Continued)***

Step	Configuration Data	Explanation
11	30008001	Write to CMD
	0000000D	Desync command
12	00000002	NOOP
	00000002	NOOP

Partial Reconfiguration Applications and Software Flows

One challenge for designers seeking to use partial reconfiguration is determining which frames to reconfigure and what the new contents of those frames should be. There are four software flows available to partial reconfiguration users: the difference-based BitGen **-r** flow, the Modular Design flow, the BitGen "PartialMask" flow, and the BlockRAM "savedata" feature.

Difference-Based BitGen -r Flow

In the difference-based BitGen "**-r**" flow, users provide two input design (.ncd) files to BitGen, designating one as the initial configuration and the other as the secondary configuration. BitGen produces an active partial bitstream that only configures the frames that are different between the two design files. The partial bitstream is intended only for downloading to a device that has been configured by a bitstream for the initial configuration. The difference-based flow is well-suited partial reconfiguration applications where only small design changes are needed—changing LUT equations on the fly, for example.

Partial bit files generated with the BitGen "**-r**" setting automatically use the multiple frame write feature and are therefore "compressed" bitstreams. The Difference-Based flow is described in greater detail in Xilinx Application note **XAPP290**, *Two Flows for Partial Reconfiguration: Module-Based or Difference-Based*.

Modular Design

The Modular Design flow for partial reconfiguration, described in Xilinx Application Note XAPP290, is intended for partially reconfiguring larger design changes into a system. The flow provides for a static, non-reconfigurable design area and a dynamic, reconfigurable area that can be partially reconfigured with two or more partial reconfiguration modules.

PartialMask

The BitGen "PartialMask" feature allows users to pick which configuration columns are included in an active reconfiguration bitstream. PartialMask bitstreams are intended only for active partial reconfiguration, and must be used with the **-g activereconfig:yes** BitGen switch. Bitstreams that are created using this flow cannot be used for initial configuration, since they do not include the START command or allow for the startup sequence.

Per the *Development System Reference Guide*, there are six PartialMask settings:

- **PartialGCLK**. Adds the center global clock column to the list of columns written to a partial bitstream. Equivalent to the PartialMask0:1 setting.
- **PartialLeft**. Adds all columns on the left side of the device, excluding the global clock column, to the list of columns written to a partial bitstream.
- **PartialRight**. Adds all columns on the right side of the device, excluding the global clock column, to the list of columns written to a partial bitstream.
- **PartialMask0 <mask>**. Adds columns in BA0 (Block Address 0: GCLK, IOB, IOI, and CLB columns) as indicated by the hexadecimal mask to the list of columns written to a

partial bitstream.

- **PartialMask1 <mask>**. Adds columns in BA1 (BRAM columns) as indicated by the hexadecimal mask to the list of columns written to a partial bitstream.
- **PartialMask2 <mask>**. Adds columns in BA2 (BRAM Interconnect columns) as indicated by the hexadecimal mask to the list of columns written to a partial bitstream.

For the PartialMaskN settings, <mask> is a hexadecimal string that indicates which columns within a memory block are included in the partial bitstream. The bits in the hex string map to configuration columns; each '1'indicates a column that is included in the partial bitstream, while each '0' indicates a column that is excluded from the partial bitstream. Nibbles are rounded out by appending '0's to the bit positions that do not have corresponding configuration columns. Figure 4-40 illustrates the mapping between the <mask> bit positions and the configuration columns.

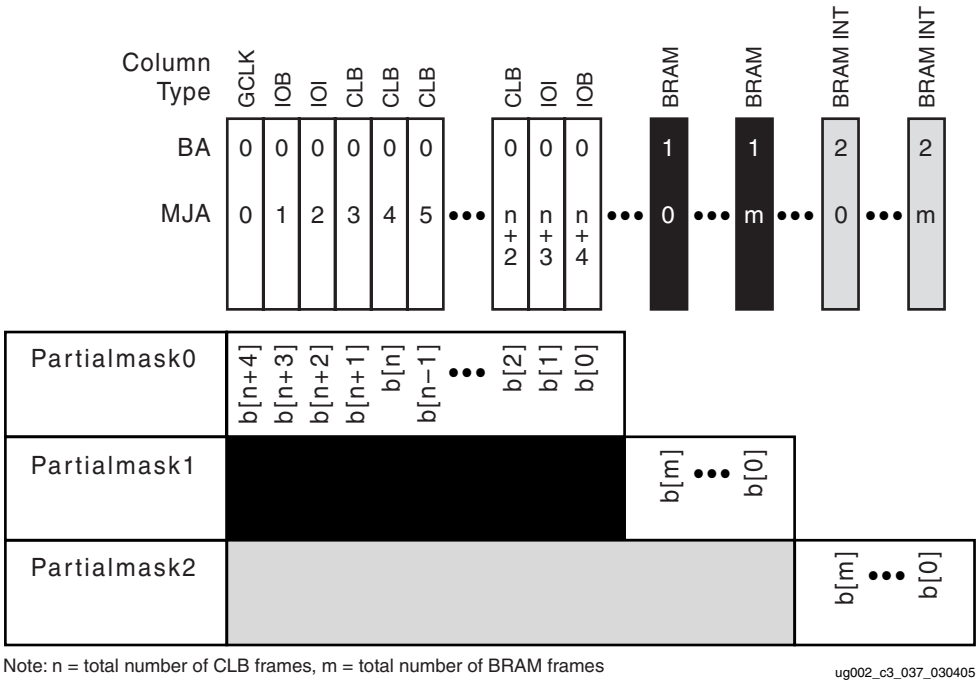


Figure 4-40: **PartialMask <mask> Bit Mapping**

Two examples follow for deriving masks for a XC2V40 device. From Table 4-17 on page 315, the XC2V40 device has eight CLB columns, two BlockRAM columns, and two BlockRAM Interconnect columns.

PartialMask Example 1: All columns included in bitstream except BRAM columns.

To include all columns in BA0 and BA1, the <mask> for PartialMask0 and PartialMask1 is created using 1's for all BA0 and BA1 columns. To exclude all columns in BA2, the <mask> for PartialMask2 is all 0's. Figure 4-41 shows how the <mask> values are derived.

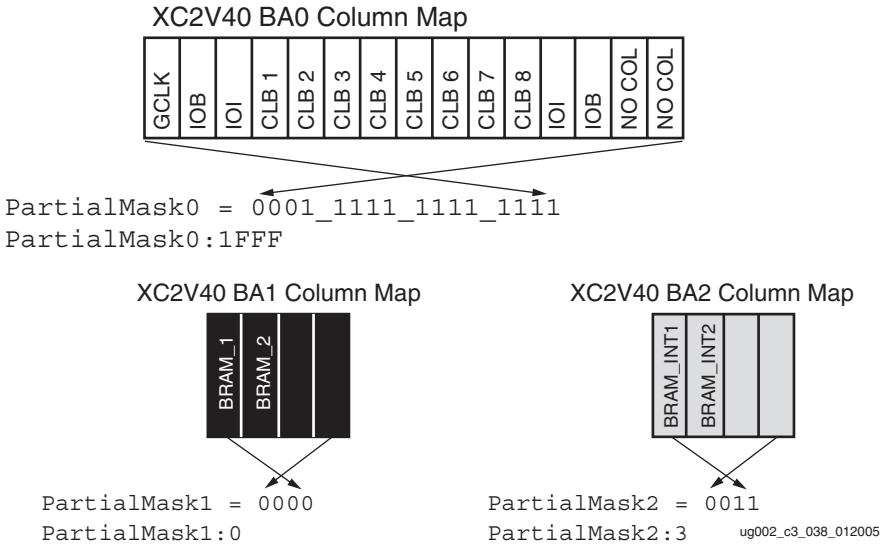


Figure 4-41: **PartialMask <mask> Derivation for Example 1**

To generate a partial bitstream for Example 1, use the command line:

```
bitgen -g ActiveReconfig:Yes -g PartialMask0:1FFF
-g PartialMask1:0 -g PartialMask2:3 <design>.ncd
```

PartialMask Example 2: Only BRAM columns included in bitstream.

To exclude all columns in BA0 and BA1, the <mask> for PartialMask0 and PartialMask1is all '0's. To include all columns in BA2, the <mask> for PartialMask2 is all '1's. Figure 4-42 shows how the <mask> values are derived.

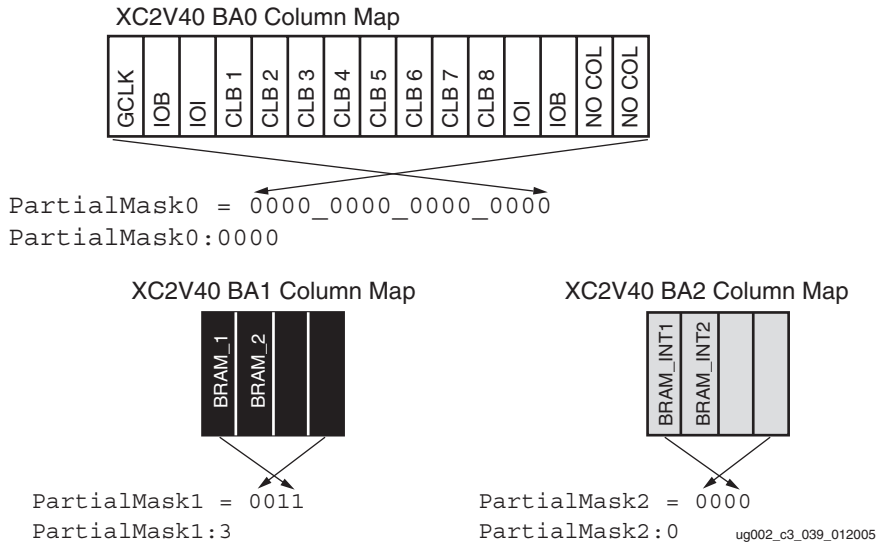


Figure 4-42: **PartialMask <mask> Derivation for Example 2**

To generate a partial bitstream for Example 2, use the command line:

```
bitgen -g ActiveReconfig:Yes -g PartialMask0:0000
-g PartialMask1:3 -g PartialMask2:0 <design>.ncd
```

BlockRAM "Savedata"

Writes to Virtex-II BlockRAMs during shutdown reconfiguration can be disabled by setting the BlockRAM "Savedata" option in the original configuration bitstream. The Savedata option is enabled using FPGA Editor, as described in Xilinx Application Note **XAPP290**.

The Savedata option is not intended for use during active reconfiguration, as it can interfere with BlockRAM operation. The Savedata option is safe to use with Shutdown reconfiguration.

Readback

Virtex-II devices allow users to read configuration memory through the SelectMAP or JTAG interface. There are two styles of readback: Readback Verify and Readback Capture. During Readback Verify, the user reads all configuration memory cells, including the current values on all user memory elements (LUT RAM, SRL16, and BlockRAM). Readback Capture is a superset of Readback Verify: in addition to reading all configuration memory cells, the current state of all internal CLB and IOB registers is read, which can be useful for design debugging. Readback is possible while the FPGA design is active or in a shutdown state, although users are advised to avoid active readback because it can corrupt BlockRAM, Distributed RAM, and SRL16 contents. During active readback, the user design continues to operate while configuration memory are being read.

To read configuration memory, users must send a sequence of commands to the device, indicating the starting configuration memory readback address and the number of words to be read from memory. Once initiated, the device sends the specified memory contents to the SelectMAP or JTAG interface. The following sections describe the steps for reading configuration memory:

- "Configuration Memory Read Procedure (SelectMAP)" on page 344
- "Configuration Memory Read Procedure (1149.1 JTAG)" on page 348
- "Configuration Memory Read Procedure (1532 JTAG)" on page 352

Users can send the readback command sequence from a microprocessor, CPLD, or FPGA-based system, or use iMPACT to perform JTAG-based readback verify. iMPACT, the device programming software provided with the Xilinx Integrated Software Environment (ISE), can perform all readback and comparison functions for Virtex-II devices and report to the user whether there were any configuration errors. iMPACT cannot, however, perform capture operations, although Readback Capture is seldom used for design debugging since Chipscope ILA, sold separately through the Xilinx website, provides superior design debugging functionality in a user-friendly interface. See "Using ChipScope ILA" on page 357 for more information.

Once configuration memory has been read from the device, the next step is to determine if there are any errors by comparing the readback bitstream to the configuration bitstream. An explanation of how this is done can be found in "Verifying Readback Data" on page 353.

Preparing a Design for Readback

There are two mandatory bitstream settings for readback: the BitGen security setting must not prohibit readback (`-g security:none`), and bitstream encryption must not be used. Additionally, if readback is to be performed through the SelectMAP interface, the port must be set to retain its function after configuration by setting the "persist" option in BitGen (`-g Persist:Yes`). Otherwise, the SelectMAP data pins revert to user I/O, precluding further configuration operations. Beyond these security and encryption requirements, no special considerations are necessary to enable readback through the Boundary-Scan port.

If capture functionality is needed, the CAPTURE_VIRTEX2 primitive must be instantiated in the user design (Figure 4-48 on page 356). To capture the state of user registers, the user design triggers the CAP input on this primitive, storing the current register values in configuration memory. The register values are later read out of the device along with all other configuration memory. See "Readback Capture" on page 356 for more information.

Readback Command Sequences

Virtex-II configuration memory is read from the FDRO (Frame Data Register - Output) register, which can be accessed from the JTAG and SelectMAP interfaces. Readback is possible while the FPGA design is active or in a shutdown state, although users are advised to avoid active readback because it can corrupt BlockRAM, Distributed RAM, and SRL16 contents.

Configuration Register Read Procedure (SelectMAP)

The simplest read operation targets a configuration register such as the COR or STAT register. Any configuration register with read access can be read through the SelectMAP interface, although not all registers offer read access, as noted in Table 4-20 on page 319. The procedure for reading the STAT register through the SelectMAP interface follows:

1. Write the Synchronization word to the device.
2. Write the "read STAT register" packet header to the device.
3. Write two pad words to the device to flush the packet buffer.
4. Read one word from the SelectMAP interface. This is the Status register value.
5. Write the DESYNC command to the device
6. Write two pad words to the device to flush the packet buffer.

Table 4-35: **Status Register Readback Command Sequence (SelectMAP)**

Step	SelectMAP Port Direction	Configuration Data (hex)	Explanation
1	Write	AA995566	Sync Word
2	Write	2800E001	Read 1 word from STAT register
3	Write	20000000	NOOP
		20000000	NOOP
4	Read	SSSSSSSS	Device writes 1 word from the STAT register to the configuration interface
5	Write	0000000D	Desync command
6	Write	20000000	NOOP
		20000000	NOOP

The user must change the SelectMAP interface from write to read control between steps 3 and 4, and back to write control after step 4, as illustrated in Figure 4-43.

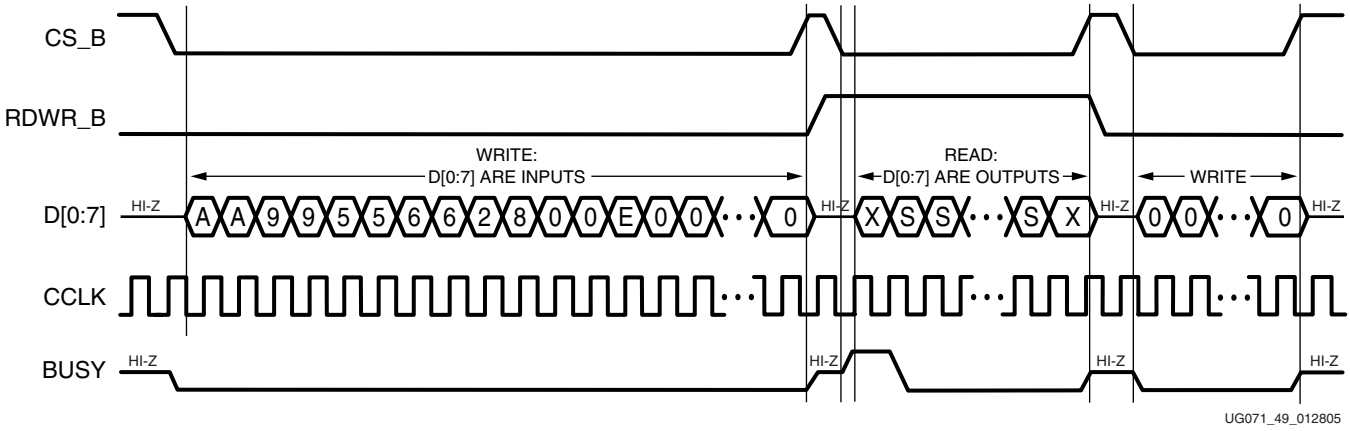


Figure 4-43: **SelectMAP Status Register Read**

To read registers other than STAT, the address specified in the Type 1 packet header in Step 2 of Table 4-35 should be modified and the word count changed if necessary. Reading from the FDRO register is a special case that is described in the procedure below.

Configuration Memory Read Procedure (SelectMAP)

The process for reading configuration memory from the FDRO register is similar to the process for reading from other registers. However, additional steps are needed to accommodate the configuration logic. Configuration data coming from the FDRO register pass through the frame buffer; therefore, the first frame of readback data should be discarded. (Refer to "Frame Data Input Register (FDRI) and Frame Data Output Register (FDRO)" on page 321).

1. Write the Synchronization word to the device.
2. Write the SHUTDOWN command.
3. Write the RCRC command.
4. Write four NOOP instructions to ensure the shutdown sequence has completed. DONE goes Low after the SHUTDOWN sequence has completed.
5. Write the Starting Frame Address to the FAR (typically `0x00000000`).
6. Write the RCFG command to the CMD register.
7. Write the "read FDRO register" packet header to the device. The FDRO read length is given by:

   FDRO Read Length = (words per frame) x (frames to read + 1)

   One extra frame is read to account for the frame buffer, which produces one pad frame at the beginning of the read.
8. Write two pad words to the device to flush the packet buffer.
9. Read the FDRO register through the SelectMAP interface. The FDRO read length is the same as in step 7 above. Frame data will be preceded by one frame of pad bits.
10. Write the START command.
11. Write the RCRC command.
12. Write the DESYNCH command.
13. Write at least 64 bits of NOOP commands to flush the packet buffer. Continue sending CCLK pulses until DONE goes High.

Each of these steps is performed by a single configuration packet except for synchronization (step 1) and the large FDRO read (step 7), which is performed by a Type 1, Type 2 packet combination.

For active readback, step 5 and step 10 are omitted. Remember that BRAM frames and CLB Frames that contain SRL16s or Distributed RAM must not be read back actively. In practice, active readback of the CLB frames should stop at the second-to-last CLB frame, since reading back the last CLB frame will cause the FAR to increment into the first BRAM frame, corrupting the BRAM contents.

Table 4-36 shows the readback command sequence.

Table 4-36: **Shutdown Readback Command Sequence (SelectMAP), XC2V40 Device**

Step	SelectMAP Port Direction	Configuration Data (hex)	Explanation
1	Write	AA995566	Sync word
2	Write	30008001	Type 1 write 1 words to CMD
	Write	0000000B	SHUTDOWN command
3	Write	30008001	Type 1 write 1 words to CMD
	Write	00000007	RCRC command
4	Write	20000000	Type 1 NOOP word 0
	Write	20000000	Type 1 NOOP word 1
	Write	20000000	Type 1 NOOP word 2
	Write	20000000	Type 1 NOOP word 3
5	Write	30002001	Type 1 write 1 words to FAR
	Write	00000000	FAR address= 0x00000000
6	Write	30008001	Type 1 write 1 words to CMD
	Write	00000004	RCFG command
7	Write	28006000	Type 1 read 0 words from FDRO
	Write	480222FA	Type 2 read 10530 words from active register (FDRO)
8	Write	20000000	Type 1 NOOP word 0
	Write	20000000	Type 1 NOOP word 1
9	Read	00000000	Packet data read FDRO word 0
	Read	. . .	
	Read	00000000	Packet data read FDRO word 10529
10	Write	30008001	Type 1 write 1 words to CMD
	Write	00000005	START command
11	Write	30008001	Type 1 write 1 words to CMD
	Write	00000007	RCRC command
12	Write	30008001	Type 1 write 1 words to CMD
	Write	0000000D	DESYNCH command
13	Write	20000000	Type 1 NOOP word 0
	Write	20000000	Type 1 NOOP word 1

Accessing Configuration Registers through the JTAG Interface

JTAG access to the Virtex-II configuration logic is provided through the JTAG CFG_IN and CFG_OUT registers. Note that the CFG_IN and CFG_OUT registers are not configuration registers; rather, they are JTAG registers like BYPASS and BOUNDARY_SCAN. Data shifted in to the CFG_IN register go to the configuration packet processor, where they are processed in the same way commands from the SelectMAP interface are processed.

Readback commands are written to the configuration logic through the CFG_IN register; configuration memory is read through the CFG_OUT register. The JTAG state transitions for accessing the CFG_IN and CFG_OUT registers are described in Table 4-37.

Table 4-37: **Shifting in the JTAG CFG_IN and CFG_OUT Instructions**

Step	Description	Set and Hold		# of Clocks (TCK)
		TDI	TMS	
1	Clock five '1's on TMS to bring the device to the TLR state	X	1	5
2	Move into the RTI state	X	0	1
3	Move into the SELECT-IR state	X	1	2
4	Move into the SHIFT-IR State	X	0	2
5	Shift the first 5 bits of the CFG_IN or CFG_OUT instruction, LSB first	00101 (CFG_IN) 00100 (CFG_OUT)	0	5
6	Shift the MSB of the CFG_IN or CFG_OUT instruction while exiting SHIFT-IR	0	1	1
7	Move into the SELECT-DR state	X	0	2
8	Move into the SHIFT-DR state	X	0	2
9	Shift data into the CFG_IN register or out of the CFG_OUT register while in SHIFT_DR, MSB first.	X	0	X
10	Shift the LSB while exiting SHIFT-DR	X	1	1
11	Reset the TAP by clocking five 1's on TMS	X	1	5

Configuration Register Read Procedure (JTAG)

The simplest read operation targets a configuration register such as the COR or STAT register. Any configuration register with read access can be read through the JTAG interface, although not all registers offer read access, as noted in Table 4-20 on page 319. The procedure for reading the STAT register through the JTAG interface follows:

1. Reset the TAP controller.
2. Shift the CFG_IN instruction into the JTAG Instruction Register through the SHIFT-IR state. The LSB of the CFG_IN instruction is shifted first; the MSB is shifted while moving the TAP controller out of the SHIFT-IR state.
3. Shift packet write commands into the CFG_IN register through the SHIFT-DR state:
   a. Write the Synchronization word to the device.
   b. Write the "Read STAT register" packet header to the device.
   c. Write exactly three pad words to the device to flush the packet buffer.

The MSB of all configuration packets sent through the CFG_IN register must be sent first; the LSB is shifted while moving the TAP controller out of the SHIFT-DR state. Note that this is the opposite of how JTAG instructions are shifted into the JTAG Instruction Register.

Since an odd number of words (1) is sent to the device for this JTAG read operation, it must end with exactly three pad words. This is because all JTAG read command sequences must consist of an even number of words (see "Packet Processor" on page 318 for details).

4. Shift the CFG_OUT instruction into the JTAG Instruction Register through the Shift-IR state. The LSB of the CFG_OUT instruction is shifted first; the MSB is shifted while moving the TAP controller out of the SHIFT-IR state.
5. Shift 32 bits out of the Status register through the SHIFT-DR state.
6. Shift the CFG_IN instruction into the JTAG Instruction Register through the Shift-IR state.
7. Shift packet write commands into the CFG_IN register through the Shift-DR state:
   a. Write the “Write to CMD register” packet header to the device.
   b. Write the DESYNC command to the device
   c. Write two pad words to the device to flush the packet processor
8. Reset the TAP controller. The device is desynchronized when the TAP controller moves into the Test-Logic-Reset state.

Table 4-38: **Status Register Readback Command Sequence (JTAG)**

Step	Description	Set and Hold		# of Clocks (TCK)
		TDI	TMS	
1	Clock five 1s on TMS to bring the device to the TLR state.	X	1	5
	Move into the RTI state.	X	0	1
	Move into the SELECT-IR state.	X	1	2
	Move into the SHIFT-IR state.	X	0	2
2	Shift the first 5 bits of the CFG_IN instruction, LSB first.	00101 (CFG_IN)	0	5
	Shift the MSB of the CFG_IN instruction while exiting the SHIFT-IR state.	0	1	1
	Move into the SELECT-DR state.	X	1	2
	Move into the SHIFT-DR state.	X	0	2
3	Shift configuration packets into the CFG_IN data register, MSB first.	a: 0xAA995566 b: 0x2800E001 c: 0x20000000 0x20000000 0x20000000	0	159
	Shift the LSB of the last configuration packet while exiting the SHIFT-DR state.	0	1	1
	Move into the SELECT-IR state.	X	1	3
	Move into the SHIFT-IR state.	X	0	2

Table 4-38: **Status Register Readback Command Sequence (JTAG) *(Continued)***

Step	Description	Set and Hold		# of Clocks (TCK)
		TDI	TMS	
4	Shift the first 5 bits of the CFG_OUT instruction, LSB first.	00100 (CFG_OUT)	0	5
	Shift the MSB of the CFG_OUT instruction while exiting the SHIFT-IR state.	0	1	1
	Move into the SELECT-DR state.	X	1	2
	Move into the SHIFT-DR state.	X	0	2
5	Shift the contents of the STAT register out of the CFG_OUT data register.	0xSSSSSSSS	0	31
	Shift the last bit of the STAT register out of the CFG_OUT data register while exiting the SHIFT-DR state.	S	1	1
	Move into the SELECT-IR state.	X	1	3
	Move into the SHIFT-IR State.	X	0	2
6	Shift the first 5 bits of the CFG_IN instruction, LSB first	00101 (CFG_IN)	0	5
	Shift the MSB of the CFG_IN instruction while exiting SHIFT-IR	0	1	1
	Move into the SELECT-DR state	X	1	2
	Move into the SHIFT-DR state	X	0	2
7	Shift configuration packets into the CFG_IN data register, MSB first.	a: 0x30008001 b: 0x0000000D c: 0x20000000 0x20000000	0	127
	Shift the LSB of the last configuration packet while exiting SHIFT-DR	0	1	1
8	Reset the TAP Controller. The device is desynchronized when the TAP controller moves into the Test-Logic-Reset state.	X	1	5

The packets shifted in to the JTAG CFG_IN register are identical to the packets shifted in through the SelectMAP interface when reading the STAT register through SelectMAP.

Configuration Memory Read Procedure (1149.1 JTAG)

The process for reading configuration memory from the FDRO register through the JTAG interface is similar to the process for reading from other registers. However, additional steps are needed to accommodate frame logic. Configuration data coming from the FDRO register pass through the frame buffer; therefore, the first frame of readback data is "pad data" and should be discarded (refer to the FDRI and FDRO Register description). The 1149.1 JTAG readback flow is recommended for most users.

1. Reset the TAP controller.
2. Shift the JSHUTDOWN instruction into the JTAG Instruction Register.
3. Move into the RTI state, and remain there for 12 TCK cycles to complete the Shutdown sequence. The DONE pin goes Low during the Shutdown sequence.

4. Shift the CFG_IN instruction into the JTAG Instruction Register. The LSB of the CFG_IN instruction is shifted first; the MSB is shifted while moving the TAP controller out of the SHIFT-IR state.
5. Shift packet write commands into the CFG_IN register through the SHIFT-DR state:
   a. Write the Synchronization word to the device.
   b. Write the "Write FAR register" Type 1 packet header to the device.
   c. Write the starting frame address to the FAR (typically `0x00000000`).
   d. Write the "Write CMD register" Type 1 packet header to the device.
   e. Write the "RCRC" command to the device.
   f. Write the "Read FDRO register" Type 1 packet header to the device. The FDRO read length is given by:

      FDRO Read Length = (words per frame) x (frames to read +1)

      One extra frame is read to account for the frame buffer, which produced one pad frame at the beginning of the read.
   g. Write a Type 2 packet header to indicate the number of words to read from the device.
   h. Write exactly two pad words to the device to flush the packet buffer.

   The MSB of all configuration packets sent through the CFG_IN register must be sent first. The LSB is shifted while moving the TAP controller out of the SHIFT-DR state.

   Since an even number of words (2) are sent to the device for this JTAG read operation, it must end with exactly two pad words. If the read operation had consisted of only a Type 1 packet header, instead of the Type 1, Type 2 packet combination that is used here, the sequence would have ended with exactly three pad words. This is because all JTAG read command sequences must consist of an even number of words (see "Packet Processor" on page 318 for details).
6. Shift the CFG_OUT instruction into the JTAG Instruction Register through the SHIFT-IR state. The LSB of the CFG_OUT instruction is shifted first; the MSB is shifted while moving the TAP controller out of the SHIFT-IR state.
7. Shift frame data from the FDRO register through the SHIFT-DR state.
8. Shift the CFG_IN instruction into the JTAG Instruction Register through the SHIFT-IR state.
9. Shift packet write commands into the CFG_IN register through the SHIFT-DR state:
   a. Write a NOOP command.
   b. Write the “Write to CMD register” packet header to the device.
   c. Write the START command.
   d. Write the “Write to CMD register” packet header to the device.
   e. Write the RCRC command.
   f. Write the “Write to CMD register” packet header to the device.
   g. Write the DESYNCH command to the device.
   h. Write two pad words to the device to flush the packet processor.
10. Shift the JSTART instruction into the JTAG Instruction Register.
11. Move to the RTI state and clock the STARTUP sequence by applying a minimum of 12 TCK cycles.
12. Reset the TAP controller.

Table 4-39: **Shutdown Readback Command Sequence (JTAG)**

Step	Description	Set and Hold		# of Clocks (TCK)
		TDI	TMS	
1	Clock five 1s on TMS to bring the device to the TLR state.	X	1	5
	Move into the RTI state.	X	0	1
	Move into the SELECT-IR state.	X	1	2
	Move into the SHIFT-IR State.	X	0	2
2	Shift the first 5 bits of the JSHUTDOWN instruction, LSB first.	01101 (JSHUTDOWN)	0	5
	Shift the MSB of the JSHUTDOWN instruction while exiting the SHIFT-IR state.	0	1	1
	Move into the UPDATE-IR state.	X	1	1
3	Move into the RTI state; remain there for 12 TCK cycles.	X	0	12
	Move into the SELECT-IR state.	X	1	2
	Move into the SHIFT-IR State.	X	0	2
4	Shift the first 5 bits of the CFG_IN instruction, LSB first.	00101 (CFG_IN)	0	5
	Shift the MSB of the CFG_IN instruction while exiting the SHIFT-IR state.	0	1	1
	Move into the SELECT-DR state.	X	1	2
	Move into the SHIFT-DR state.	X	0	2
5	Shift configuration packets into the CFG_IN data register, MSB first.	a: 0xAA995566 b: 0x30002001 c: 0x00000000 d: 0x30008001 e: 0x00000007 f: 0x28006000 g: 0x40002922 h: 0x20000000 0x20000000	0	287
	Shift the LSB of the last configuration packet while exiting the SHIFT-DR state.	0	1	1
	Move into the SELECT-IR state.	X	1	3
	Move into the SHIFT-IR state.	X	0	2
6	Shift the first 5 bits of the CFG_OUT instruction, LSB first	00100 (CFG_OUT)	0	5
	Shift the MSB of the CFG_OUT instruction while exiting the SHIFT-IR state.	0	1	1
	Move into the SELECT-DR state.	X	1	2
	Move into the SHIFT-DR state.	X	0	2

Table 4-39: **Shutdown Readback Command Sequence (JTAG) *(Continued)***

Step	Description	Set and Hold		# of Clocks (TCK)
		TDI	TMS	
7	Shift the contents of the FDRO register out of the CFG_OUT data register.	X	0	number of readback bits – 1
	Shift the last bit of the FDRO register out of the CFG_OUT data register while exiting the SHIFT-DR state.	X	1	1
	Move into the SELECT-IR state.	X	1	3
	Move into the SHIFT-IR state.	X	0	2
8	Shift the first 5 bits of the CFG_IN instruction, LSB first.	00101 (CFG_IN)	0	5
	Shift the MSB of the CFG_IN or CFG_OUT instruction while exiting the SHIFT-IR state.	0	1	1
	Move into the SELECT-DR state.	X	1	2
	Move into the SHIFT-DR state.	X	0	2
9	Shift configuration packets into the CFG_IN data register, MSB first.	a: 0x20000000 b: 0x30008001 c: 0x00000005 d: 0x30008001 e: 0x00000007 f: 0x30008001 g: 0x0000000D h: 0x20000000 0x20000000	0	287
	Shift the LSB of the last configuration packet while exiting the SHIFT-DR state.	0	1	1
	Move into the UPDATE-DR state.	X	1	1
10	Shift the first 5 bits of the JSTART instruction, LSB first.	01100 (JSTART)	0	5
	Shift the MSB of the JSTART instruction while exiting the SHIFT-IR state.	0	1	1
	Move into the UPDATE-IR state.	X	1	1
11	Move into the RTI state; remain there for 12 TCK cycles.	X	0	12
12	End by placing the TAP controller in the TLR state.	X	1	5

Configuration Memory Read Procedure (1532 JTAG)

The IEEE 1532 JTAG readback procedure differs slightly from the IEEE 1149.1 JTAG readback procedure in that readback commands are not sent to the configuration logic via the CFG_IN JTAG register; rather, the ISC_READ JTAG register is used to directly read configuration memory.

At the end of 1532 JTAG readback, CRC Error status must be cleared by issuing a Reset CRC command or writing the correct CRC value to the CRC register. The 1532 JTAG readback procedure is illustrated in Figure 4-44.

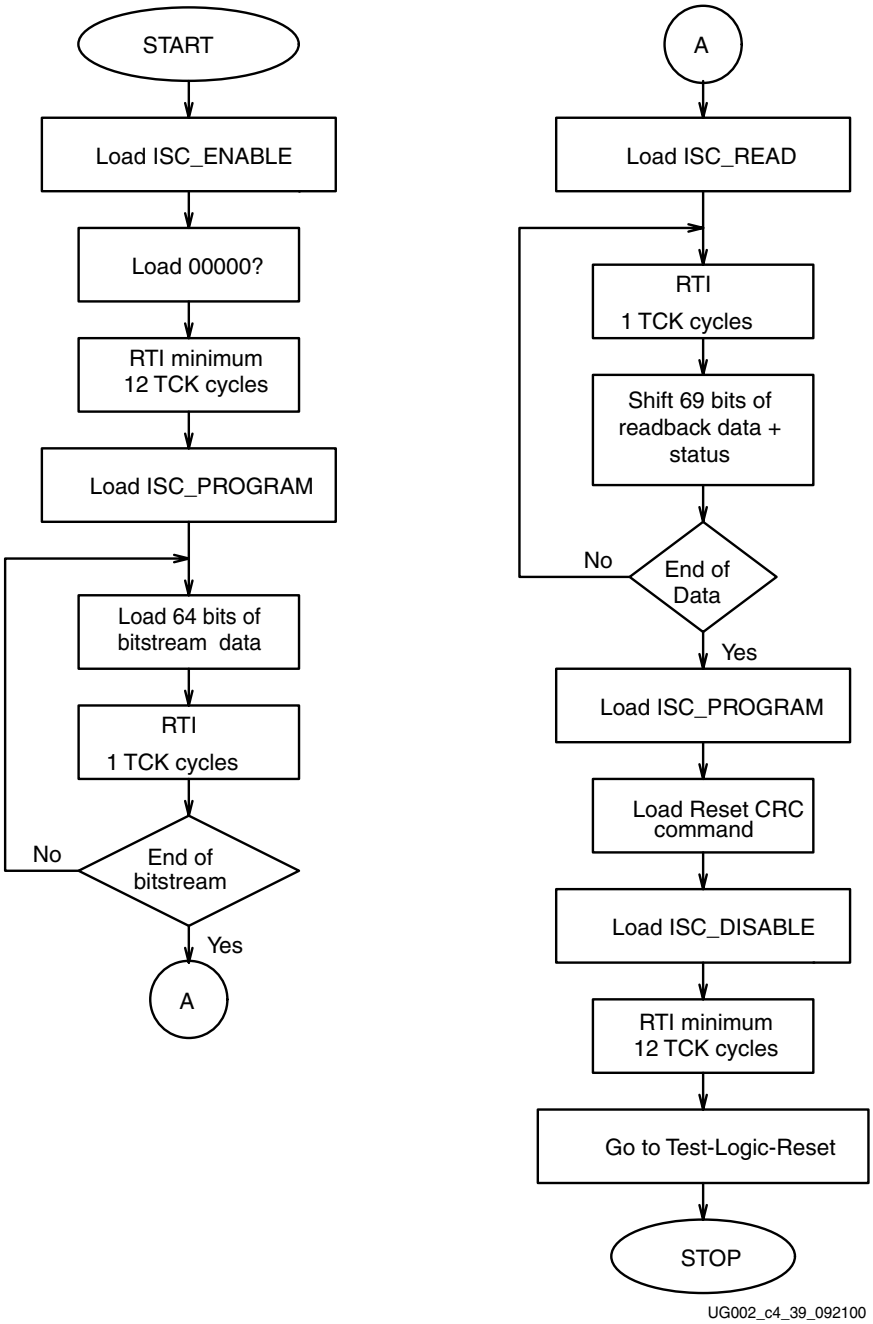


Figure 4-44: **IEEE 1532 JTAG Readback Flow**

Readback Files

The Xilinx Bitstream Generator can produce several files to assist the user in reading and comparing configuration memory. These files are described in Table 4-40.

Table 4-40: **Readback Files**

File Ext.	File Type	BitGen Setting	Description
.rba	ASCII	**-b readback -g readback**	An ASCII file that contains readback commands, rather than configuration commands, and expected readback data where the configuration data would normally be. This file must be used with the .msk file
.rbb	Binary	**-g readback**	Binary version of .rba file. This file must be used with the .msk file.
.rbd	ASCII	**-g readback**	An ASCII file that contains only expected readback data, including the initial pad frame. No commands are included. This file must be used with the .msd file.
.msk	Binary	**-m**	A binary file that contains the same configuration commands as a .bit file, but replaces the contents of the FDRI write packet with mask data that indicate whether the corresponding bits in the .bit file should be compared. If a mask bit is 0, the corresponding bits in the readback data stream should be compared. If a mask bit is 1, the corresponding bit in the readback data stream should be ignored.
.msd	ASCII	**-g readback**	An ASCII file that contains only mask bits. The first bit in the .msd file corresponds to the first bit in the .rbd file. Pad data in the actual readback stream are accounted for in the .msd and .rbd files. If a mask bit is 0, that bit should be verified against the bit stream data. If a mask bit is 1, that bit should not be verified.
.ll	ASCII	**-l**	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.

The design.rba and design.rbb files combine readback commands with expected readback data, whereas the .rbd file contains only expected readback data. Systems that use a .rbd file for readback must store readback commands elsewhere. The actual readback data must be masked against a .msk or .msd mask file, as certain bits within the expected readback stream in the .rba, .rbb, and .rbd files should be ignored.

The readback command set files do not indicate when users must change the SelectMAP or JTAG interface from write to read control; the user must handle this based on the Readback Command Sequences described above.

Verifying Readback Data

The readback data stream contains configuration frame data that are preceded by one frame of pad data, as described in the Configuration Memory Read procedure. The readback stream does not contain any of the commands or packet information found in the configuration bitstream, and no CRC calculation is performed during readback. The readback data stream is shown in Figure 4-45.

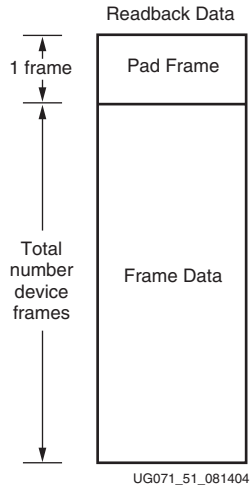


Figure 4-45: **Readback Data Stream**

The readback data stream is verified by comparing it to the original configuration frame data that were programmed into the device. Certain bits within the readback data stream must not be compared, however, since these might correspond to user memory or null memory locations. The location of "don't care" bits in the readback data stream is given by the mask files, the .msk and .msd files. These files have different formats although both convey essentially the same information. Once readback data have been obtained from the device, either of the following comparison procedures can be used:

1. **Compare readback data to the .rbd "golden" readback file, mask using the .msd file.**

   The simplest way to verify the readback data stream is to compare it to the .rbd "golden" readback file, masking readback bits with the .msd file. This approach is simple because there is a 1:1 correspondence between the start of the readback data stream and the start of the .rbd and .msd files, making the task of aligning readback, mask, and expected data easier.

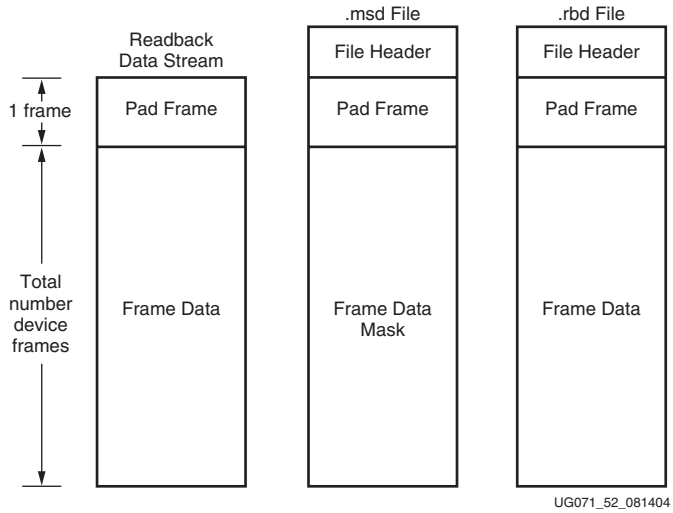


Figure 4-46: **Comparing Readback Data Using the .msd and .rbd Files**

The .rbd and .msd files contain an ASCII representation of the readback and mask data along with a file header that lists the file name, etc. This header information should be ignored or deleted. The ASCII '1s' and '0s' in the .rbd and .msd files correspond to the binary readback data from the device. Take care to interpret these files as text, not

binary sources. Users might wish to convert the .rbd and .msd files to a binary format using a script or text editor, as this can simplify the verify procedure for some systems and reduce the size of the files by a factor of eight.

The drawback to this approach is that in addition to storing the initial configuration bitstream and the .msd file, the golden .rbd file must be stored somewhere, increasing the overall storage requirement.

2. **Compare readback data to the configuration .bit file, mask using the .msk file.**

   Another approach for verifying readback data is to compare the readback data stream to the frame data within the FDRI write in the original configuration bitstream, masking readback bits with the .msk file.

   After sending readback commands to the device, comparison begins by aligning the beginning of the readback frame data to the beginning of the FDRI write in the .bit and .msk files. The comparison ends when the end of the FDRI write is reached.

   This approach requires the least in-system storage space, since only the .bit, .msk, and readback commands must be stored.

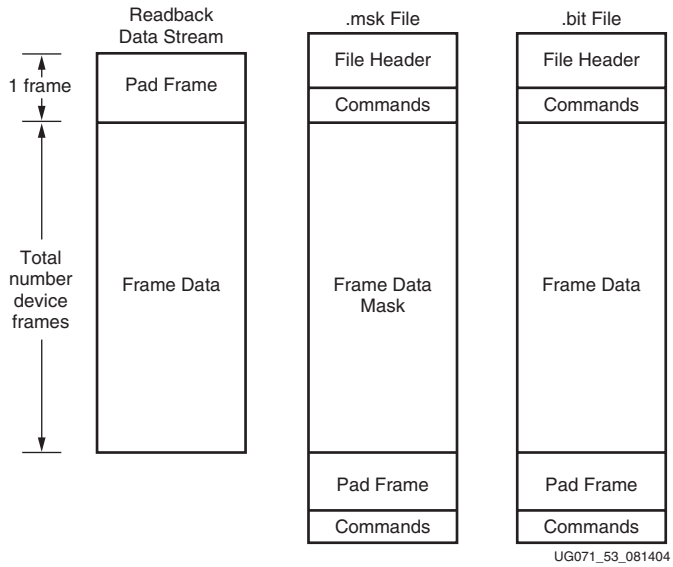


Figure 4-47: **Comparing Readback Data Using the .msk and .bit Files**

The .rba and .rbb files contain expected readback data along with readback command sets. They are intended for use with the .msk file, although they are better suited to readback for Virtex devices (see Xilinx Application Note **XAPP138**) than for Virtex-II devices.

Readback Capture

The configuration memory readback command sequence is identical for both Readback Verify and Readback Capture. However, the Capture sequence requires an additional step to sample internal register values.

Users can sample CLB and IOB registers by instantiating the CAPTURE_VIRTEX2 primitive in their design (Figure 4-48) and asserting the CAP input on that primitive while the design is operating. On the next rising clock edge on the CAPTURE_VIRTEX2 CLK input, the internal GRDBK signal is asserted, storing all CLB and IOB register values into configuration memory cells. These values can then be read out of the device along with the IOB and CLB configuration columns by reading configuration memory through the readback process described above. The capture sequence causes the current register value to be stored to the same memory cell that programs the register's initial state. If the GRESTORE command is sent after the capture sequence has changed the state of this cell, the registers return to an unintended state.

The capture operation can also be performed by writing the GCAPTURE command to the CMD register.

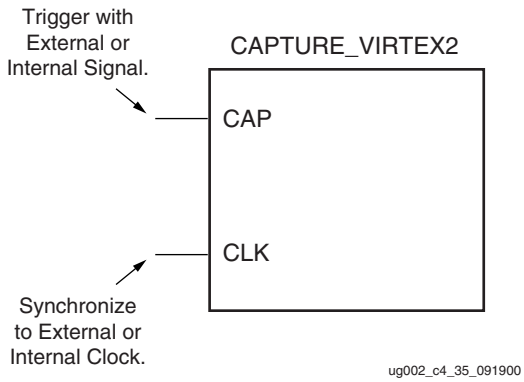


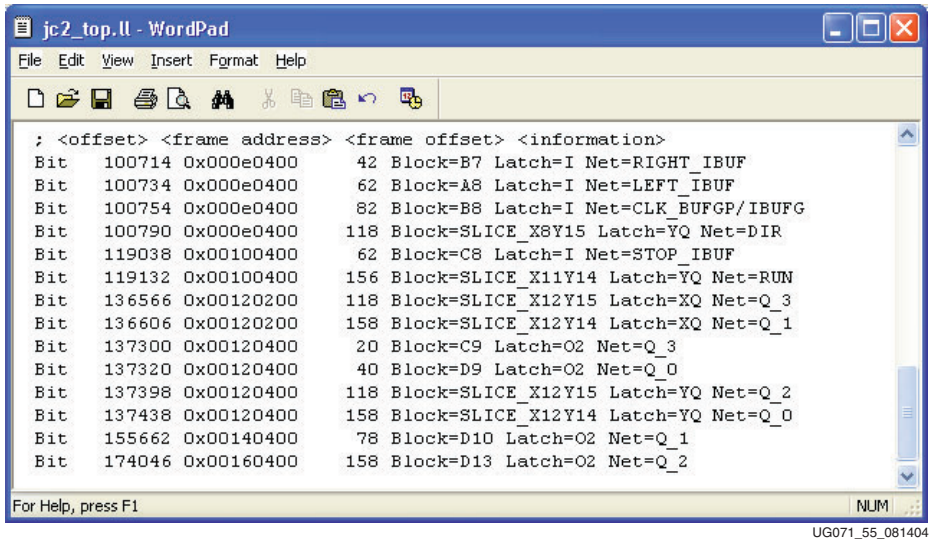
Figure 4-48: **Virtex-II and Virtex-II Pro Library Primitive**

Table 4-41: **Capture Signals**

Signal	Description	Access
GCAPTURE	Captures the state of all slice and IOB registers. Complement of GRESTORE.	"CAP" input on capture block, user controlled. Also CMD register via GCAPTURE command.
GRESTORE	Initializes all registers as configured.	CMD register.

If the CAP signal is left asserted over multiple clock cycles, the Capture cell is updated with the new register value on each rising clock edge. To limit the capture operation to the first rising clock edge, the user can add the ONESHOT attribute to the CAPTURE_VIRTEX2 primitive. More information on the ONESHOT attribute can be found in the Constraints Guide.

Once the configuration memory frames have been read out of the device, the user can pick the captured register values out of the readback data stream. The capture bit locations are given in the logic allocation file (`design.ll`) as described below.



UG071_55_081404

Figure 4-49: **Logic Allocation File Format**

Figure 4-49 shows a snippet from a logic allocation file for the ISE `"jc2_top"` example design. The line from the header comments explaining the line format has been moved to the start of the bit offset data for clarity. The `<offset>` field gives the absolute bit offset from the beginning of the configuration memory (frame address `0x00000000`). The `<frame address>` field gives the frame address that the capture bit is located in, and the `<frame offset>` field gives the bit offset from the start of the frame. The `<information>` field gives the mapping between the bit and the user design—for example, the "DIR" register (Figure 4-49) that is located in Slice X8Y15 is located at bit offset 100790.

Note that captured DFF values, along with LUTRAM and SRL16 values, are stored in their inverted sense.

Using ChipScope ILA

The ChipScope ILA functional verification tool is currently sold separately through the Xilinx web site. This program uses a combination of PC software and instantiated soft cores to capture states of internal signals. This data is read out of the JTAG USER1 scan chain using the MultiLINX cable or a parallel cable. ChipScope ILA supports only the Virtex architecture and allows for the functional verification and debugging of an FPGA configured design.

ChipScope ILA supports the high-speed USB interface to the MultiLINX cable set on Windows 98/2000/XP platforms, and the RS232 connection on Windows 95/98/2000/NT/XP platforms. UNIX support is not available. More details are available under ChipScope ILA at: **www.xilinx.com**

XILINX®

Chapter 5

PCB Design Considerations

Summary

This chapter covers the following topics:

- Pinout Information
- Pinout Diagrams
- Package Specifications
- Flip-Chip Packages
- Thermal Data
- Printed Circuit Board Considerations
- Board Routability Guidelines
- Power Consumption
- IBIS Models
- BSDL and Boundary Scan Models

Pinout Information

Introduction

This section describes the pinouts for Virtex-II devices in the following packages:

- CS144: wire-bond chip-scale ball grid array (BGA) of 0.80 mm pitch
- FG256, FG456, and FG676: wire-bond fine-pitch BGA of 1.00 mm pitch
- FF896, FF1152, FF1517: flip-chip fine-pitch BGA of 1.00 mm pitch
- BG575 and BG728: wire-bond BGA of 1.27 mm pitch
- BF957: flip-chip BGA of 1.27 mm pitch

All of the devices supported in a particular package are pinout compatible and are listed in the same table (one table per package). In addition, the FG456 and FG676 packages are compatible, as are the FF896 and FF1152 packages. Pins that are not available for the smallest devices are listed in right-hand columns.

Each device is split into eight I/O banks to allow for flexibility in the choice of I/O standards (see the **Virtex-II Data Sheet (DS031)**). Global pins, including JTAG, configuration, and power/ground pins, are listed at the end of each table. Table 5-1 provides definitions for all pin types.

The FG256 pinouts (Table 5-2) is included as an example. All Virtex-II pinout tables are available on the distribution CD-ROM, or on the **www.xilinx.com** website.

Pin Definitions

Table 5-1 provides a description of each pin type listed in Virtex-II pinout tables.

Table 5-1: **Virtex-II Pin Definitions**

Pin Name	Direction	Description
User I/O Pins		
IO_LXXY_#	Input/Output	All user I/O pins are capable of differential signalling and can implement LVDS, ULVDS, BLVDS, LVPECL, or LDT pairs. Each user I/O is labeled "**IO_LXXY_#**", where: **IO** indicates a user I/O pin. **LXXY** indicates a differential pair, with **XX** a unique pair in the bank and **Y = P/N** for the positive and negative sides of the differential pair. **#** indicates the bank number (0 through 7)
Dual-Function Pins		
IO_LXXY_#/ZZZ		The dual-function pins are labelled "**IO_LXXY_#/ZZZ**", where **ZZZ** can be one of the following pins: Per Bank - **VRP, VRN,** or **VREF** Globally - **GCLKX(S/P), BUSY/DOUT, INIT_B, DIN/D0 – D7, RDWR_B,** or **CS_B**
With /ZZZ:		
DIN / D0, D1, D2, D3, D4, D5, D6, D7	Input/Output	In SelectMAP mode, D0 through D7 are configuration data pins. These pins become user I/Os after configuration, unless the SelectMAP port is retained. In bit-serial modes, DIN (D0) is the single-data input. This pin becomes a user I/O after configuration.
CS_B	Input	In SelectMAP mode, this is the active-low Chip Select signal. The pin becomes a user I/O after configuration, unless the SelectMAP port is retained.
RDWR_B	Input	In SelectMAP mode, this is the active-low Write Enable signal. The pin becomes a user I/O after configuration, unless the SelectMAP port is retained.

Table 5-1: **Virtex-II Pin Definitions *(Continued)***

Pin Name	Direction	Description
BUSY/DOUT	Output	In SelectMAP mode, BUSY controls the rate at which configuration data is loaded. The pin becomes a user I/O after configuration, unless the SelectMAP port is retained. In bit-serial modes, DOUT provides preamble and configuration data to down-stream devices in a daisy-chain. The pin becomes a user I/O after configuration.
INIT_B	Bidirectional (open-drain)	When Low, this pin indicates that the configuration memory is being cleared. When held Low, the start of configuration is delayed. During configuration, a Low on this output indicates that a configuration data error has occurred. The pin becomes a user I/O after configuration.
GCLKx (S/P)	Input	These are clock input pins that connect to Global Clock Buffers. These pins become regular user I/Os when not needed for clocks.
VRP	Input	This pin is for the DCI voltage reference resistor of P transistor (per bank).
VRN	Input	This pin is for the DCI voltage reference resistor of N transistor (per bank).
ALT_VRP	Input	This is the alternative pin for the DCI voltage reference resistor of P transistor.
ALT_VRN	Input	This is the alternative pin for the DCI voltage reference resistor of N transistor.
V_{REF}	Input	These are input threshold voltage pins. They become user I/Os when an external threshold voltage is not needed (per bank).
Dedicated Pins[1]		
CCLK	Input/Output	Configuration clock. Output in Master mode or Input in Slave mode.
PROG_B	Input	Active Low asynchronous reset to configuration logic. This pin has a permanent weak pull-up resistor.
DONE	Input/Output	DONE is a bidirectional signal with an optional internal pull-up resistor. As an output, this pin indicates completion of the configuration process. As an input, a Low level on DONE can be configured to delay the start-up sequence.
M2, M1, M0	Input	Configuration mode selection.
HSWAP_EN	Input	Enable I/O pullups during configuration.
TCK	Input	Boundary Scan Clock.
TDI	Input	Boundary Scan Data Input.
TDO	Output	Boundary Scan Data Output.
TMS	Input	Boundary Scan Mode Select.
PWRDWN_B	Input	Active Low power down pin. PWRDWN_B should always be pulled High, which is its default value. Driving this pin Low can affect device operation and configuration. PWRDWN_B does not require an external pull-up or pull-down.
Other Pins		
DXN, DXP	N/A	Temperature-sensing diode pins (Anode: DXP, Cathode: DXN).
V_{BATT}	Input	Decryptor key memory backup supply. Connect to V_{CCAUX} or GND if battery is not used.
RSVD	N/A	Reserved pin - do not connect.
V_{CCO}	Input	Power-supply pins for the output drivers (per bank).
V_{CCAUX}	Input	Power-supply pins for auxiliary circuits.
V_{CCINT}	Input	Power-supply pins for the internal core logic.
GND	Input	Ground.

Notes:

1. All dedicated pins (JTAG and configuration) are powered by V_{CCAUX} (independent of the bank V_{CCO} voltage).

FG256 Fine-Pitch BGA Package

As shown in Table 5-2, XC2V40, XC2V80, XC2V250, XC2V500, and XC2V1000 Virtex-II devices are available in the FG256 fine-pitch BGA package. Pins in the XC2V250, XC2V500, and XC2V1000 devices are the same. The No Connect column shows pin differences for the XC2V40 and XC2V80 devices.

The FG256 pinout information (Table 5-2) is included as an example. All Virtex-II pinout tables are available on the distribution CD-ROM, or on the web (http://www.xilinx.com).

Table 5-2: **FG256 BGA — XC2V40, XC2V80, XC2V250, XC2V500, and XC2V1000**

Bank	Pin Description	Pin Number	No Connect in XC2V40	No Connect in XC2V80
0	IO_L01N_0	C4		
0	IO_L01P_0	B4		
0	IO_L02N_0	D5		
0	IO_L02P_0	C5		
0	IO_L03N_0/VRP_0	B5		
0	IO_L03P_0/VRN_0	A5		
0	IO_L04N_0/VREF_0	D6	NC	NC
0	IO_L04P_0	C6	NC	NC
0	IO_L05N_0	B6	NC	NC
0	IO_L05P_0	A6	NC	NC
0	IO_L92N_0	E6	NC	NC
0	IO_L92P_0	E7	NC	NC
0	IO_L93N_0	D7	NC	NC
0	IO_L93P_0	C7	NC	NC
0	IO_L94N_0/VREF_0	B7		
0	IO_L94P_0	A7		
0	IO_L95N_0/GCLK7P	D8		
0	IO_L95P_0/GCLK6S	C8		
0	IO_L96N_0/GCLK5P	B8		
0	IO_L96P_0/GCLK4S	A8		
1	IO_L96N_1/GCLK3P	A9		
1	IO_L96P_1/GCLK2S	B9		
1	IO_L95N_1/GCLK1P	C9		
1	IO_L95P_1/GCLK0S	D9		
1	IO_L94N_1	A10		
1	IO_L94P_1/VREF_1	B10		
1	IO_L93N_1	C10	NC	NC
1	IO_L93P_1	D10	NC	NC
1	IO_L92N_1	E10	NC	NC
1	IO_L92P_1	E11	NC	NC
1	IO_L05N_1	A11	NC	NC
1	IO_L05P_1	B11	NC	NC

Table 5-2: **FG256 BGA — XC2V40, XC2V80, XC2V250, XC2V500, and XC2V1000**

Bank	Pin Description	Pin Number	No Connect in XC2V40	No Connect in XC2V80
1	IO_L04N_1	C11	NC	NC
1	IO_L04P_1/VREF_1	D11	NC	NC
1	IO_L03N_1/VRP_1	A12		
1	IO_L03P_1/VRN_1	B12		
1	IO_L02N_1	C12		
1	IO_L02P_1	D12		
1	IO_L01N_1	B13		
1	IO_L01P_1	C13		
2	IO_L01N_2	C16		
2	IO_L01P_2	D16		
2	IO_L02N_2/VRP_2	D14		
2	IO_L02P_2/VRN_2	D15		
2	IO_L03N_2	E13		
2	IO_L03P_2/VREF_2	E14		
2	IO_L04N_2	E15	NC	
2	IO_L04P_2	E16	NC	
2	IO_L06N_2	F13	NC	
2	IO_L06P_2	F14	NC	
2	IO_L43N_2	F15	NC	NC
2	IO_L43P_2	F16	NC	NC
2	IO_L45N_2	F12	NC	NC
2	IO_L45P_2/VREF_2	G12	NC	NC
2	IO_L91N_2	G13	NC	
2	IO_L91P_2	G14	NC	
2	IO_L93N_2	G15	NC	
2	IO_L93P_2/VREF_2	G16	NC	
2	IO_L94N_2	H13		
2	IO_L94P_2	H14		
2	IO_L96N_2	H15		
2	IO_L96P_2	H16		
3	IO_L96N_3	J16		
3	IO_L96P_3	J15		
3	IO_L94N_3	J14		
3	IO_L94P_3	J13		
3	IO_L93N_3/VREF_3	K16	NC	
3	IO_L93P_3	K15	NC	
3	IO_L91N_3	K14	NC	

Table 5-2: **FG256 BGA — XC2V40, XC2V80, XC2V250, XC2V500, and XC2V1000**

Bank	Pin Description	Pin Number	No Connect in XC2V40	No Connect in XC2V80
3	IO_L91P_3	K13	NC	
3	IO_L45N_3/VREF_3	K12	NC	NC
3	IO_L45P_3	L12	NC	NC
3	IO_L43N_3	L16	NC	NC
3	IO_L43P_3	L15	NC	NC
3	IO_L06N_3	L14	NC	
3	IO_L06P_3	L13	NC	
3	IO_L04N_3	M16	NC	
3	IO_L04P_3	M15	NC	
3	IO_L03N_3/VREF_3	M14		
3	IO_L03P_3	M13		
3	IO_L02N_3/VRP_3	N15		
3	IO_L02P_3/VRN_3	N14		
3	IO_L01N_3	N16		
3	IO_L01P_3	P16		
4	IO_L01N_4/DOUT	T14		
4	IO_L01P_4/INIT_B	T13		
4	IO_L02N_4/D0	P13		
4	IO_L02P_4/D1	R13		
4	IO_L03N_4/D2/ALT_VRP_4	N12		
4	IO_L03P_4/D3/ALT_VRN_4	P12		
4	IO_L04N_4/VREF_4	R12	NC	NC
4	IO_L04P_4	T12	NC	NC
4	IO_L05N_4/VRP_4	N11	NC	NC
4	IO_L05P_4/VRN_4	P11	NC	NC
4	IO_L91N_4/VREF_4	R11	NC	NC
4	IO_L91P_4	T11	NC	NC
4	IO_L92N_4	M11	NC	NC
4	IO_L92P_4	M10	NC	NC
4	IO_L93N_4	N10	NC	NC
4	IO_L93P_4	P10	NC	NC
4	IO_L94N_4/VREF_4	R10		
4	IO_L94P_4	T10		
4	IO_L95N_4/GCLK3S	N9		
4	IO_L95P_4/GCLK2P	P9		
4	IO_L96N_4/GCLK1S	R9		
4	IO_L96P_4/GCLK0P	T9		

Table 5-2: **FG256 BGA — XC2V40, XC2V80, XC2V250, XC2V500, and XC2V1000**

Bank	Pin Description	Pin Number	No Connect in XC2V40	No Connect in XC2V80
5	IO_L96N_5/GCLK7S	T8		
5	IO_L96P_5/GCLK6P	R8		
5	IO_L95N_5/GCLK5S	P8		
5	IO_L95P_5/GCLK4P	N8		
5	IO_L94N_5	T7		
5	IO_L94P_5/VREF_5	R7		
5	IO_L93N_5	P7	NC	NC
5	IO_L93P_5	N7	NC	NC
5	IO_L92N_5	M7	NC	NC
5	IO_L92P_5	M6	NC	NC
5	IO_L91N_5	T6	NC	NC
5	IO_L91P_5/VREF_5	R6	NC	NC
5	IO_L05N_5/VRP_5	P6	NC	NC
5	IO_L05P_5/VRN_5	N6	NC	NC
5	IO_L04N_5	T5	NC	NC
5	IO_L04P_5/VREF_5	R5	NC	NC
5	IO_L03N_5/D4/ALT_VRP_5	P5		
5	IO_L03P_5/D5/ALT_VRN_5	N5		
5	IO_L02N_5/D6	R4		
5	IO_L02P_5/D7	P4		
5	IO_L01N_5/RDWR_B	T4		
5	IO_L01P_5/CS_B	T3		
6	IO_L01P_6	P1		
6	IO_L01N_6	N1		
6	IO_L02P_6/VRN_6	N3		
6	IO_L02N_6/VRP_6	N2		
6	IO_L03P_6	M4		
6	IO_L03N_6/VREF_6	M3		
6	IO_L04P_6	M2	NC	
6	IO_L04N_6	M1	NC	
6	IO_L06P_6	L4	NC	
6	IO_L06N_6	L3	NC	
6	IO_L43P_6	L2	NC	NC
6	IO_L43N_6	L1	NC	NC
6	IO_L45P_6	L5	NC	NC
6	IO_L45N_6/VREF_6	K5	NC	NC
6	IO_L91P_6	K4	NC	
6	IO_L91N_6	K3	NC	

Table 5-2: **FG256 BGA — XC2V40, XC2V80, XC2V250, XC2V500, and XC2V1000**

Bank	Pin Description	Pin Number	No Connect in XC2V40	No Connect in XC2V80
6	IO_L93P_6	K2	NC	
6	IO_L93N_6/VREF_6	K1	NC	
6	IO_L94P_6	J4		
6	IO_L94N_6	J3		
6	IO_L96P_6	J2		
6	IO_L96N_6	J1		
7	IO_L96P_7	H1		
7	IO_L96N_7	H2		
7	IO_L94P_7	H3		
7	IO_L94N_7	H4		
7	IO_L93P_7/VREF_7	G1	NC	
7	IO_L93N_7	G2	NC	
7	IO_L91P_7	G3	NC	
7	IO_L91N_7	G4	NC	
7	IO_L45P_7/VREF_7	G5	NC	NC
7	IO_L45N_7	F5	NC	NC
7	IO_L43P_7	F1	NC	NC
7	IO_L43N_7	F2	NC	NC
7	IO_L06P_7	F3	NC	
7	IO_L06N_7	F4	NC	
7	IO_L04P_7	E1	NC	
7	IO_L04N_7	E2	NC	
7	IO_L03P_7/VREF_7	E3		
7	IO_L03N_7	E4		
7	IO_L02P_7/VRN_7	D2		
7	IO_L02N_7/VRP_7	D3		
7	IO_L01P_7	D1		
7	IO_L01N_7	C1		
0	VCCO_0	F8		
0	VCCO_0	F7		
0	VCCO_0	E8		
1	VCCO_1	F10		
1	VCCO_1	F9		
1	VCCO_1	E9		
2	VCCO_2	H12		
2	VCCO_2	H11		
2	VCCO_2	G11		

Table 5-2: **FG256 BGA — XC2V40, XC2V80, XC2V250, XC2V500, and XC2V1000**

Bank	Pin Description	Pin Number	No Connect in XC2V40	No Connect in XC2V80
3	VCCO_3	K11		
3	VCCO_3	J12		
3	VCCO_3	J11		
4	VCCO_4	M9		
4	VCCO_4	L10		
4	VCCO_4	L9		
5	VCCO_5	M8		
5	VCCO_5	L8		
5	VCCO_5	L7		
6	VCCO_6	K6		
6	VCCO_6	J6		
6	VCCO_6	J5		
7	VCCO_7	H6		
7	VCCO_7	H5		
7	VCCO_7	G6		
NA	CCLK	P15		
NA	PROG_B	A2		
NA	DONE	R14		
NA	M0	T2		
NA	M1	P2		
NA	M2	R3		
NA	HSWAP_EN	B3		
NA	TCK	A15		
NA	TDI	C2		
NA	TDO	C15		
NA	TMS	B14		
NA	PWRDWN_B	T15		
NA	RSVD	A4		
NA	RSVD	A3		
NA	VBATT	A14		
NA	RSVD	A13		
NA	VCCAUX	R16		
NA	VCCAUX	R1		
NA	VCCAUX	B16		
NA	VCCAUX	B1		
NA	VCCINT	N13		
NA	VCCINT	N4		

Table 5-2: **FG256 BGA — XC2V40, XC2V80, XC2V250, XC2V500, and XC2V1000**

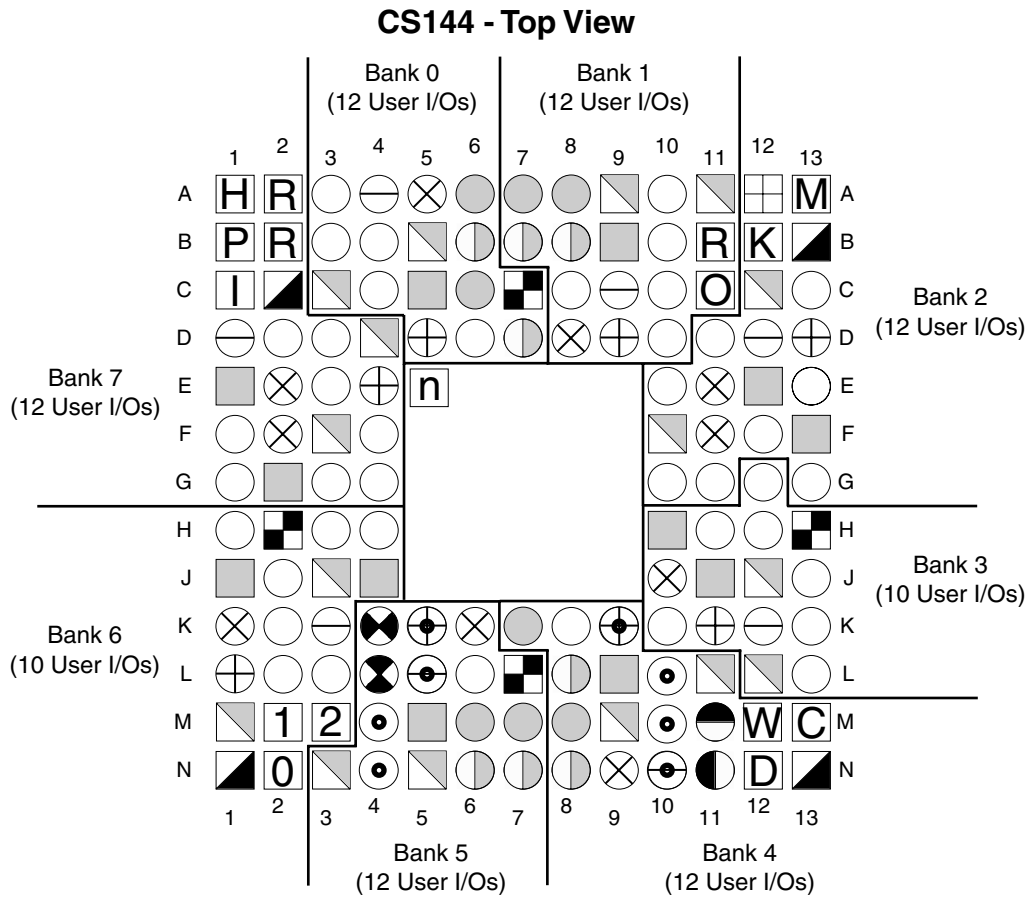
Bank	Pin Description	Pin Number	No Connect in XC2V40	No Connect in XC2V80
NA	VCCINT	M12		
NA	VCCINT	M5		
NA	VCCINT	E12		
NA	VCCINT	E5		
NA	VCCINT	D13		
NA	VCCINT	D4		
NA	GND	T16		
NA	GND	T1		
NA	GND	R15		
NA	GND	R2		
NA	GND	P14		
NA	GND	P3		
NA	GND	L11		
NA	GND	L6		
NA	GND	K10		
NA	GND	K9		
NA	GND	K8		
NA	GND	K7		
NA	GND	J10		
NA	GND	J9		
NA	GND	J8		
NA	GND	J7		
NA	GND	H10		
NA	GND	H9		
NA	GND	H8		
NA	GND	H7		
NA	GND	G10		
NA	GND	G9		
NA	GND	G8		
NA	GND	G7		
NA	GND	F11		
NA	GND	F6		
NA	GND	C14		
NA	GND	C3		
NA	GND	B15		
NA	GND	B2		
NA	GND	A16		
NA	GND	A1		

Pinout Diagrams

This section contains pinout diagrams for the largest device in each of the following Virtex-II packages:

- "CS144 Chip-Scale BGA Composite Pinout Diagram (XC2V250)" on page 370
- "FG256 Fine-Pitch BGA Composite Pinout Diagram (XC2V1000)" on page 371
  - FG256 Bank Information
  - FG256 Dedicated Pins
- "FG456 Fine-Pitch BGA Composite Pinout Diagram (XC2V1000)" on page 375
  - FG456 Bank Information
  - FG456 Dedicated Pins
- "FG676 Fine-Pitch BGA Composite Pinout Diagram (XC2V3000)" on page 379
  - FG676 Bank Information
  - FG676 Dedicated Pins
- "BG575 Standard BGA Composite Pinout Diagram (XC2V2000)" on page 383
  - BG575 Bank Information
  - BG575 Dedicated Pins
- "BG728 Standard BGA Composite Pinout Diagram (XC2V3000)" on page 387
  - BG728 Bank Information
  - BG728 Dedicated Pins
- "FF896 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V2000)" on page 391
  - FF896 Bank Information
  - FF896 Dedicated Pins
- "FF1152 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V8000)" on page 395
  - FF1152 Bank Information
  - FF1152 Dedicated Pins
- "FF1517 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V8000)" on page 399
  - FF1517 Bank Information
  - FF1517 Dedicated Pins
- "BF957 Flip-Chip BGA Composite Pinout Diagram (XC2V6000)" on page 403
  - BF957 Bank Information
  - BF957 Dedicated Pins
- "FG456 - FG676 Pinout Compatibility Diagram" on page 406
- "FF896 - FF1152 Pinout Compatibility Diagram" on page 407

CS144 Chip-Scale BGA Composite Pinout Diagram (XC2V250)

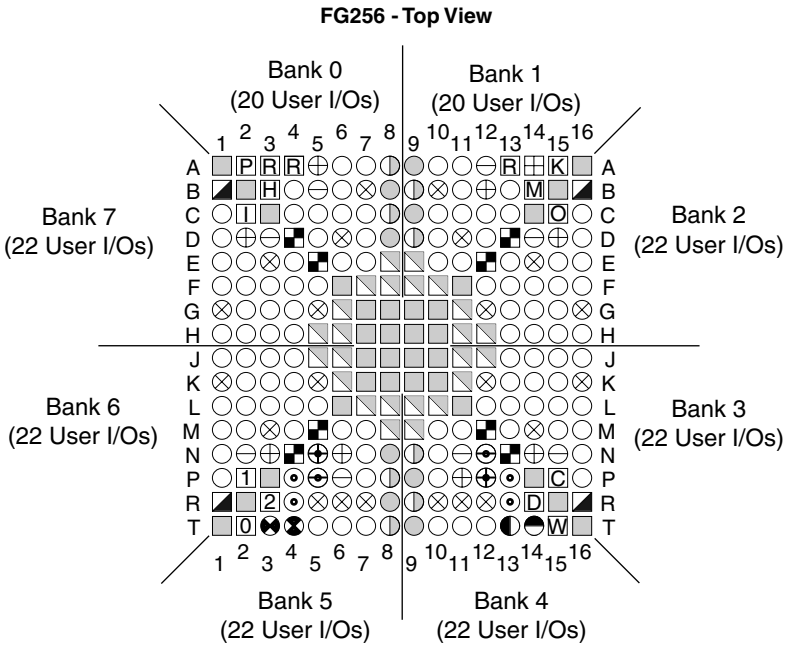


User I/O Pins	Dedicated Pins	
○ IO_LXXY_#	C CCLK	⊞ VBATT
Dual-Purpose Pins:	P PROG_B	R RSVD
⊙ DIN/D0-D7	D DONE	▨ VCCO
⊗ CS_B	2 1 0 M2, M1, M0	◪ VCCAUX
⊗ RDWR_B	H HSWAP_EN	▪ VCCINT
◓ BUSY/DOUT	K TCK	▨ GND
◐ INIT_B	I TDI	n NO CONNECT
● GCLKx (P)	O TDO	
◑ GCLKx (S)	M TMS	
⊖ VRP	W PWRDWN_B	
⊕ VRN		
⊗ VREF		
Triple-Purpose Pins:		
⊙ D2, D4/ALT_VRP		
⊕ D3, D5/ALT_VRN		

ug002_c4_46_031501

Figure 5-1: **CS144 Chip-Scale BGA Composite Pinout Diagram (XC2V250)**

FG256 Fine-Pitch BGA Composite Pinout Diagram (XC2V1000)

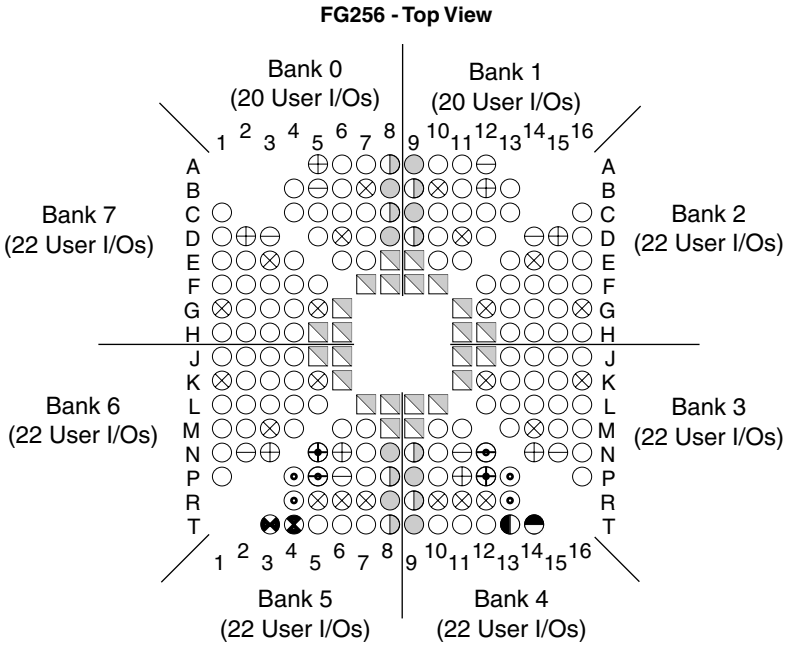


User I/O Pins		Dedicated Pins			
○	IO_LXXY_#	C	CCLK		
Dual-Purpose Pins:		P	PROG_B		
⊙	DIN/D0-D7	D	DONE	⊞	VBATT
⊠	CS_B	2 1 0	M2, M1, M0	R	RSVD
⧗	RDWR_B	H	HSWAP_EN	▨	VCCO
◓	BUSY/DOUT	K	TCK	◪	VCCAUX
◐	INIT_B	I	TDI	▪	VCCINT
●	GCLKx (P)	O	TDO	■	GND
◑	GCLKx (S)	M	TMS	n	NO CONNECT
⊖	VRP	W	PWRDWN_B		
⊕	VRN				
⊗	VREF				
Triple-Purpose Pins:					
⊙	D2, D4/ALT_VRP				
⊕	D3, D5/ALT_VRN				

ug002_c4_47_031501

Figure 5-2: **FG256 Fine-Pitch BGA Composite Pinout Diagram (XC2V1000)**

FG256 Bank Information



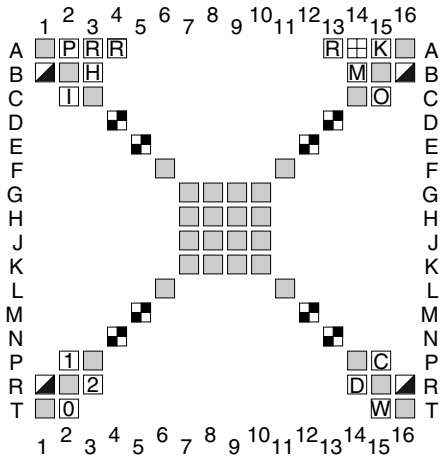
User I/O Pins	Dedicated Pins	
○ IO_LXXY_#		
Dual-Purpose Pins:		
⊙ DIN/D0-D7		
⊗ CS_B		
⊗ RDWR_B		▨ VCCO
◓ BUSY/DOUT		
◐ INIT_B		
● GCLKx (P)		
◑ GCLKx (S)		
⊖ VRP		
⊕ VRN		
⊗ VREF		
Triple-Purpose Pins:		
⊙ D2, D4/ALT_VRP		
⊕ D3, D5/ALT_VRN		

ug002_c4_47b_031501

Figure 5-3: **FG256 Bank Information**

FG256 Dedicated Pins

FG256 - Top View

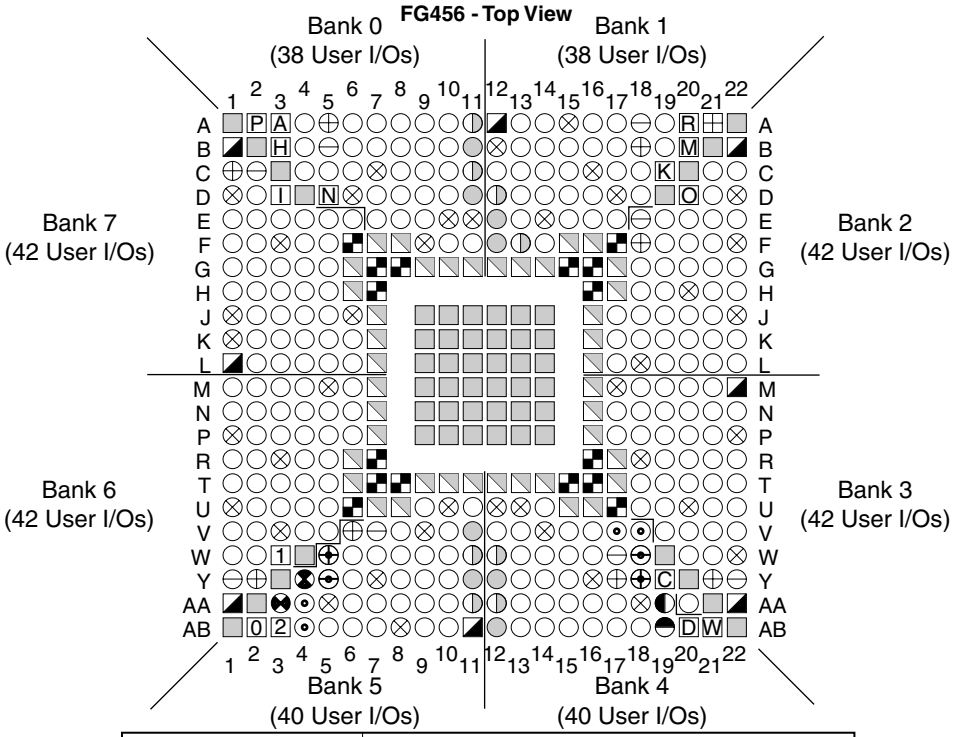


User I/O Pins	Dedicated Pins	
	C CCLK	
	P PROG_B	
	D DONE	VBATT
	2 1 0 M2, M1, M0	R RSVD
	H HSWAP_EN	
	K TCK	VCCAUX
	I TDI	VCCINT
	O TDO	GND
	M TMS	n NO CONNECT
	W PWRDWN_B	

ug002_c4_47c_120400

Figure 5-4: **FG256 Dedicated Pins**

FG456 Fine-Pitch BGA Composite Pinout Diagram (XC2V1000)



User I/O Pins		Dedicated Pins			
○	IO_LXXY_#	C	CCLK	N	DXN
Dual-Purpose Pins:		P	PROG_B	A	DXP
⊙	DIN/D0-D7	D	DONE	⊞	VBATT
⊠	CS_B	2 1 0	M2, M1, M0	R	RSVD
⊠	RDWR_B	H	HSWAP_EN	▨	VCCO
◓	BUSY/DOUT	K	TCK	◢	VCCAUX
◐	INIT_B	I	TDI	▚	VCCINT
●	GCLKx (P)	O	TDO	■	GND
◑	GCLKx (S)	M	TMS	n	NO CONNECT
⊖	VRP	W	PWRDWN_B		
⊕	VRN				
⊗	VREF				
Triple-Purpose Pins:					
⊖	D2, D4/ALT_VRP				
⊕	D3, D5/ALT_VRN				

ug002_c4_48_031501

Figure 5-5: **FG456 Fine-Pitch BGA Composite Pinout Diagram (XC2V1000)**

FG456 Bank Information

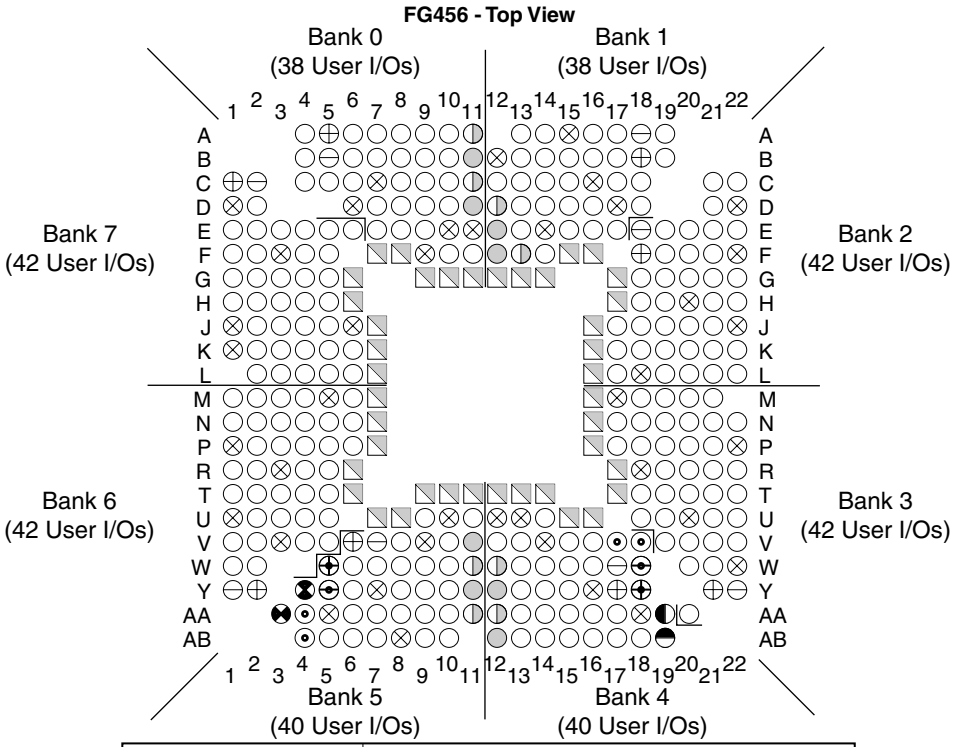
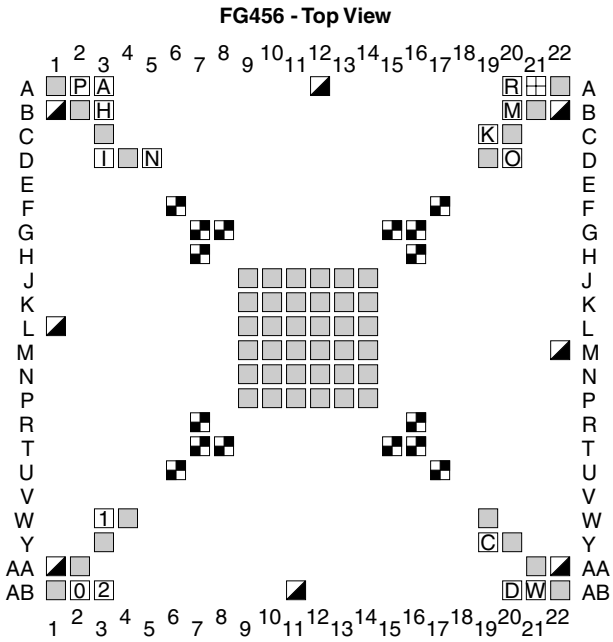


Figure 5-6: **FG456 Bank Information**

FG456 Dedicated Pins

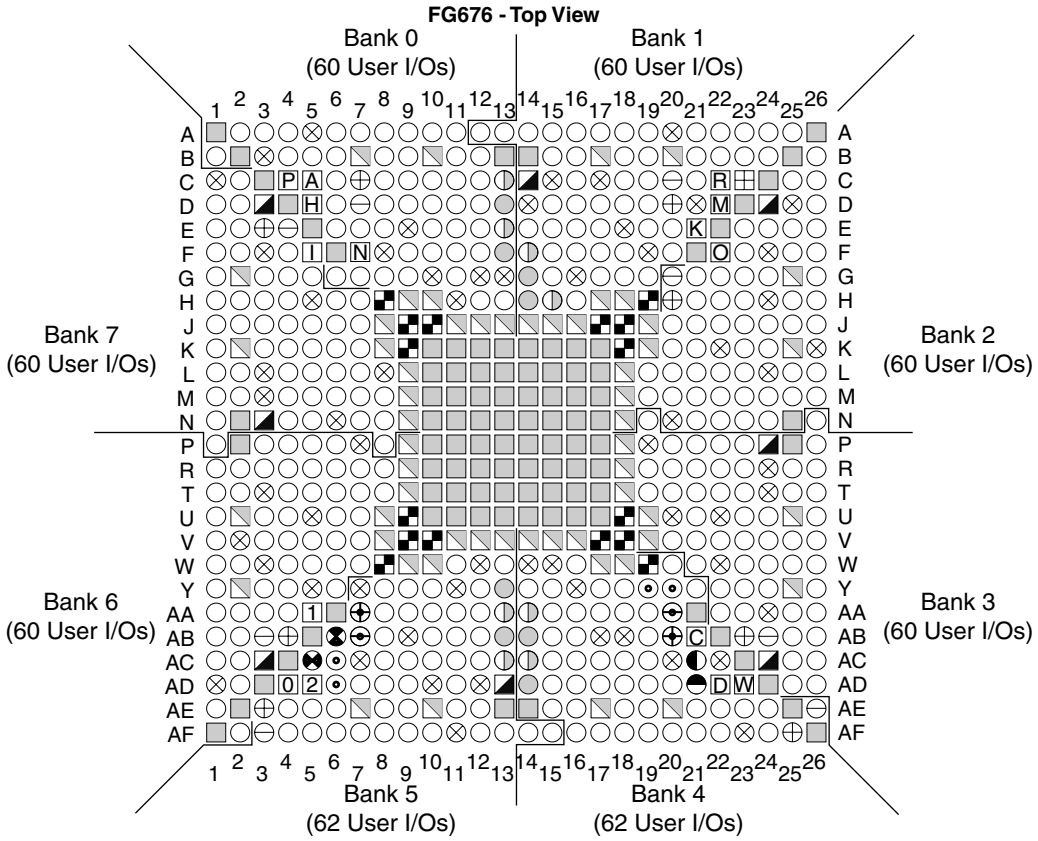


User I/O Pins	Dedicated Pins	
	C CCLK	N DXN
	P PROG_B	A DXP
	D DONE	VBATT
	2 1 0 M2, M1, M0	R RSVD
	H HSWAP_EN	
	K TCK	VCCAUX
	I TDI	VCCINT
	O TDO	GND
	M TMS	n NO CONNECT
	W PWRDWN_B	

ug002_c4_48c_120400

Figure 5-7: **FG456 Dedicated Pins**

FG676 Fine-Pitch BGA Composite Pinout Diagram (XC2V3000)

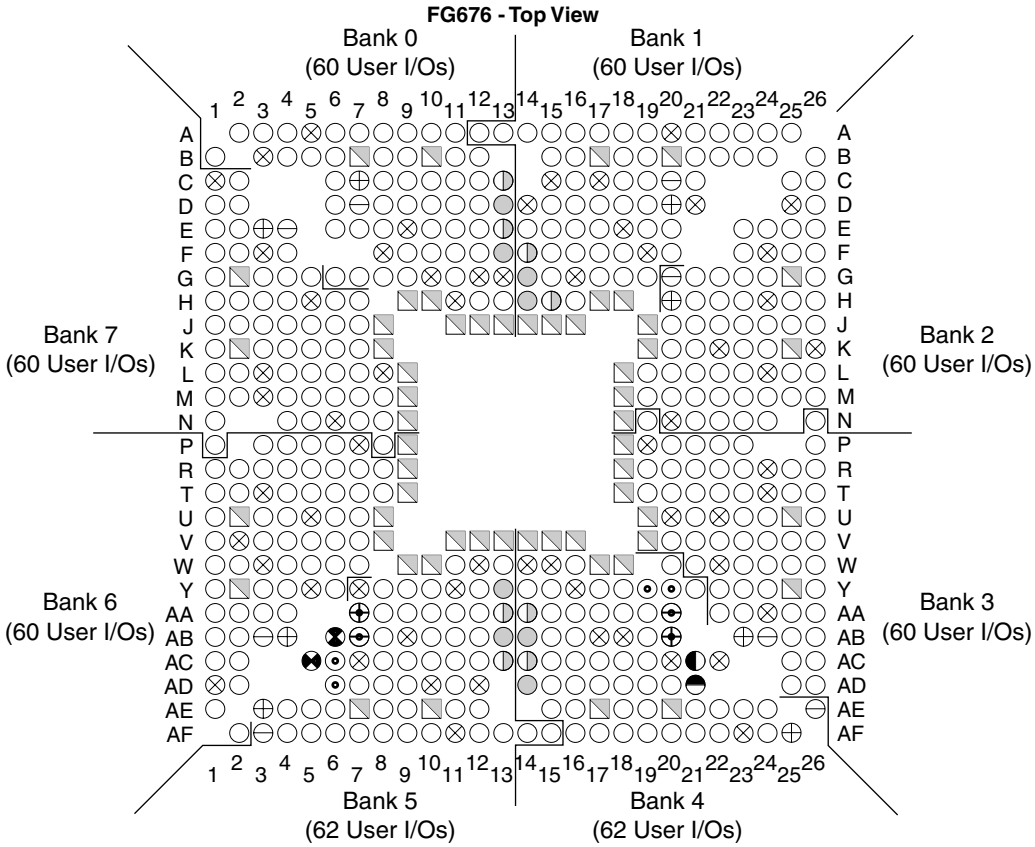


User I/O Pins		Dedicated Pins			
○	IO_LXXY_#	C	CCLK	N	DXN
Dual-Purpose Pins:		P	PROG_B	A	DXP
⊙	DIN/D0-D7	D	DONE	⊞	VBATT
⊗	CS_B	2 1 0	M2, M1, M0	R	RSVD
⊗	RDWR_B	H	HSWAP_EN	▨	VCCO
◓	BUSY/DOUT	K	TCK	◪	VCCAUX
◐	INIT_B	I	TDI	▩	VCCINT
●	GCLKx (P)	O	TDO	■	GND
◑	GCLKx (S)	M	TMS	n	NO CONNECT
⊖	VRP	W	PWRDWN_B		
⊕	VRN				
⊗	VREF				
Triple-Purpose Pins:					
⊛	D2, D4/ALT_VRP				
⊕	D3, D5/ALT_VRN				

ug002_c4_49_031501

Figure 5-8: **FG676 Fine-Pitch BGA Composite Pinout Diagram (XC2V3000)**

FG676 Bank Information



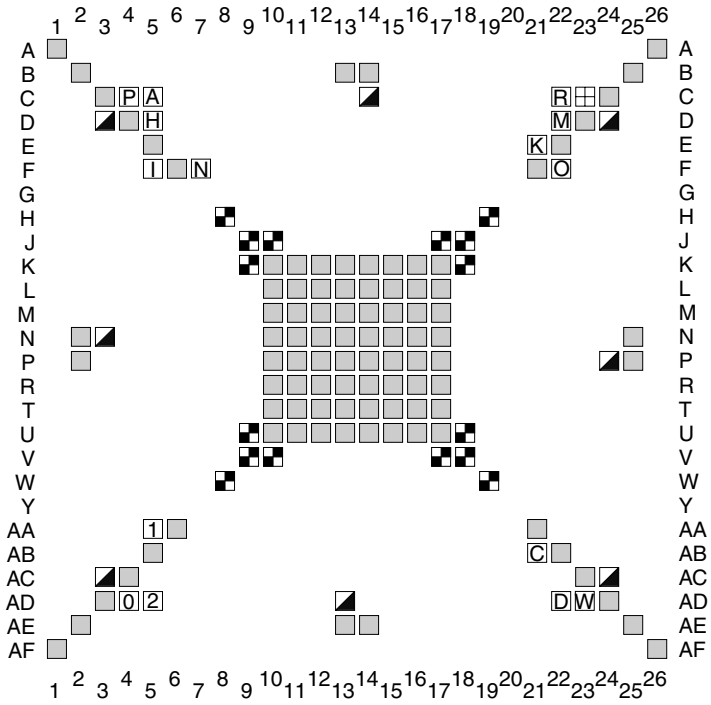
User I/O Pins	Dedicated Pins	
IO_LXXY_#		
Dual-Purpose Pins:		
DIN/D0-D7		
CS_B		
RDWR_B		VCCO
BUSY/DOUT		
INIT_B		
GCLKx (P)		
GCLKx (S)		
VRP		
VRN		
VREF		
Triple-Purpose Pins:		
D2, D4/ALT_VRP		
D3, D5/ALT_VRN		

ug002_c4_49b_031501

Figure 5-9: **FG676 Bank Information**

FG676 Dedicated Pins

FG676 - Top View

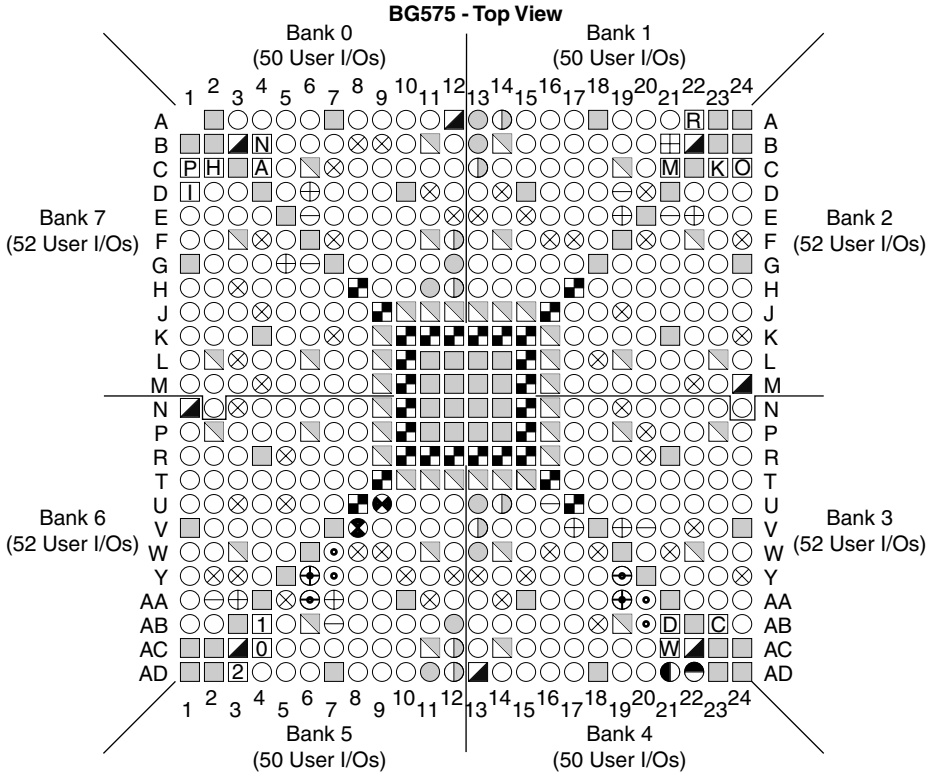


User I/O Pins	Dedicated Pins	
	C CCLK	N DXN
	P PROG_B	A DXP
	D DONE	VBATT
	2 1 0 M2, M1, M0	R RSVD
	H HSWAP_EN	
	K TCK	VCCAUX
	I TDI	VCCINT
	O TDO	GND
	M TMS	n NO CONNECT
	W PWRDWN_B	

ug002_c4_49c_120400

Figure 5-10: **FG676 Dedicated Pins**

BG575 Standard BGA Composite Pinout Diagram (XC2V2000)

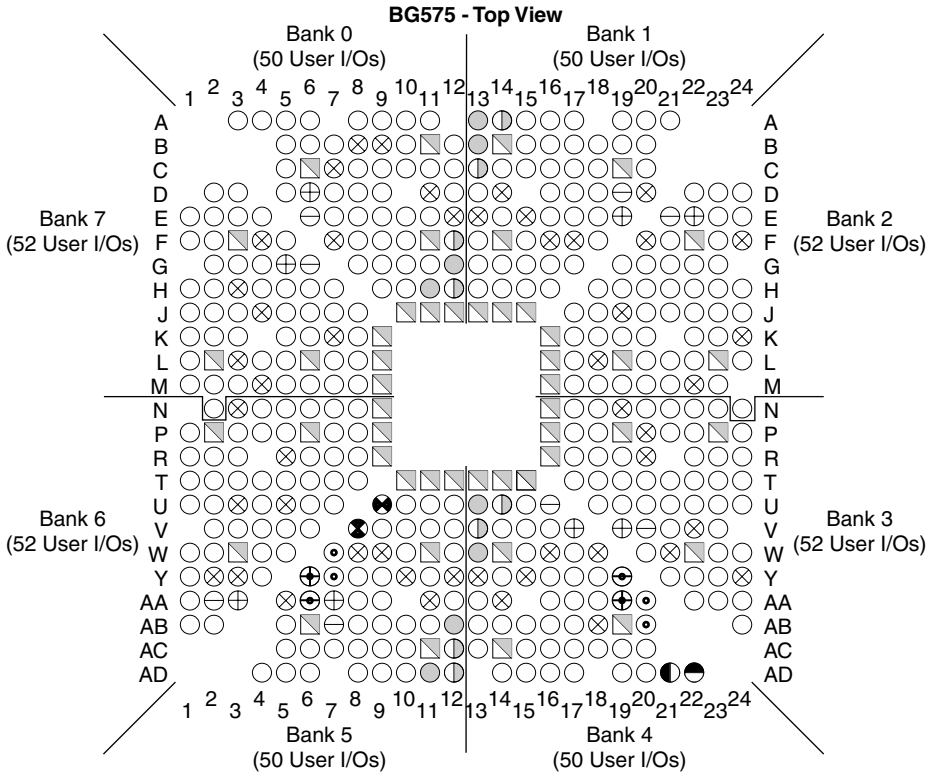


User I/O Pins		Dedicated Pins			
○	IO_LXXY_#	C	CCLK	N	DXN
Dual-Purpose Pins:		P	PROG_B	A	DXP
⊙	DIN/D0-D7	D	DONE	⊞	VBATT
⊗	CS_B	2 1 0	M2, M1, M0	R	RSVD
⧓	RDWR_B	H	HSWAP_EN	◹	VCCO
◓	BUSY/DOUT	K	TCK	◢	VCCAUX
◐	INIT_B	I	TDI	▞	VCCINT
●	GCLKx (P)	O	TDO	■	GND
◑	GCLKx (S)	M	TMS	n	NO CONNECT
⊖	VRP	W	PWRDWN_B		
⊕	VRN				
⊗	VREF				
Triple-Purpose Pins:					
⊙	D2, D4/ALT_VRP				
⊕	D3, D5/ALT_VRN				

ug002_c4_50_031501

Figure 5-11: **BG575 Standard BGA Composite Pinout Diagram (XC2V2000)**

BG575 Bank Information



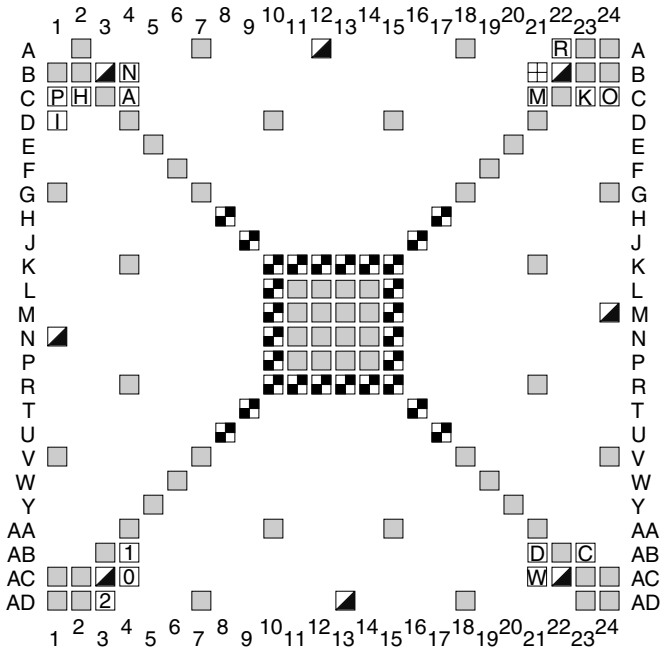
User I/O Pins	Dedicated Pins	
○ IO_LXXY_#		
Dual-Purpose Pins:		
⊙ DIN/D0-D7		
⊗ CS_B		
⊗ RDWR_B		▨ VCCO
◓ BUSY/DOUT		
◐ INIT_B		
● GCLKx (P)		
◑ GCLKx (S)		
⊖ VRP		
⊕ VRN		
⊗ VREF		
Triple-Purpose Pins:		
⊖ D2, D4/ALT_VRP		
⊕ D3, D5/ALT_VRN		

ug002_c4_50b_031501

Figure 5-12: **BG575 Bank Information**

BG575 Dedicated Pins

BG575 - Top View

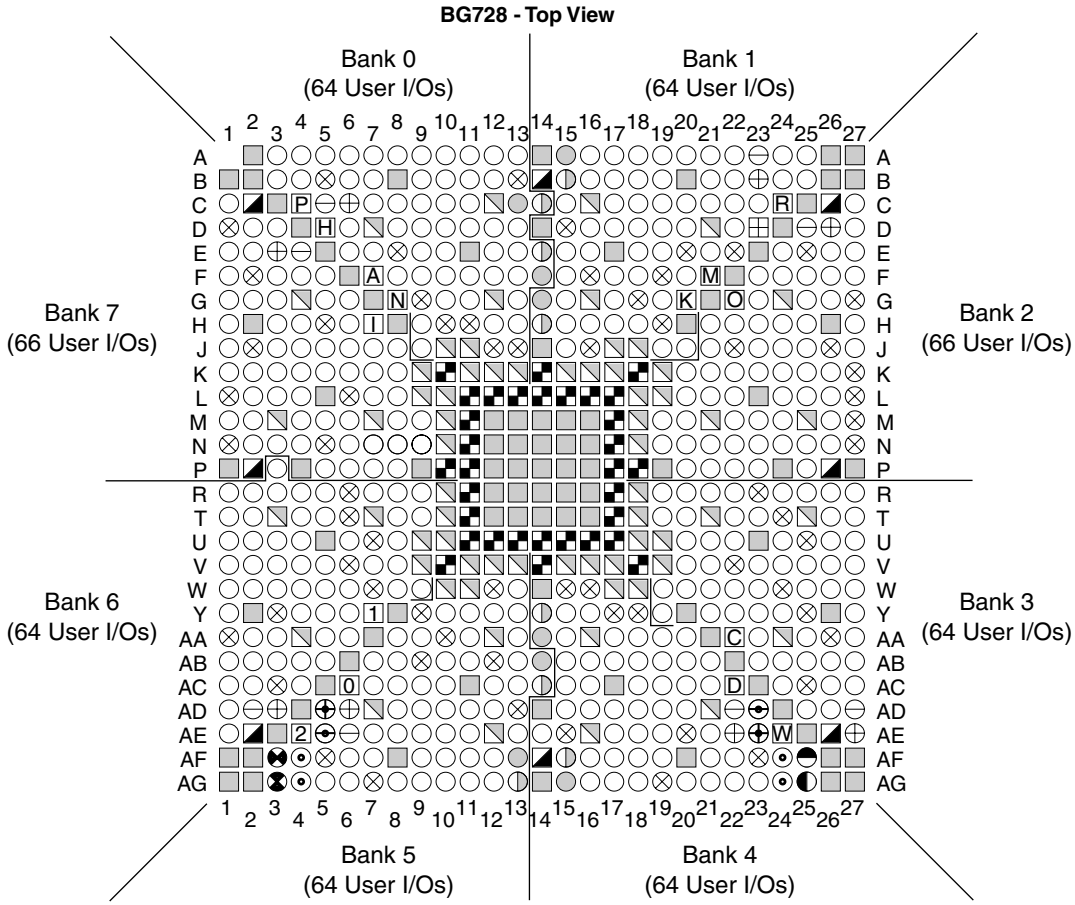


User I/O Pins	Dedicated Pins	
	C CCLK	N DXN
	P PROG_B	A DXP
	D DONE	VBATT
	2 1 0 M2, M1, M0	R RSVD
	H HSWAP_EN	
	K TCK	VCCAUX
	I TDI	VCCINT
	O TDO	GND
	M TMS	n NO CONNECT
	W PWRDWN_B	

ug002_c4_50c_120400

Figure 5-13: **BG575 Dedicated Pins**

BG728 Standard BGA Composite Pinout Diagram (XC2V3000)

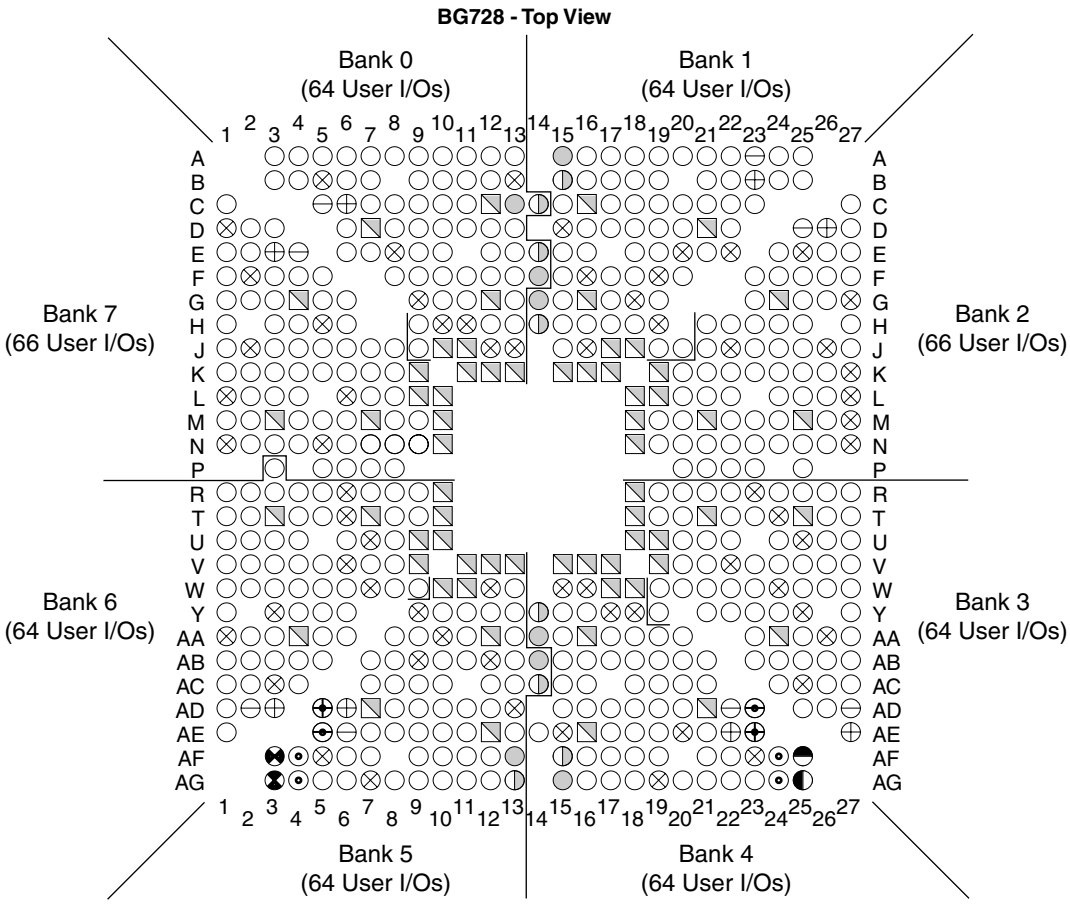


User I/O Pins		Dedicated Pins			
○	IO_LXXY_#	C	CCLK	N	DXN
Dual-Purpose Pins:		P	PROG_B	A	DXP
⊙	DIN/D0-D7	D	DONE	⊞	VBATT
⊠	CS_B	2 1 0	M2, M1, M0	R	RSVD
⊠	RDWR_B	H	HSWAP_EN	▧	VCCO
◓	BUSY/DOUT	K	TCK	◢	VCCAUX
◐	INIT_B	I	TDI	▩	VCCINT
●	GCLKx (P)	O	TDO	■	GND
◑	GCLKx (S)	M	TMS	n	NO CONNECT
⊖	VRP	W	PWRDWN_B		
⊕	VRN				
⊗	VREF				
Triple-Purpose Pins:					
⊖	D2, D4/ALT_VRP				
⊕	D3, D5/ALT_VRN				

ug002_c4_51_031501

Figure 5-14: **BG728 Standard BGA Composite Pinout Diagram (XC2V3000)**

BG728 Bank Information



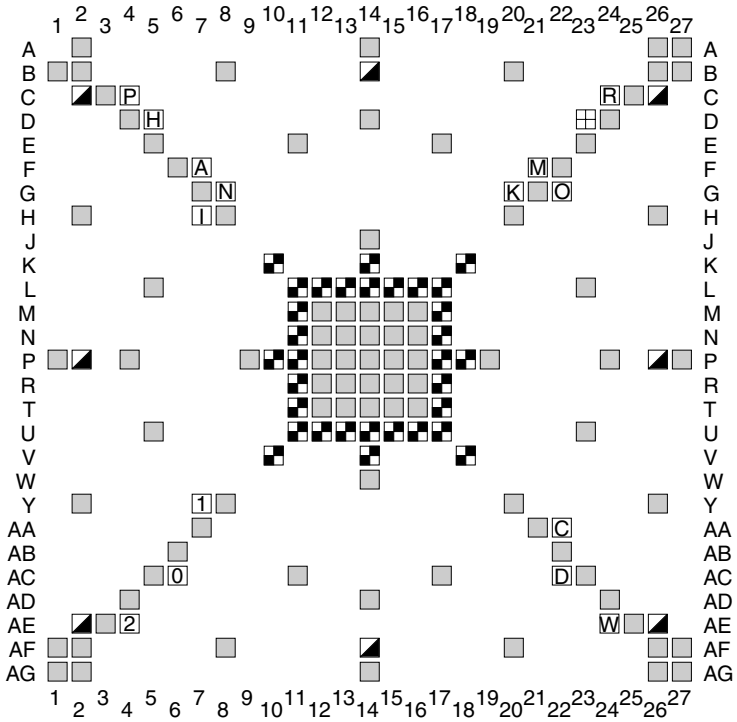
User I/O Pins	Dedicated Pins	
○ IO_LXXY_#		
Dual-Purpose Pins:		
⊙ DIN/D0-D7		
⊗ CS_B		
⧗ RDWR_B		▨ VCCO
◓ BUSY/DOUT		
◐ INIT_B		
● GCLKx (P)		
◑ GCLKx (S)		
⊖ VRP		
⊕ VRN		
⊗ VREF		
Triple-Purpose Pins:		
⊖ D2, D4/ALT_VRP		
⊕ D3, D5/ALT_VRN		

ug002_c4_51b_031501

Figure 5-15: **BG728 Bank Information**

BG728 Dedicated Pins

BG728 - Top View

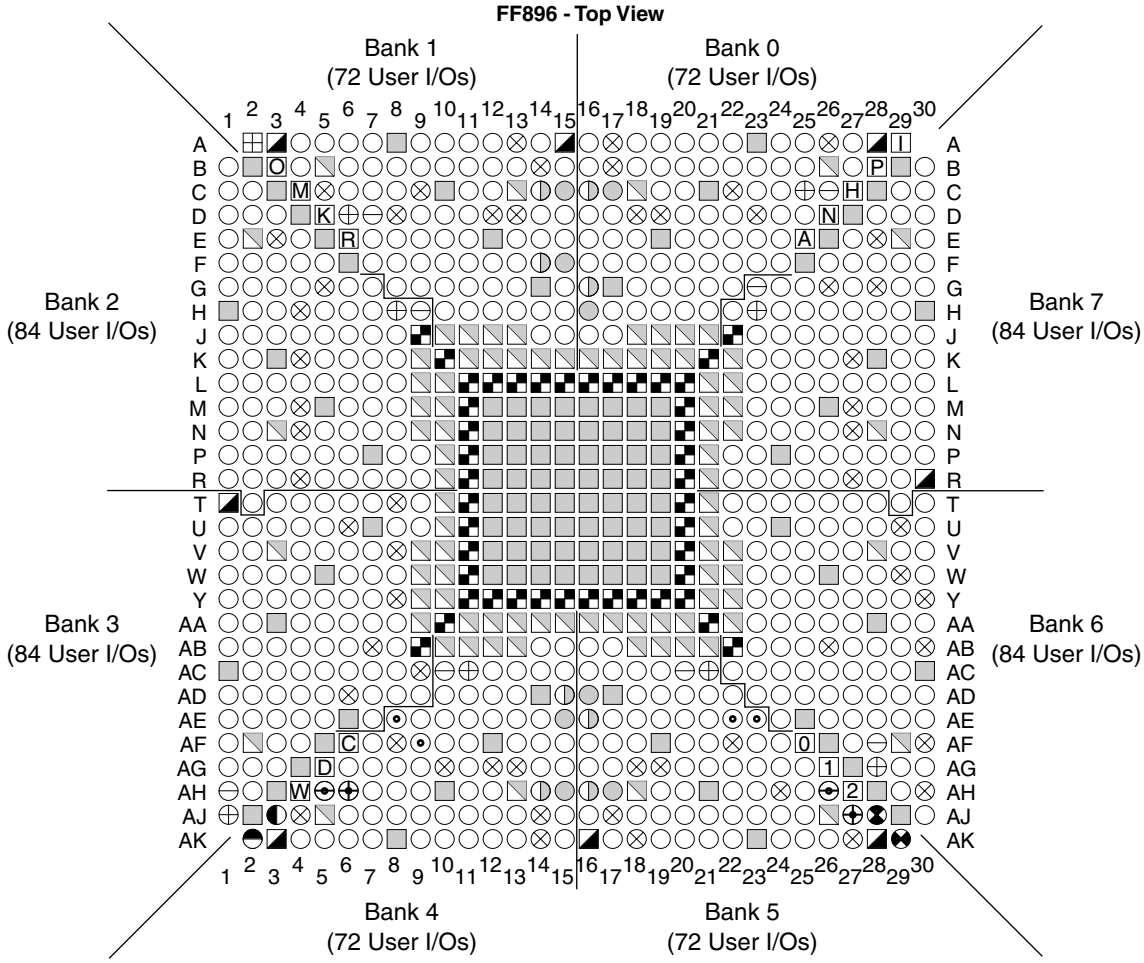


User I/O Pins	Dedicated Pins	
	C CCLK	N DXN
	P PROG_B	A DXP
	D DONE	VBATT
	2 1 0 M2, M1, M0	R RSVD
	H HSWAP_EN	
	K TCK	VCCAUX
	I TDI	VCCINT
	O TDO	GND
	M TMS	n NO CONNECT
	W PWRDWN_B	

ug002_c4_51c_120400

Figure 5-16: **BG728 Dedicated Pins**

FF896 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V2000)

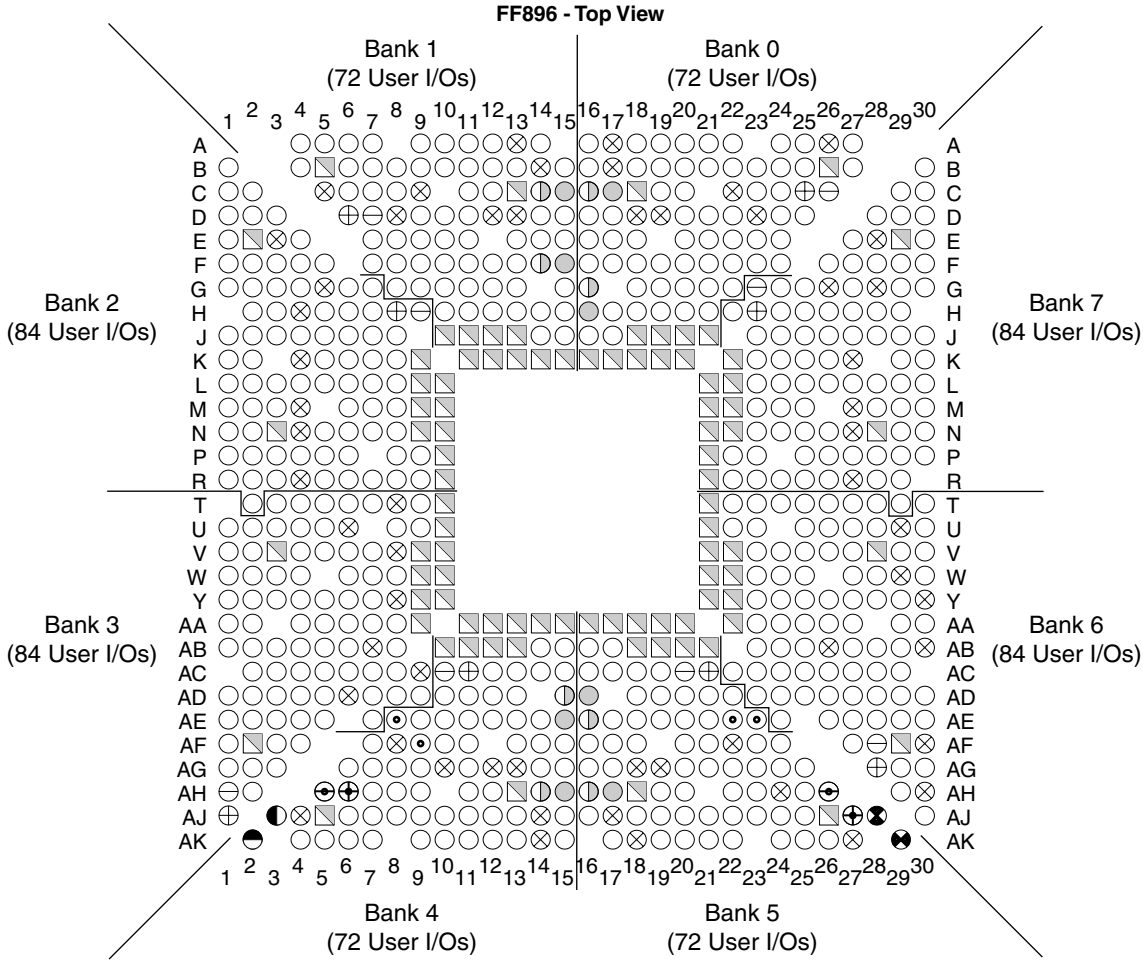


User I/O Pins		Dedicated Pins			
○	IO_LXXY_#	C	CCLK	N	DXN
Dual-Purpose Pins:		P	PROG_B	A	DXP
⊙	DIN/D0-D7	D	DONE	⊞	VBATT
⊠	CS_B	2 1 0	M2, M1, M0	R	RSVD
⊗	RDWR_B	H	HSWAP_EN	▨	VCCO
◓	BUSY/DOUT	K	TCK	◪	VCCAUX
◐	INIT_B	I	TDI	▩	VCCINT
●	GCLKx (P)	O	TDO	■	GND
◑	GCLKx (S)	M	TMS	n	NO CONNECT
⊖	VRP	W	PWRDWN_B		
⊕	VRN				
⊗	VREF				
Triple-Purpose Pins:					
⊖	D2,D4/ALT_VRP				
⊕	D3,D5/ALT_VRN				

ug002_c4_52_031501

Figure 5-17: **FF896 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V2000)**

FF896 Bank Information

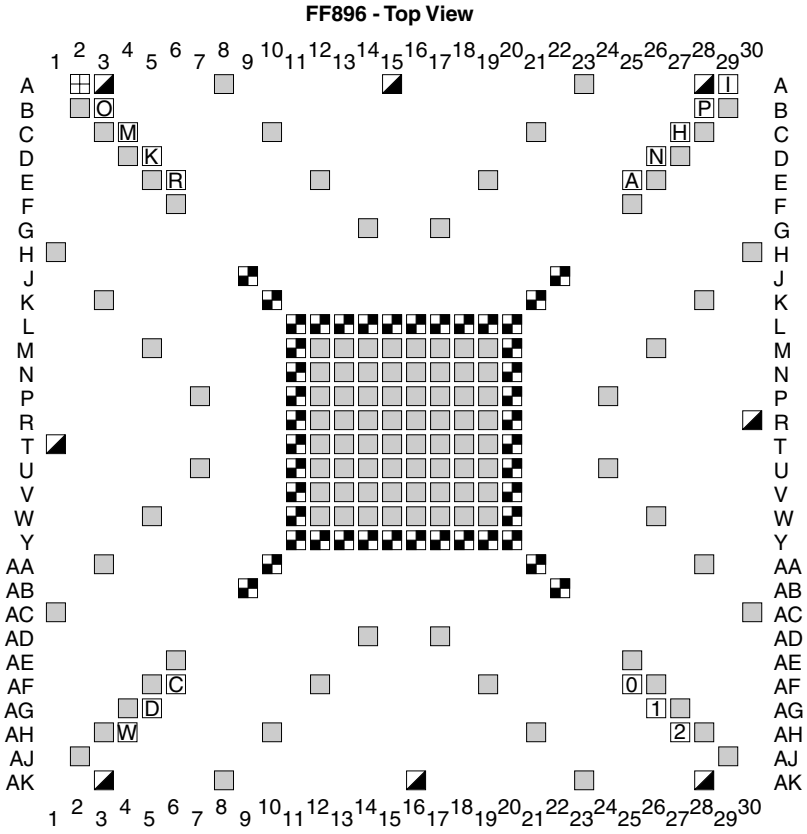


User I/O Pins	Dedicated Pins	
○ IO_LXXY_#		
Dual-Purpose Pins:		
⊙ DIN/D0-D7		
⊗ CS_B		
⊗ RDWR_B		▨ VCCO
◓ BUSY/DOUT		
◐ INIT_B		
● GCLKx (P)		
◑ GCLKx (S)		
⊖ VRP		
⊕ VRN		
⊗ VREF		
Triple-Purpose Pins:		
⊙ D2, D4/ALT_VRP		
⊕ D3, D5/ALT_VRN		

ug002_c4_52b_031501

Figure 5-18: **FF896 Bank Information**

FF896 Dedicated Pins

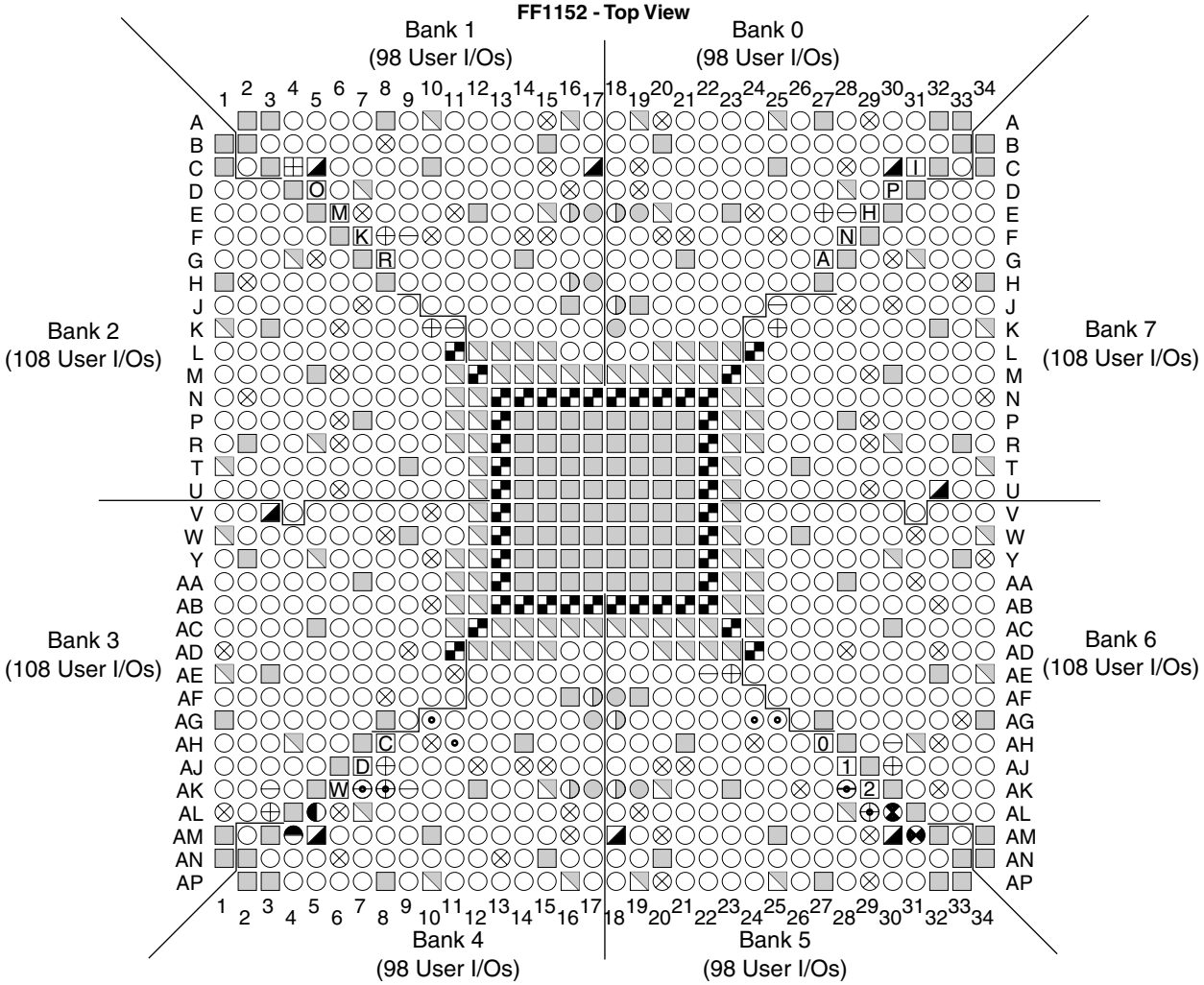


User I/O Pins	Dedicated Pins			
	C	CCLK	N	DXN
	P	PROG_B	A	DXP
	D	DONE		VBATT
	2 1 0	M2, M1, M0	R	RSVD
	H	HSWAP_EN		
	K	TCK		VCCAUX
	I	TDI		VCCINT
	O	TDO		GND
	M	TMS	n	NO CONNECT
	W	PWRDWN_B		

ug002_c4_52c_120400

Figure 5-19: **FF896 Dedicated Pins**

FF1152 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V8000)

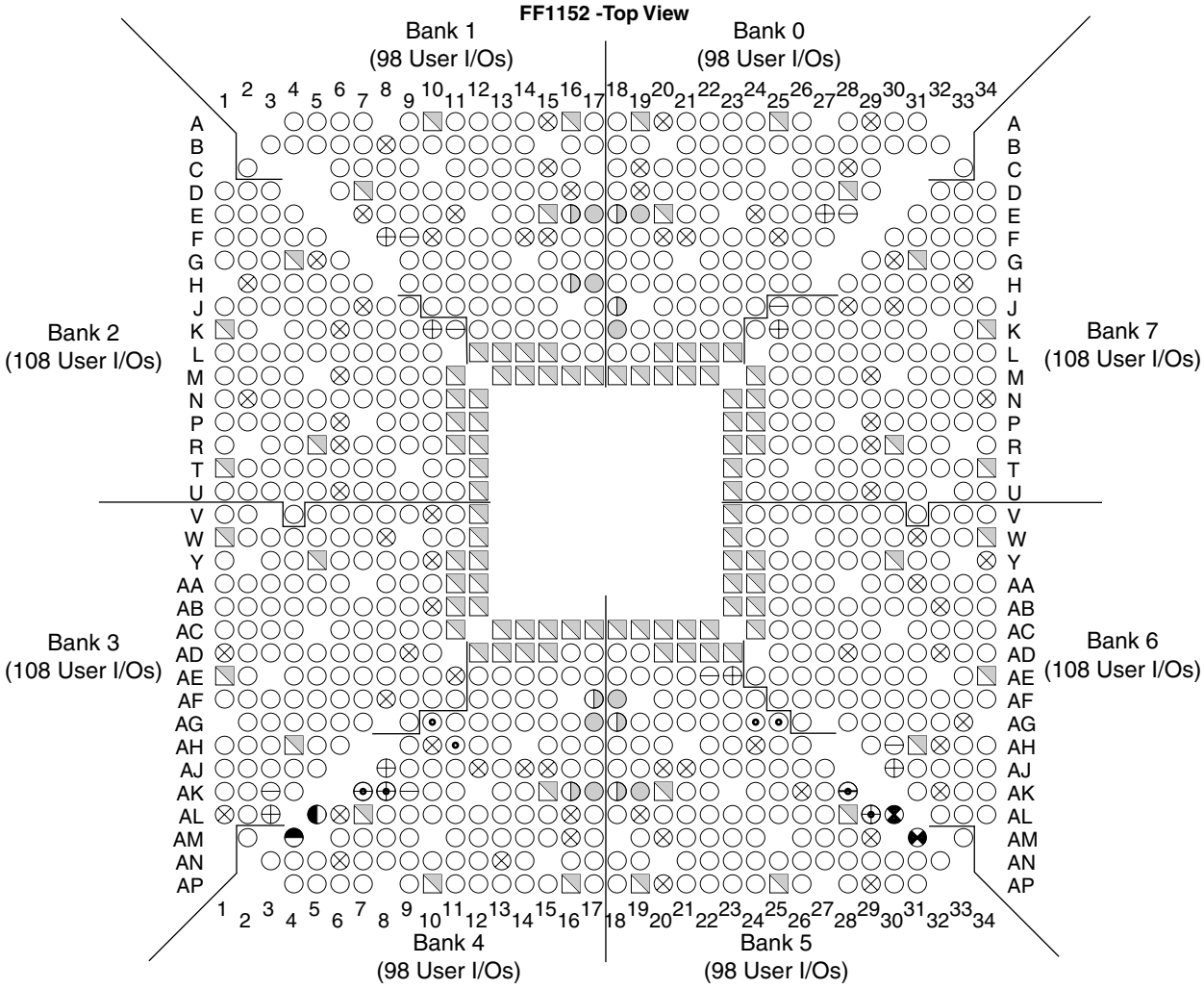


User I/O Pins		Dedicated Pins			
○	IO_LXXY_#	C	CCLK	N	DXN
Dual-Purpose Pins:		P	PROG_B	A	DXP
⊙	DIN/D0-D7	D	DONE	⊞	VBATT
⊗	CS_B	2 1 0	M2, M1, M0	R	RSVD
⊗	RDWR_B	H	HSWAP_EN	◪	VCCO
◓	BUSY/DOUT	K	TCK	◢	VCCAUX
◐	INIT_B	I	TDI	▚	VCCINT
●	GCLKx (P)	O	TDO	■	GND
◑	GCLKx (S)	M	TMS	n	NO CONNECT
⊖	VRP	W	PWRDWN_B		
⊕	VRN				
⊗	VREF				
Triple-Purpose Pins:					
⊙	D2, D4/ALT_VRP				
⊕	D3, D5/ALT_VRN				

ug002_c4_53_031501

Figure 5-20: **FF1152 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V8000)**

FF1152 Bank Information

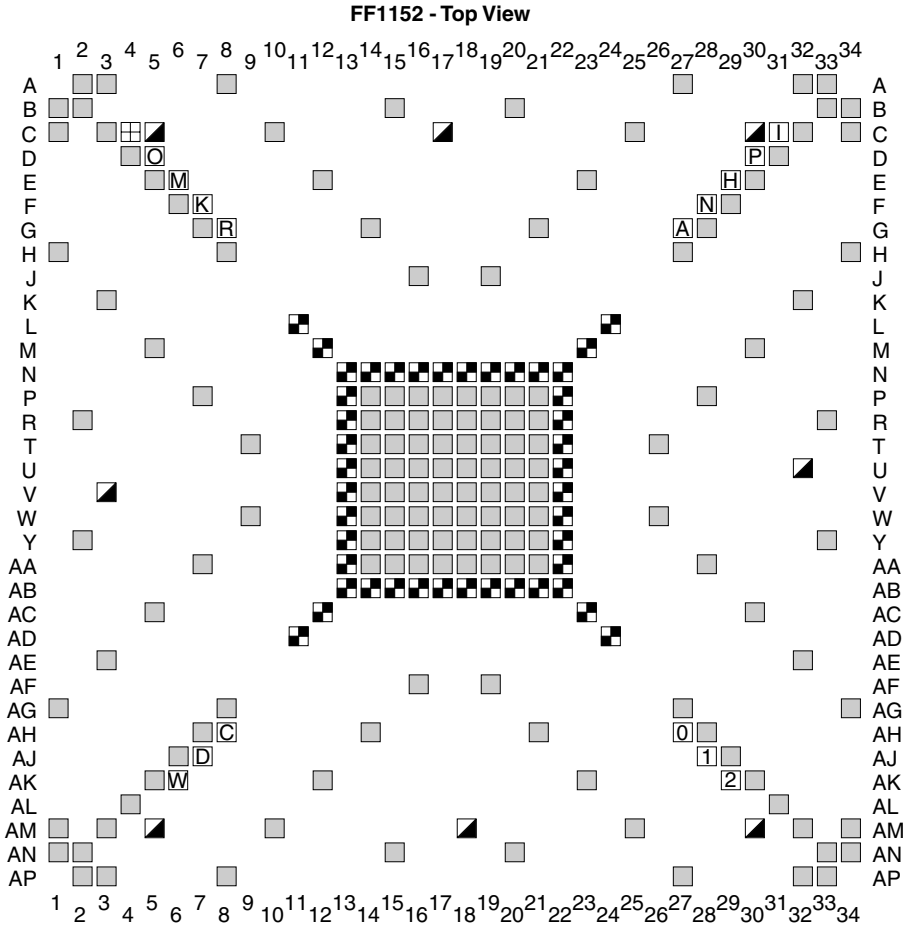


User I/O Pins	Dedicated Pins	
○ IO_LXXY_#		
Dual-Purpose Pins:		
⊙ DIN/D0-D7		
⊗ CS_B		
⊗ RDWR_B		▨ VCCO
◓ BUSY/DOUT		
◐ INIT_B		
● GCLKx (P)		
◑ GCLKx (S)		
⊖ VRP		
⊕ VRN		
⊗ VREF		
Triple-Purpose Pins:		
⊖ D2, D4/ALT_VRP		
⊕ D3, D5/ALT_VRN		

ug002_c4_53b_031501

Figure 5-21: **FF1152 Bank Information**

FF1152 Dedicated Pins

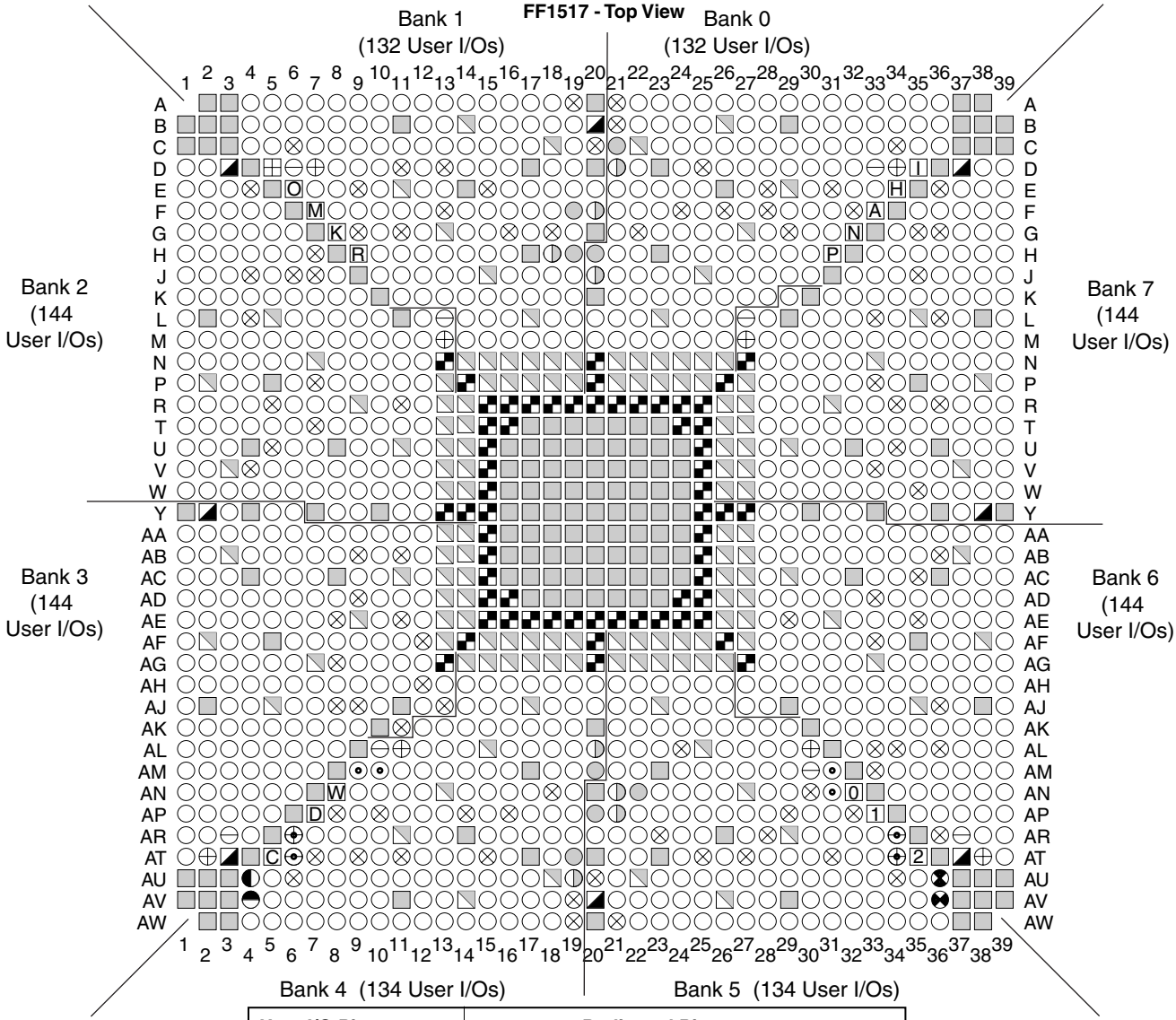


User I/O Pins	Dedicated Pins			
	C	CCLK	N	DXN
	P	PROG_B	A	DXP
	D	DONE		VBATT
	2 1 0	M2, M1, M0	R	RSVD
	H	HSWAP_EN		
	K	TCK		VCCAUX
	I	TDI		VCCINT
	O	TDO		GND
	M	TMS	n	NO CONNECT
	W	PWRDWN_B		

ug002_c4_53c_120400

Figure 5-22: **FF1152 Dedicated Pins**

FF1517 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V8000)

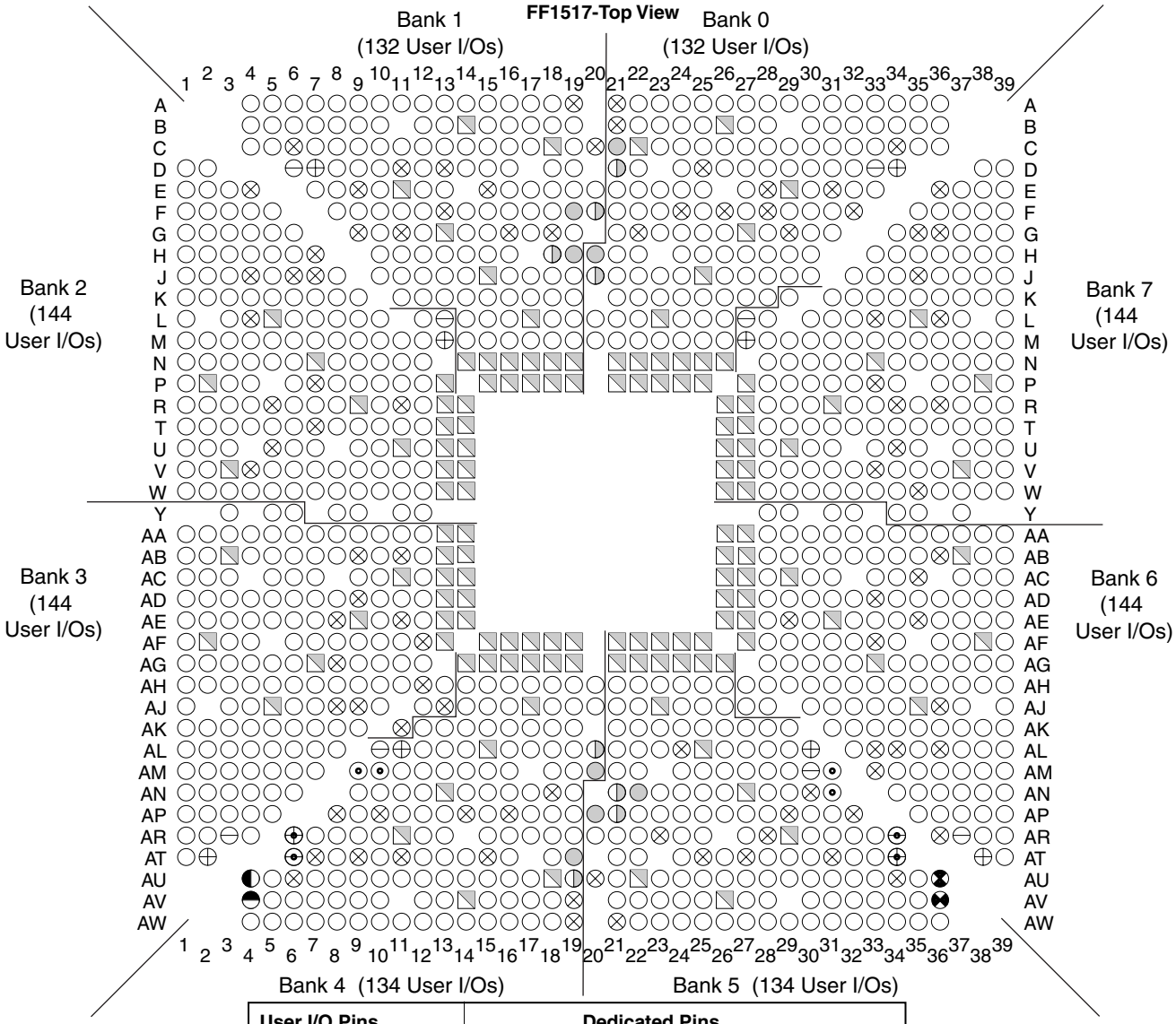


User I/O Pins		Dedicated Pins			
○	IO_LXXY_#	C	CCLK	N	DXN
Dual-Purpose Pins:		P	PROG_B	A	DXP
⊙	DIN/D0-D7	D	DONE	⊞	VBATT
⊗	CS_B	2 1 0	M2, M1, M0	R	RSVD
⊗	RDWR_B	H	HSWAP_EN	▨	VCCO
◓	BUSY/DOUT	K	TCK	◪	VCCAUX
◐	INIT_B	I	TDI	▩	VCCINT
●	GCLKx (P)	O	TDO	■	GND
◑	GCLKx (S)	M	TMS	n	NO CONNECT
⊖	VRP	W	PWRDWN_B		
⊕	VRN				
⊗	VREF				
Triple-Purpose Pins:					
⊛	D2, D4/ALT_VRP				
⊕	D3, D5/ALT_VRN				

ug002_c4_45_031501

Figure 5-23: **FF1517 Flip-Chip Fine-Pitch BGA Composite Pinout Diagram (XC2V8000)**

FF1517 Bank Information

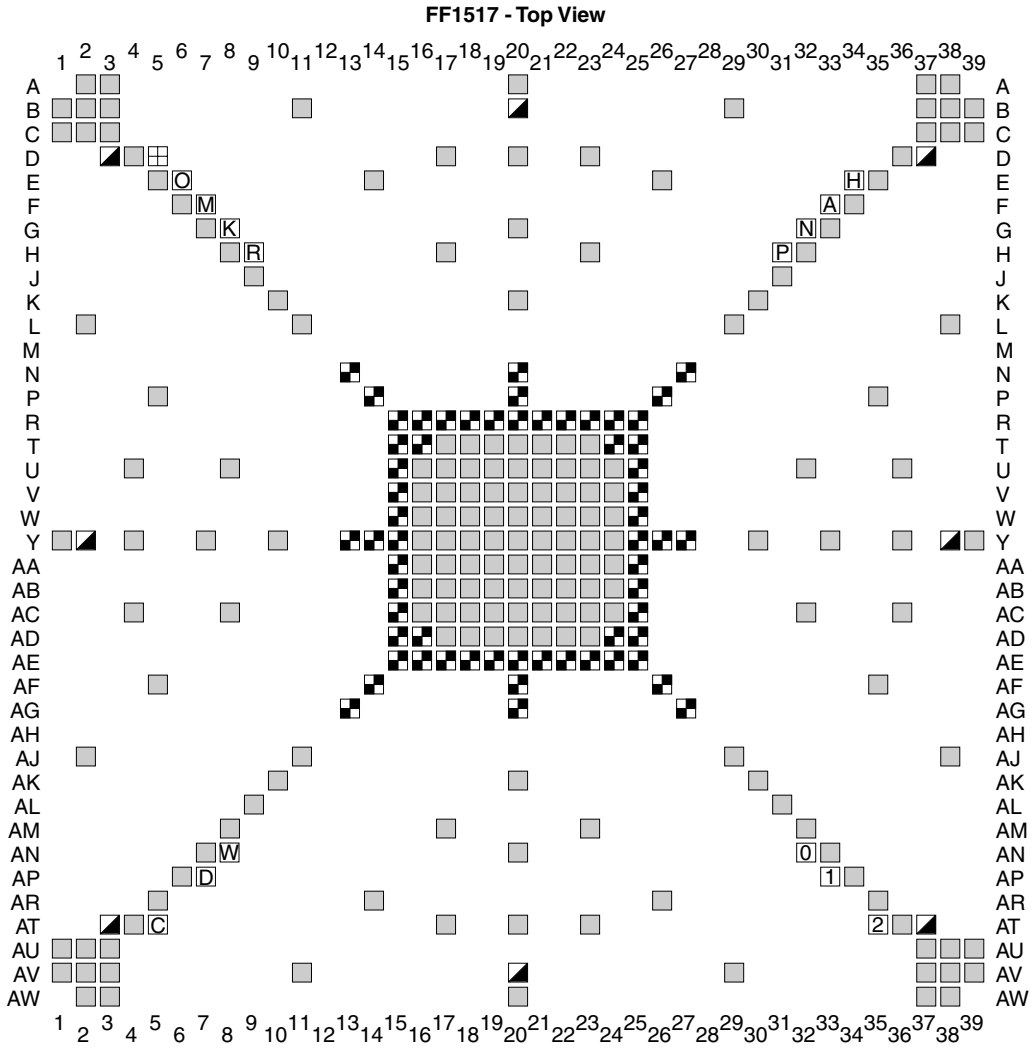


User I/O Pins	Dedicated Pins	
○ IO_LXXY_#		
Dual-Purpose Pins:		
⊙ DIN/D0-D7		
⊗ CS_B		
⊗ RDWR_B		▨ VCCO
◓ BUSY/DOUT		
◐ INIT_B		
● GCLKx (P)		
◑ GCLKx (S)		
⊖ VRP		
⊕ VRN		
⊗ VREF		
Triple-Purpose Pins:		
⊙ D2, D4/ALT_VRP		
⊕ D3, D5/ALT_VRN		

ug002_c4_45b_031501

Figure 5-24: **FF1517 Bank Information**

FF1517 Dedicated Pins

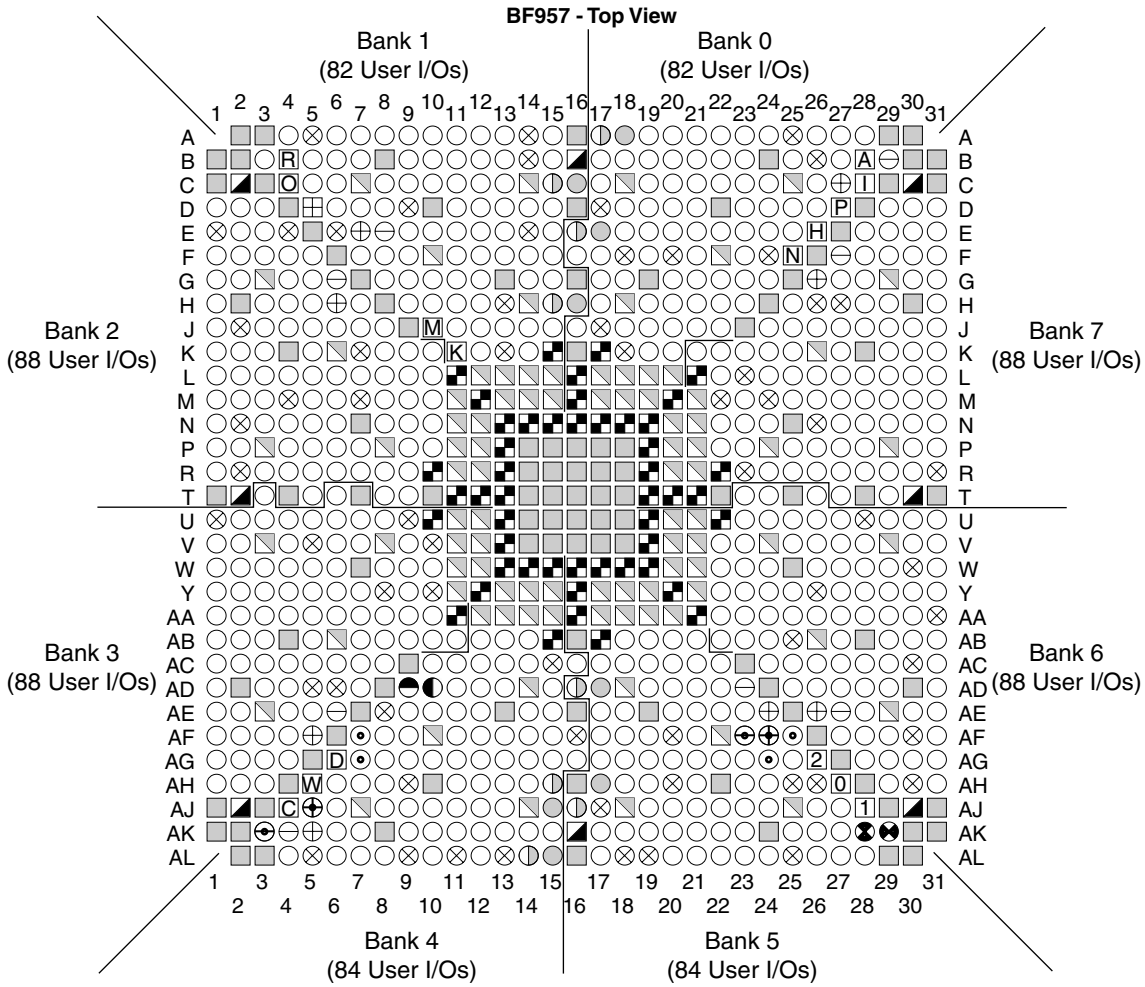


User I/O Pins	Dedicated Pins			
	C	CCLK	N	DXN
	P	PROG_B	A	DXP
	D	DONE		VBATT
	2 1 0	M2, M1, M0	R	RSVD
	H	HSWAP_EN		
	K	TCK		VCCAUX
	I	TDI		VCCINT
	O	TDO		GND
	M	TMS	n	NO CONNECT
	W	PWRDWN_B		

ug002_c4_45c_120400

Figure 5-25: **FF1517 Dedicated Pins**

BF957 Flip-Chip BGA Composite Pinout Diagram (XC2V6000)

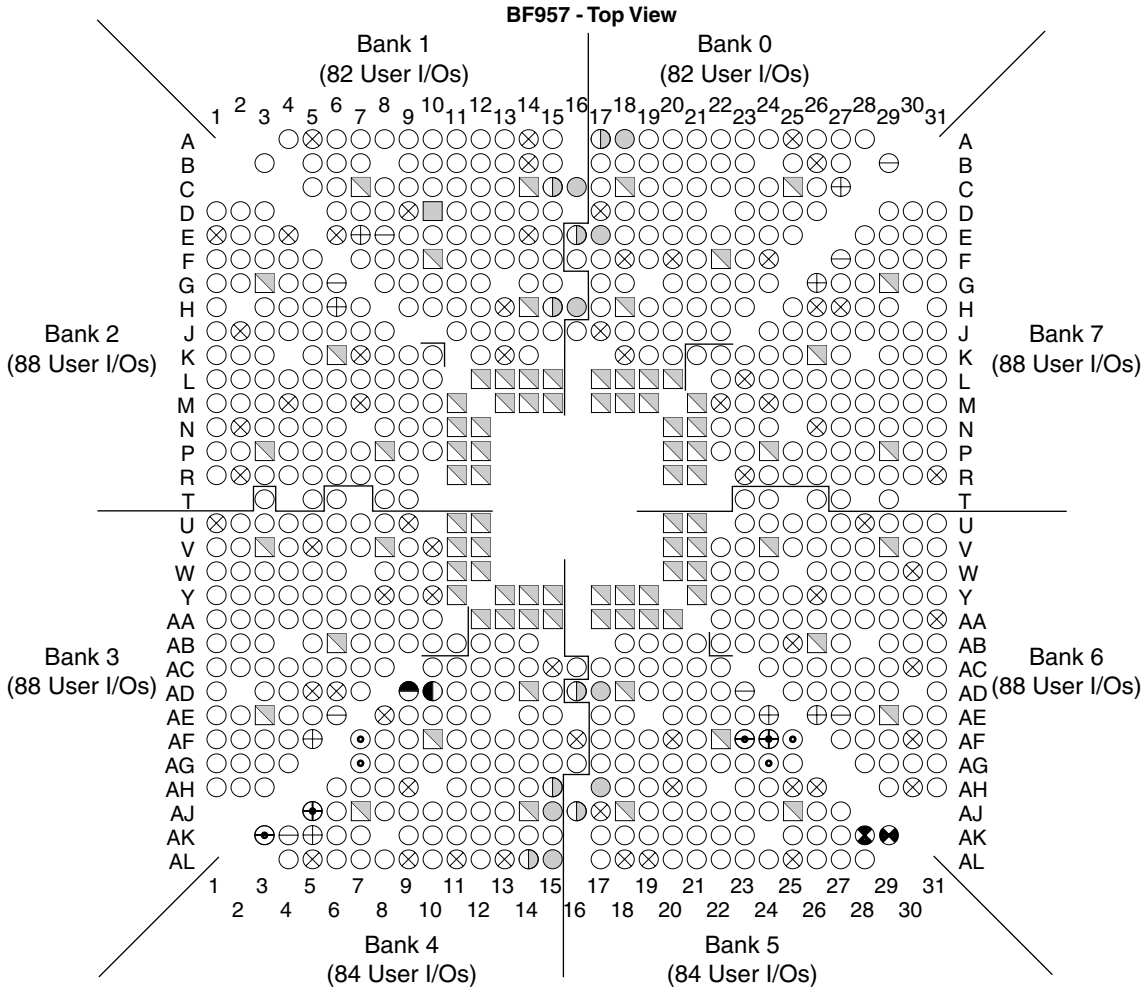


User I/O Pins		Dedicated Pins			
○	IO_LXXY_#	C	CCLK	N	DXN
Dual-Purpose Pins:		P	PROG_B	A	DXP
⊙	DIN/D0-D7	D	DONE	⊞	VBATT
⊗	CS_B	2 1 0	M2, M1, M0	R	RSVD
⊗	RDWR_B	H	HSWAP_EN	▨	VCCO
◓	BUSY/DOUT	K	TCK	◪	VCCAUX
◐	INIT_B	I	TDI	▩	VCCINT
●	GCLKx (P)	O	TDO	▨	GND
◑	GCLKx (S)	M	TMS	n	NO CONNECT
⊖	VRP	W	PWRDWN_B		
⊕	VRN				
⊗	VREF				
Triple-Purpose Pins:					
⊖	D2, D4/ALT_VRP				
⊕	D3, D5/ALT_VRN				

ug002_c4_54_032901

Figure 5-26: **BF957 Flip-Chip BGA Composite Pinout Diagram (XC2V6000)**

BF957 Bank Information

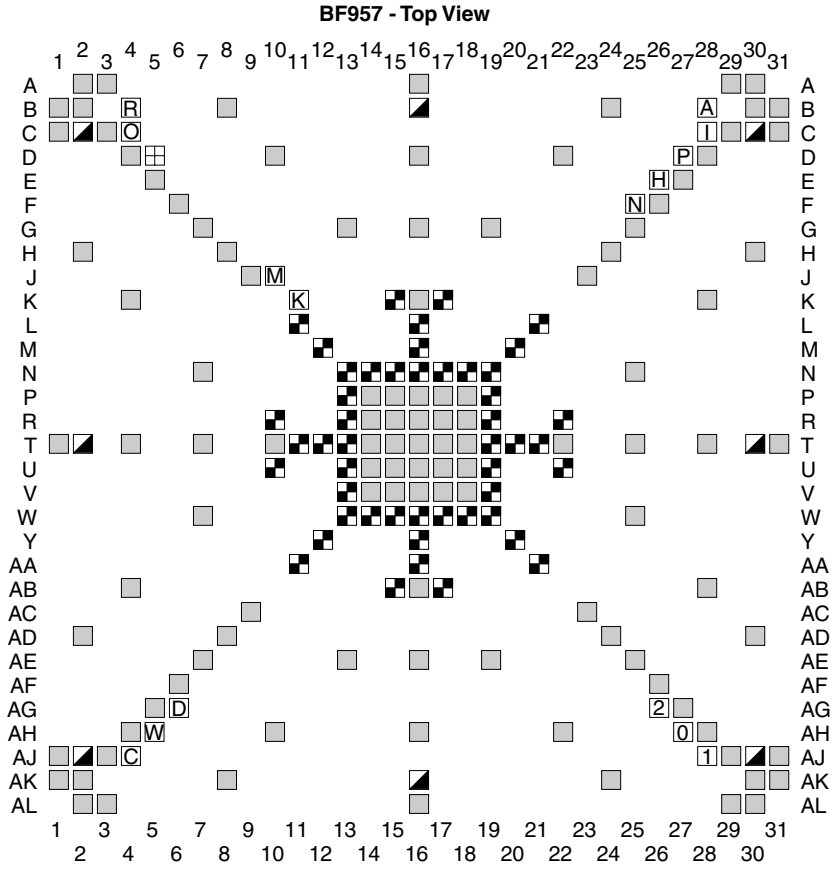


User I/O Pins	Dedicated Pins	
○ IO_LXXY_#		
Dual-Purpose Pins:		
⊙ DIN/D0-D7		
⊗ CS_B		
⧗ RDWR_B		▨ VCCO
◓ BUSY/DOUT		
◐ INIT_B		
● GCLKx (P)		
◑ GCLKx (S)		
⊖ VRP		
⊕ VRN		
⊗ VREF		
Triple-Purpose Pins:		
⊖ D2, D4/ALT_VRP		
⊕ D3, D5/ALT_VRN		

ug002_c4_54b_032901

Figure 5-27: **BF957 Bank Information**

BF957 Dedicated Pins



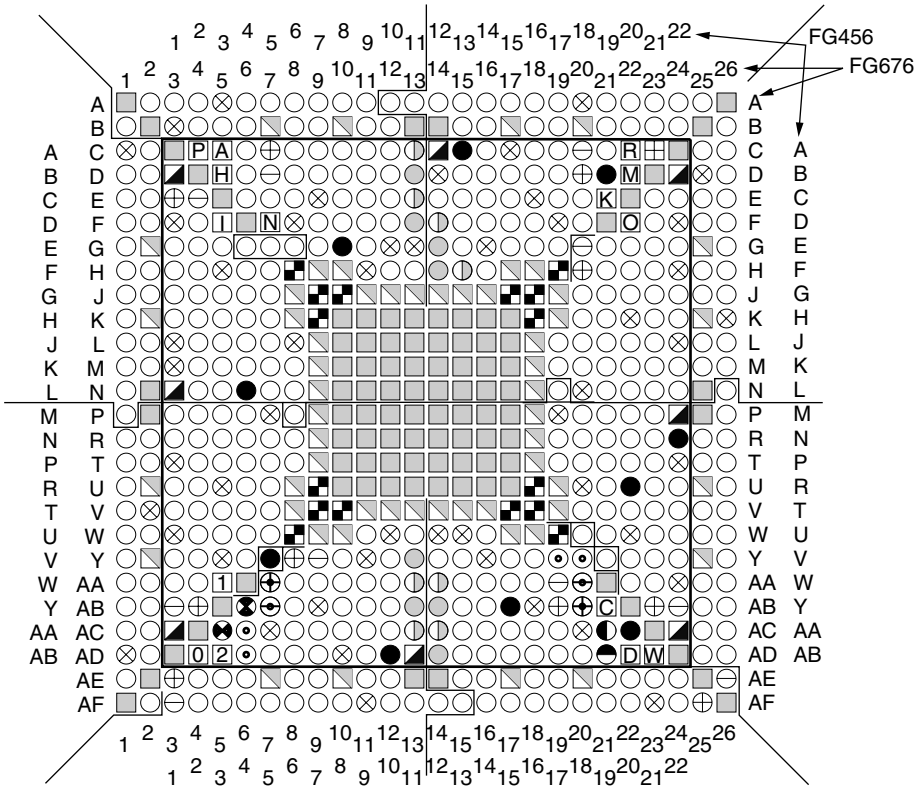
User I/O Pins	Dedicated Pins			
	C	CCLK	N	DXN
	P	PROG_B	A	DXP
	D	DONE	⊞	VBATT
	2 1 0	M2, M1, M0	R	RSVD
	H	HSWAP_EN		
	K	TCK	◪	VCCAUX
	I	TDI	◪	VCCINT
	O	TDO		GND
	M	TMS	n	NO CONNECT
	W	PWRDWN_B		

ug002_c4_54c_120400

Figure 5-28: **BF957 Dedicated Pins**

FG456 - FG676 Pinout Compatibility Diagram

FG676/FG456 Pinout Compatible Diagram - Top View



Note: FF456 and FG676 are pinout compatible with with the exception of the LVDS pairs. I/O V_{REF} pins in FG676 are user I/O pins in FG456. In addition, some user I/O pins are not in the same bank (see ◯◯ lines). VRP (V7) and VRN (V6) in Bank 5 and VRP (W17) and VRN (Y17) in Bank 4 are only user I/Os in FG676.

User I/O Pins	Dedicated Pins	
◯ IO_LXXY_#	C CCLK	N DXN
Dual-Purpose Pins:	P PROG_B	A DXP
⊙ DIN/D0-D7	D DONE	⊞ VBATT
⊗ CS_B	2 1 0 M2, M1, M0	R RSVD
⊗ RDWR_B	H HSWAP_EN	▨ VCCO
◓ BUSY/DOUT	K TCK	◪ VCCAUX
◐ INIT_B	I TDI	▩ VCCINT
● GCLKx (P)	O TDO	▩ GND
◑ GCLKx (S)	M TMS	n NO CONNECT
⊖ VRP	W PWRDWN_B	
⊕ VRN		
⊗ VREF		
● VREF on FG676 User I/O on FG456		
Triple-Purpose Pins:		
⊙ D2, D4/ALT_VRP		
⊕ D3, D5/ALT_VRN		

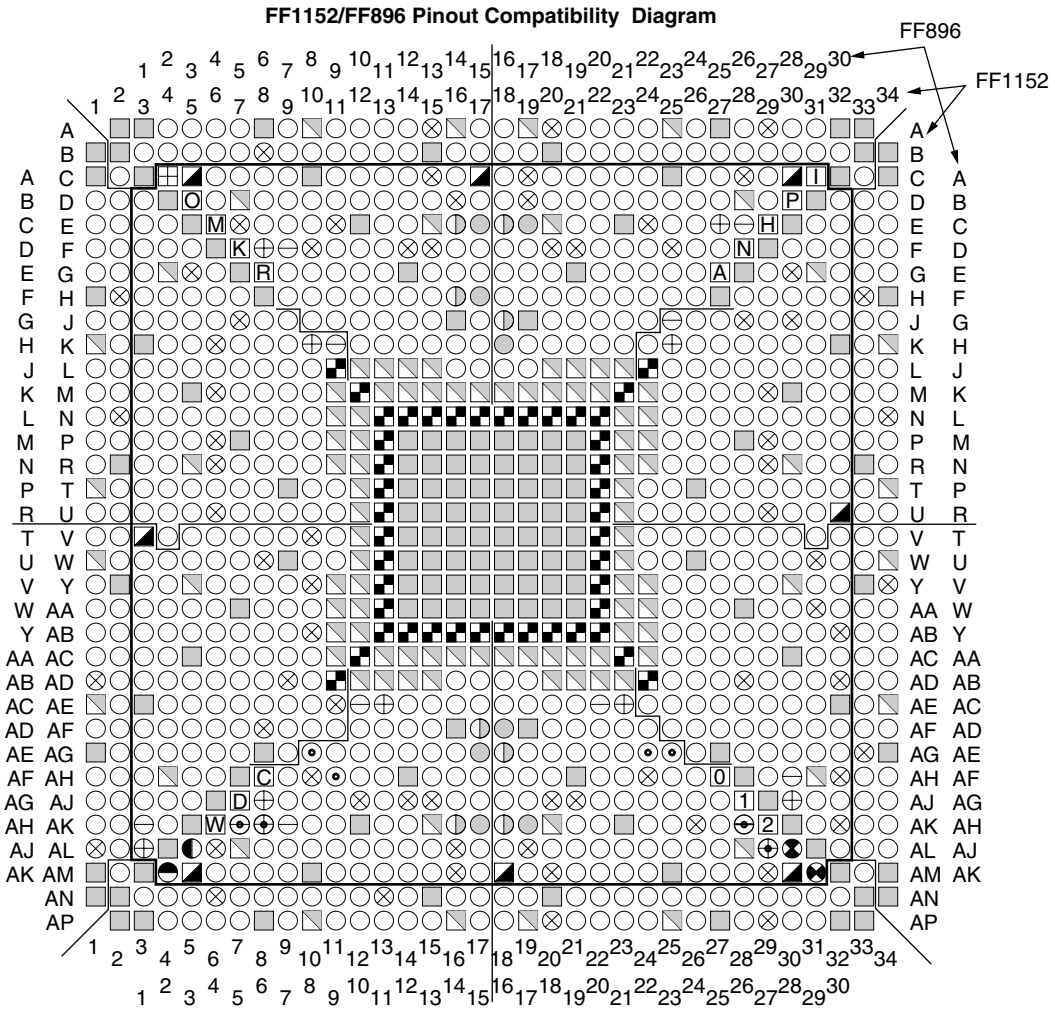
Corresponding Pinouts

FG456	FG676
A1	C3
.	.
.	.
.	.
.	.
.	.
AB22	AD24

ug002_c4_56_080601

Figure 5-29: **FG456 - FG676 Pinout Compatibility Diagram**

FF896 - FF1152 Pinout Compatibility Diagram



Note: FF896 is pinout compatible with the FF1152 except for LVDS pairs. Also, in Bank 4, VRP/VRN pins are not compatible: for FF896, VRP is in AC10 and VRN is in AC11, and for FF1152, VRP is in AK9 and VRN is in AJ8. If DCI is not used in Bank 4, or is used with ALT_VRP or ALT_VRN, then the user I/Os are compatible.

User I/O Pins	Dedicated Pins	
○ IO_LXXY_#	C CCLK	N DXN
Dual-Purpose Pins:	P PROG_B	A DXP
⊙ DIN/D0-D7	D DONE	⊞ VBATT
⊗ CS_B	2 1 0 M2, M1, M0	R RSVD
⊗ RDWR_B	H HSWAP_EN	▨ VCCO
◓ BUSY/DOUT	K TCK	◪ VCCAUX
◐ INIT_B	I TDI	▪ VCCINT
● GCLKx (P)	O TDO	▩ GND
◑ GCLKx (S)	M TMS	n NO CONNECT
⊖ VRP	W PWRDWN_B	
⊕ VRN		
⊗ VREF		
Triple-Purpose Pins:		
⊙ D2, D4/ALT_VRP		
⊕ D3, D5/ALT_VRN		

Corresponding Pinouts

FF896	FF1152
A2	C4
.	.
.	.
.	.
.	.
.	.
AK29	AM31

ug002_c4_55_032901

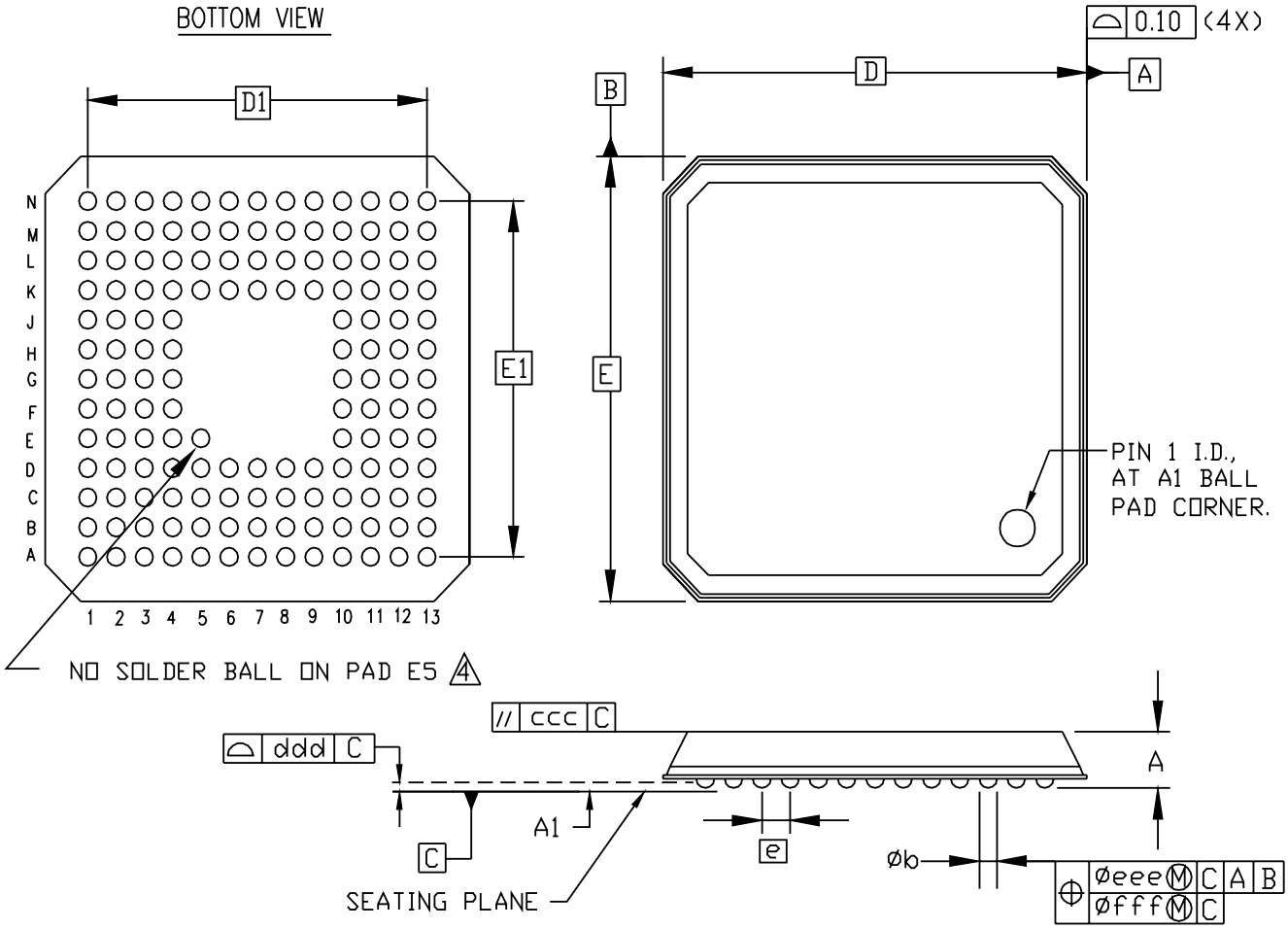
Figure 5-30: **FF896 - FF1152 Pinout Compatibility Diagram**

Package Specifications

This section contains specifications for the following Virtex-II packages:

- "CS144 Chip-Scale BGA Package (0.80 mm Pitch)" on page 410
- "FG256 Fine-Pitch BGA Package (1.00 mm Pitch)" on page 411
- "FG456 Fine-Pitch BGA Package (1.00 mm Pitch)" on page 412
- "FG676 Fine-Pitch BGA Package (1.00 mm Pitch)" on page 413
- "BG575 Standard BGA Package (1.27 mm Pitch)" on page 414
- "BG728 Standard BGA Package (1.27 mm Pitch)" on page 415
- "FF896 Flip-Chip Fine-Pitch BGA Package (1.00 mm Pitch)" on page 416
- "FF1152 Flip-Chip Fine-Pitch BGA Package (1.00 mm Pitch)" on page 417
- "FF1517 Flip-Chip Fine-Pitch BGA Package (1.00 mm Pitch)" on page 418
- "BF957 Flip-Chip BGA Package (1.27 mm Pitch)" on page 419

CS144 Chip-Scale BGA Package (0.80 mm Pitch)



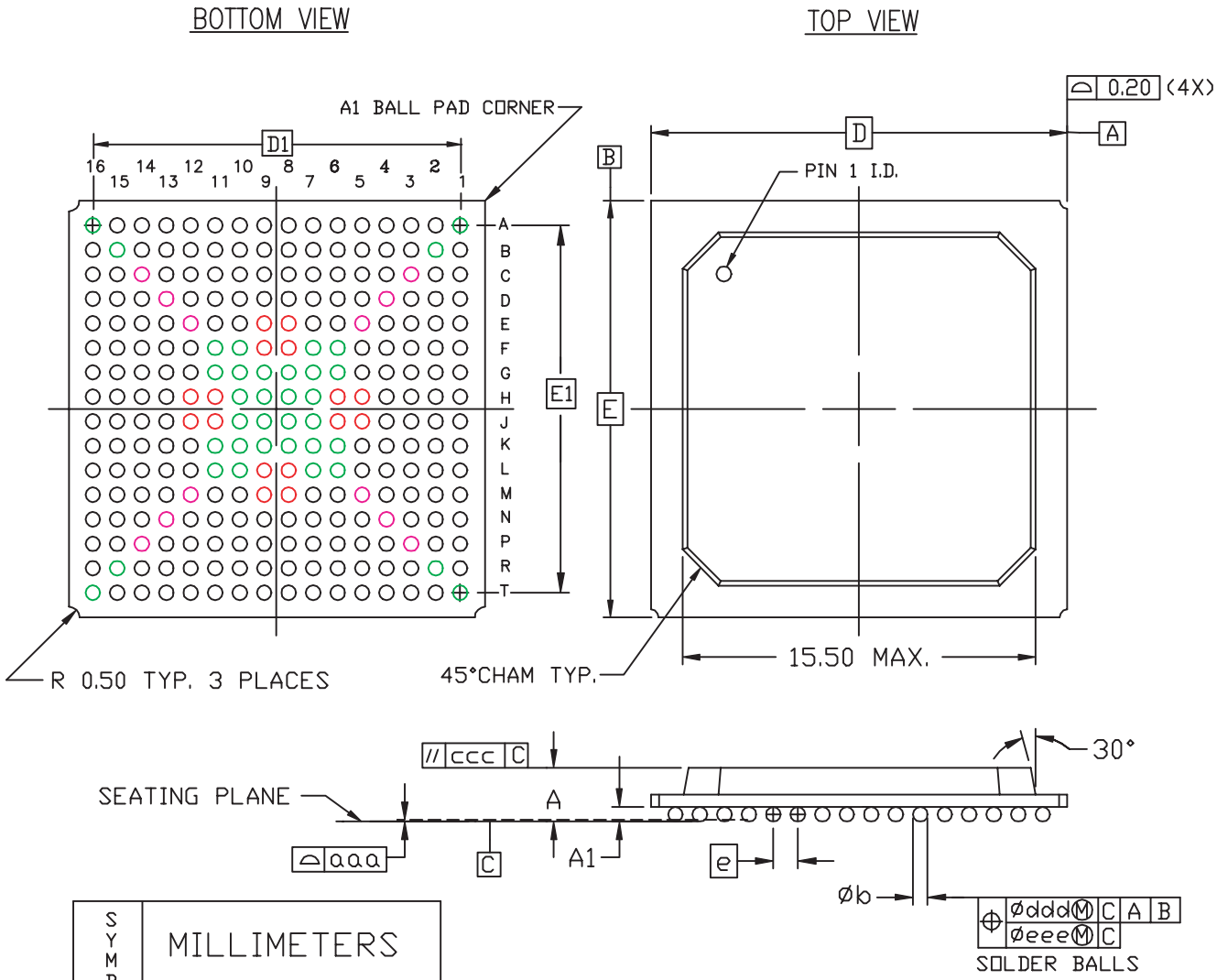
SYMBOL	MILLIMETERS		
	MIN.	NOM.	MAX.
A			1.20
A_1	0.35	0.40	0.45
D/E	12.00 BSC		
D_1/E_1	9.60 BSC		
e	0.80 BSC		
øb	0.45	0.50	0.55
ccc			0.10
ddd			0.12
eee			0.15
fff			0.08
M	13		

NOTES:

1. ALL DIMENSIONING AND TOLERANCING CONFORM TO ASME Y14.5M–1994
2. SYMBOL "M" IS THE PIN MATRIX SIZE.
3. CONFORMS TO JEDEC MO–205–BE (DEPOPULATED).
4. PAD 'E5' IS FOR PAD 'A1' CORNER INDICATION.

Figure 5-31: **CS144 Chip-Scale BGA Package**

FG256 Fine-Pitch BGA Package (1.00 mm Pitch)



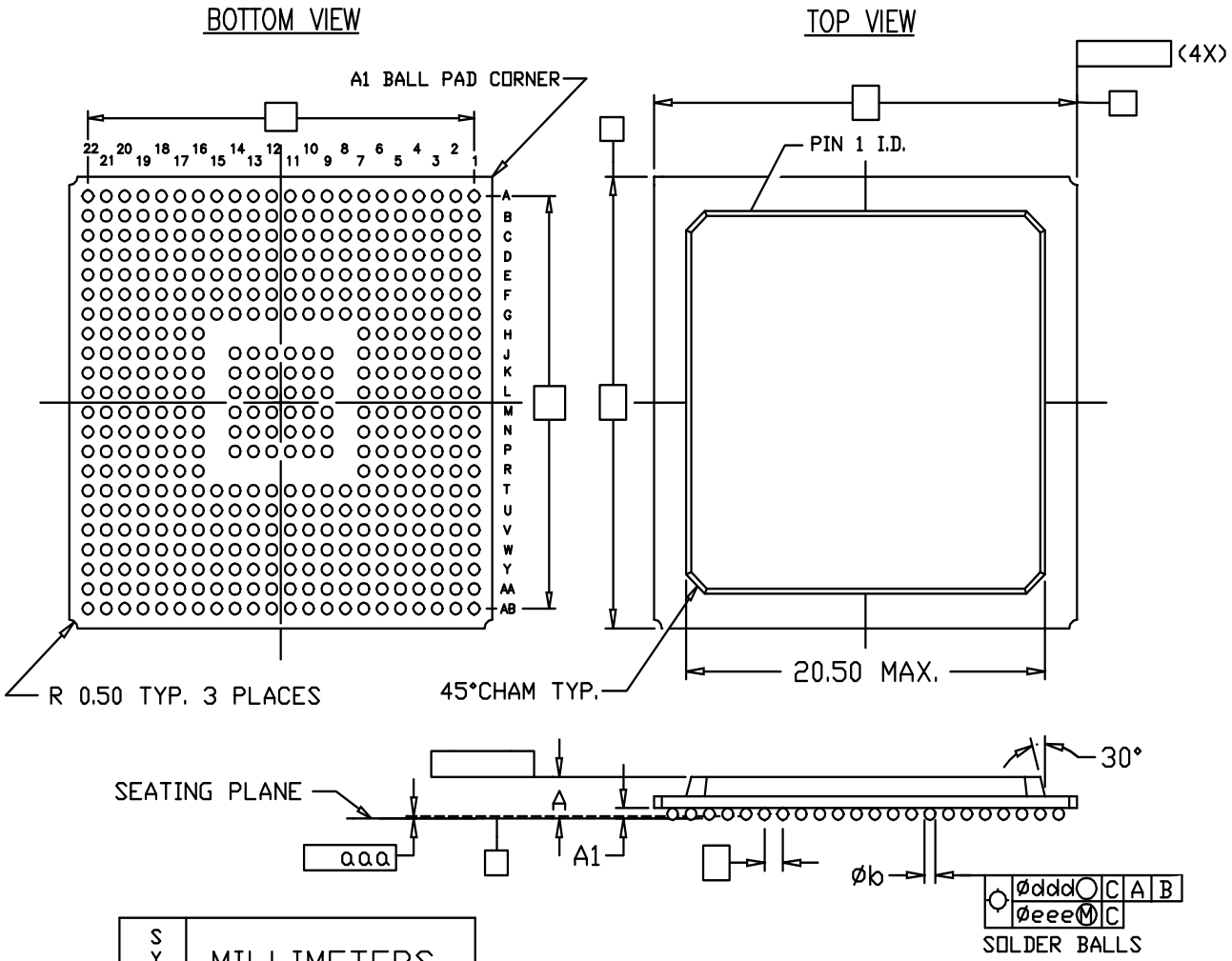
SYMBOL	MILLIMETERS		
	MIN.	NOM.	MAX.
A		1.73	2.00
A_1	0.40	0.50	0.60
D/E	17.00 BSC		
D_1/E_1	15.00 REF		
e	1.00 BSC		
Øb	0.50	0.60	0.70
aaa			0.20
ccc			0.35
ddd			0.30
eee			0.10
M	16		

NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M–1994
2. SYMBOL 'M' IS THE BALL MATRIX SIZE.
3. CONFORMS TO JEDEC MO–151 AAF–1.

Figure 5-32: **FG256 Fine-Pitch BGA Package**

FG456 Fine-Pitch BGA Package (1.00 mm Pitch)



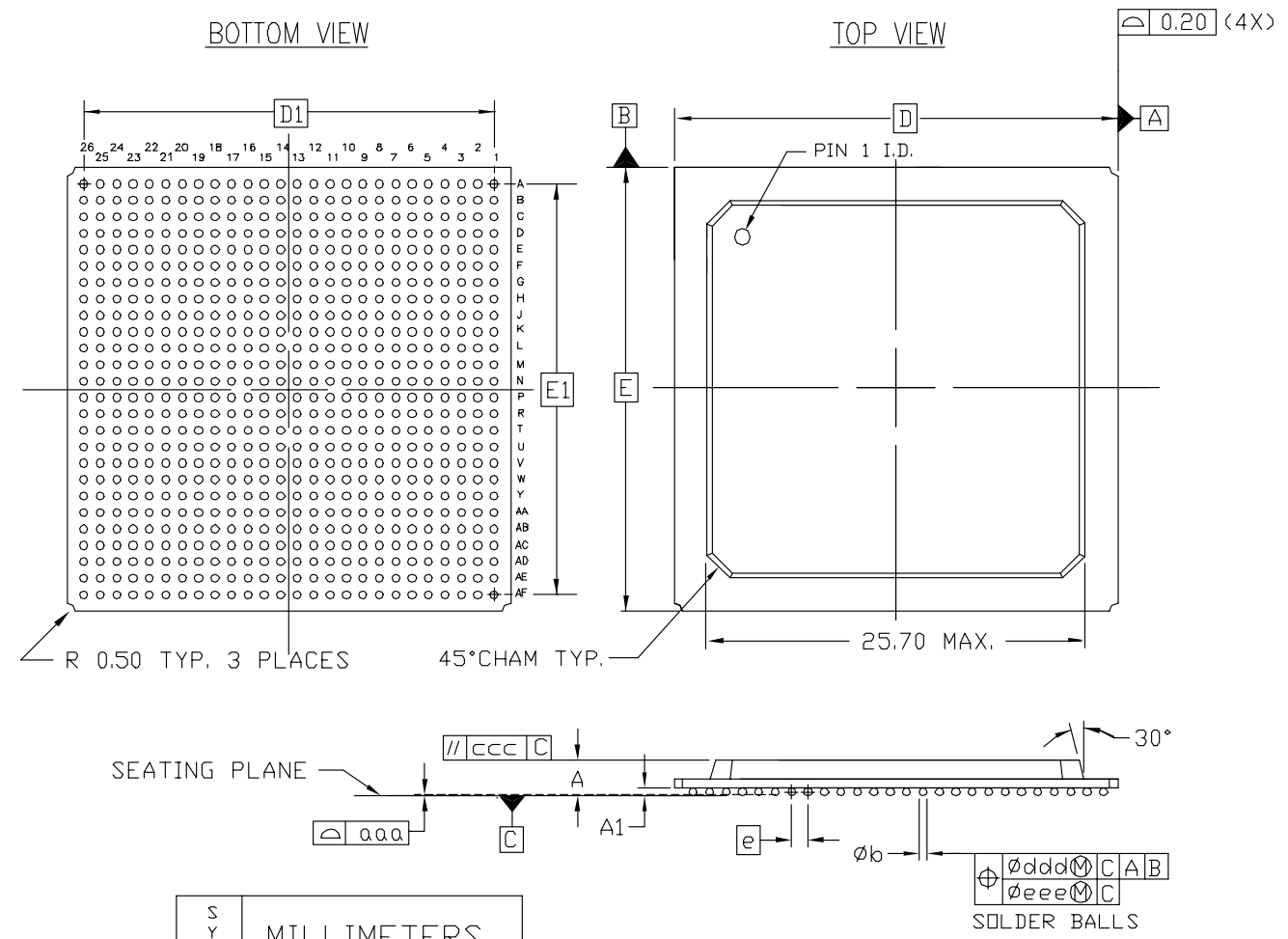
SYMBOL	MILLIMETERS		
	MIN.	NOM.	MAX.
A		2.20	2.60
A_1	0.40	0.50	0.60
D/E	23.00 BSC		
D_1/E_1	21.00 REF		
e	1.00 BSC		
øb	0.50	0.60	0.70
aaa			0.20
ccc			0.35
ddd			0.30
eee			0.10
M	22		

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)

Figure 5-33: **FG456 Fine-Pitch BGA Package**

FG676 Fine-Pitch BGA Package (1.00 mm Pitch)



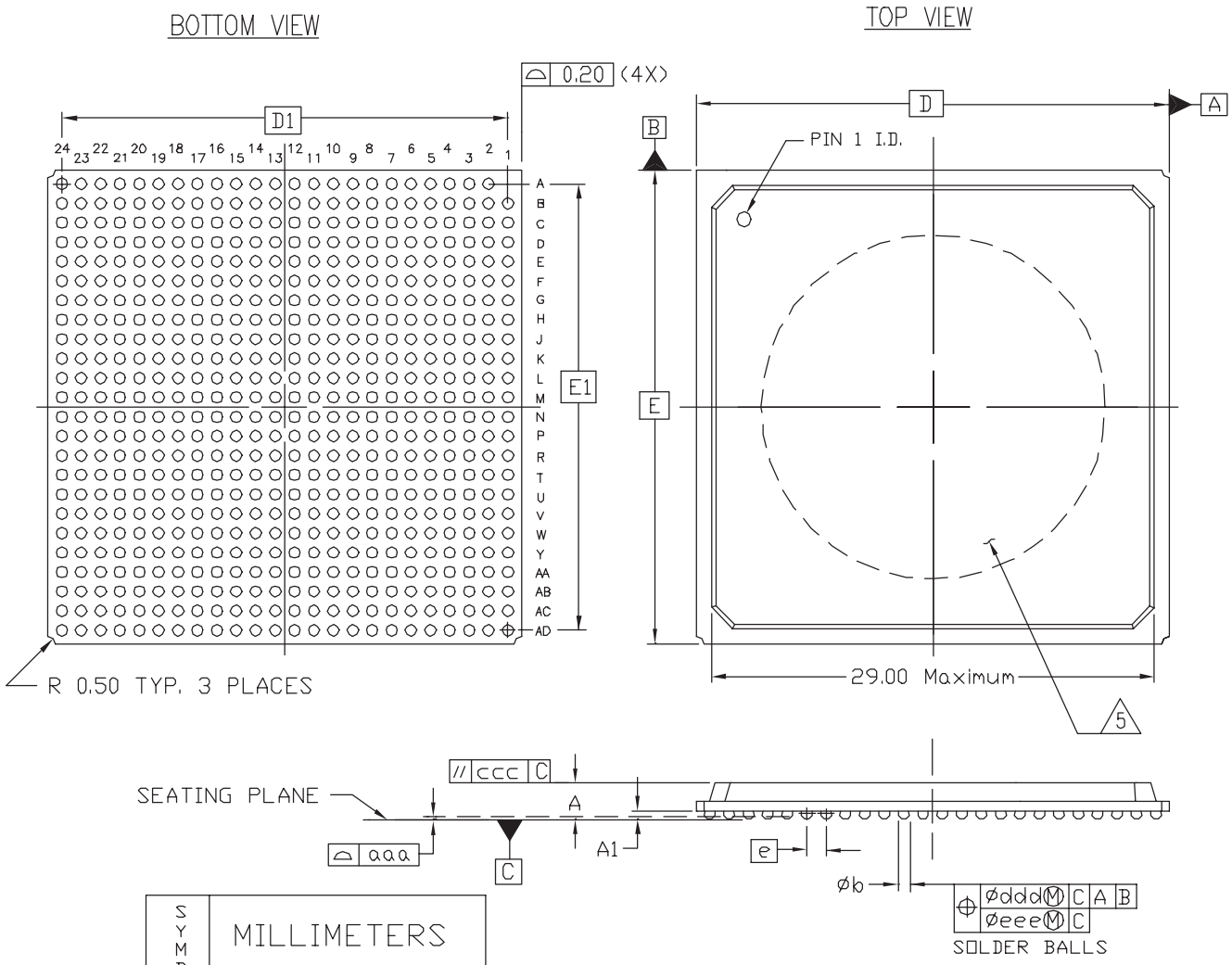
SYMBOL	MILLIMETERS		
	MIN.	NOM.	MAX.
A	---	2.25	2.60
A_1	0.40	0.50	0.60
D/E	27.00 BSC		
D_1/E_1	25.00 REF		
e	1.00 BSC		
øb	0.50	0.60	0.70
aaa	---	---	0.20
ccc	---	---	0.35
ddd	---	---	0.30
eee	---	---	0.10
M	26		

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–AAL–1

Figure 5-34: **FG676 Fine-Pitch BGA Package**

BG575 Standard BGA Package (1.27 mm Pitch)



SYMBOL	MILLIMETERS		
	MIN.	NOM.	MAX.
A		2.33	2.60
A_1	0.50	0.60	0.70
D/E	31.00 BSC		
D_1/E_1	29.21 REF		
e	1.27BSC		
øb	0.60	0.75	0.90
aaa			0.20
ccc			0.35
ddd			0.30
eee			0.15
M	24		

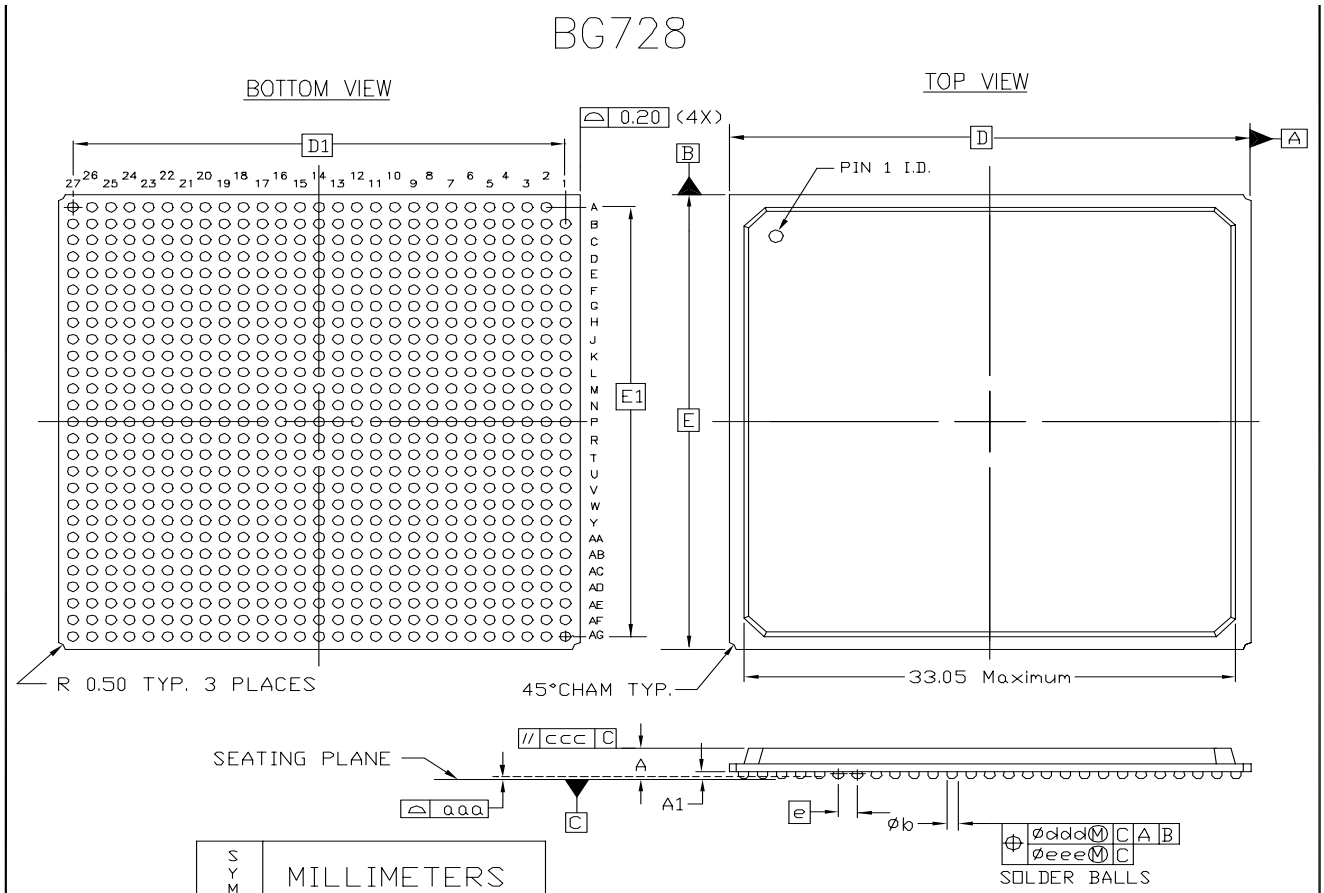
NOTES:

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 BAN–1 (DEPOPULATED)
4. NO SOLDER BALL AND LAND ON A1.

5 HEAT SINK OPTION

Figure 5-35: **BG575 Standard BGA Package**

BG728 Standard BGA Package (1.27 mm Pitch)



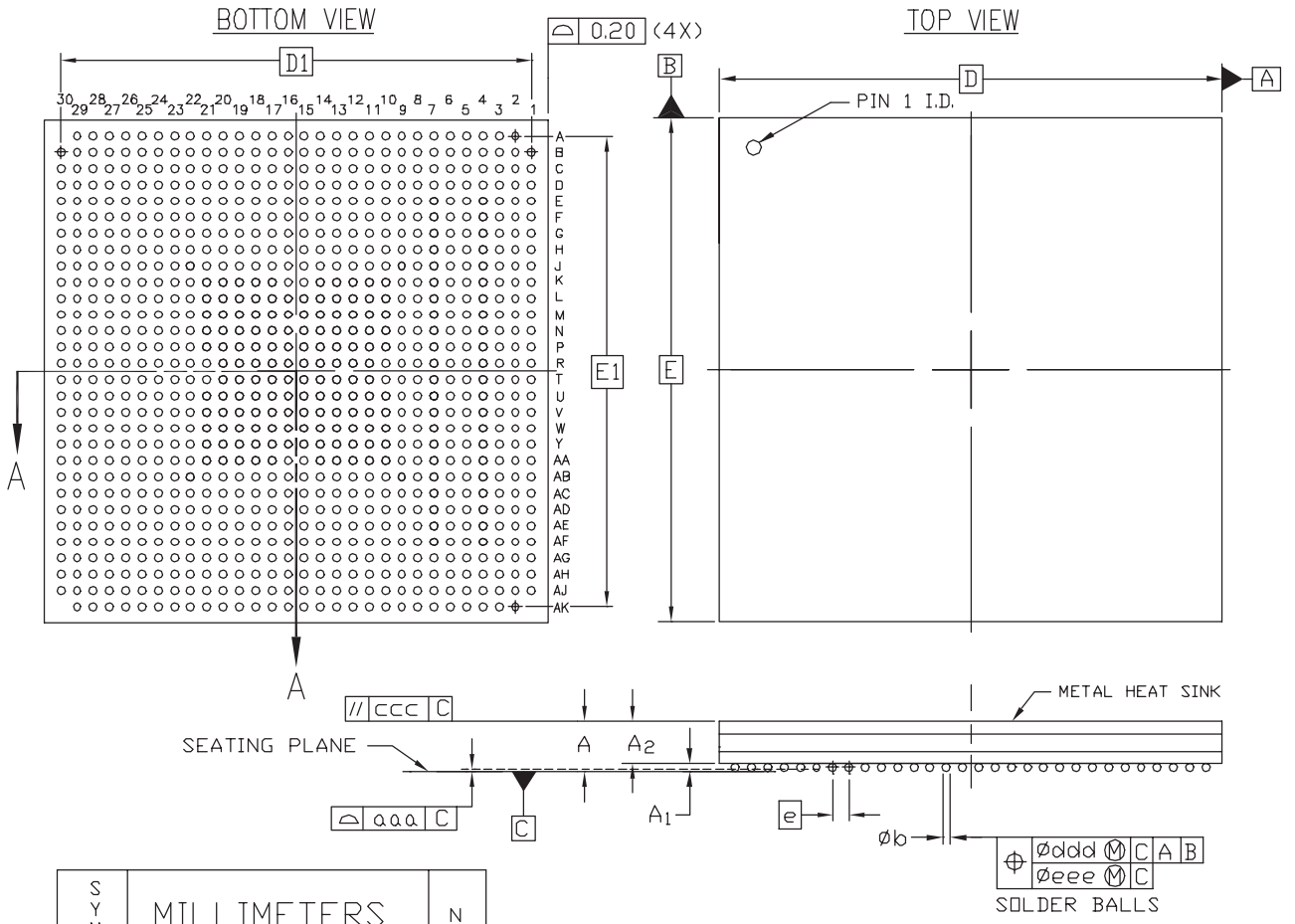
SYMBOL	MILLIMETERS		
	MIN.	NOM.	MAX.
A	---	2.33	2.60
A_1	0.50	0.60	0.70
D/E	35.00 BSC		
D_1/E_1	33.02 REF		
e	1.27BSC		
øb	0.60	0.75	0.90
aaa	---	---	0.20
ccc	---	---	0.35
ddd	---	---	0.30
eee	---	---	0.15
M	27		

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–BAR–1
4. NO SOLDER BALL AND LAND ON A1.

Figure 5-36: **BG728 Standard BGA Package**

FF896 Flip-Chip Fine-Pitch BGA Package (1.00 mm Pitch)



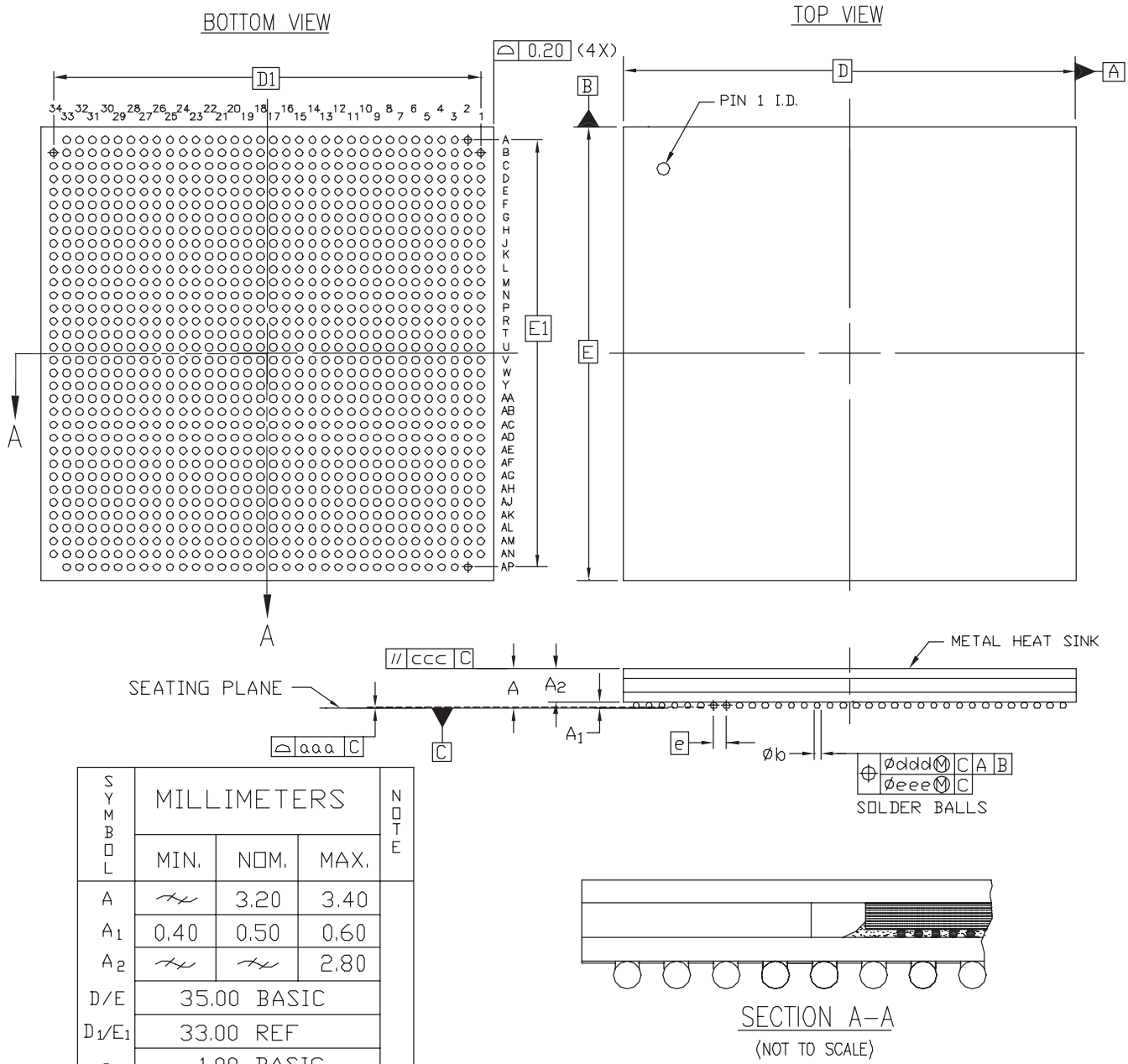
SYMBOL	MILLIMETERS			NOTE
	MIN.	NOM.	MAX.	
A		3.20	3.40	
A_1	0.40	0.50	0.60	
A_2			2.80	
D/E	31.00 BASIC			
D_1/E_1	29.00 REF			
e	1.00 BASIC			
øb	0.50	0.60	0.70	
aaa			0.20	
ccc			0.35	
ddd			0.30	
eee			0.10	
M	30			2

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–AAN–1 (DEPOPULATED)

Figure 5-37: **FF896 Flip-Chip Fine-Pitch BGA Package**

FF1152 Flip-Chip Fine-Pitch BGA Package (1.00 mm Pitch)



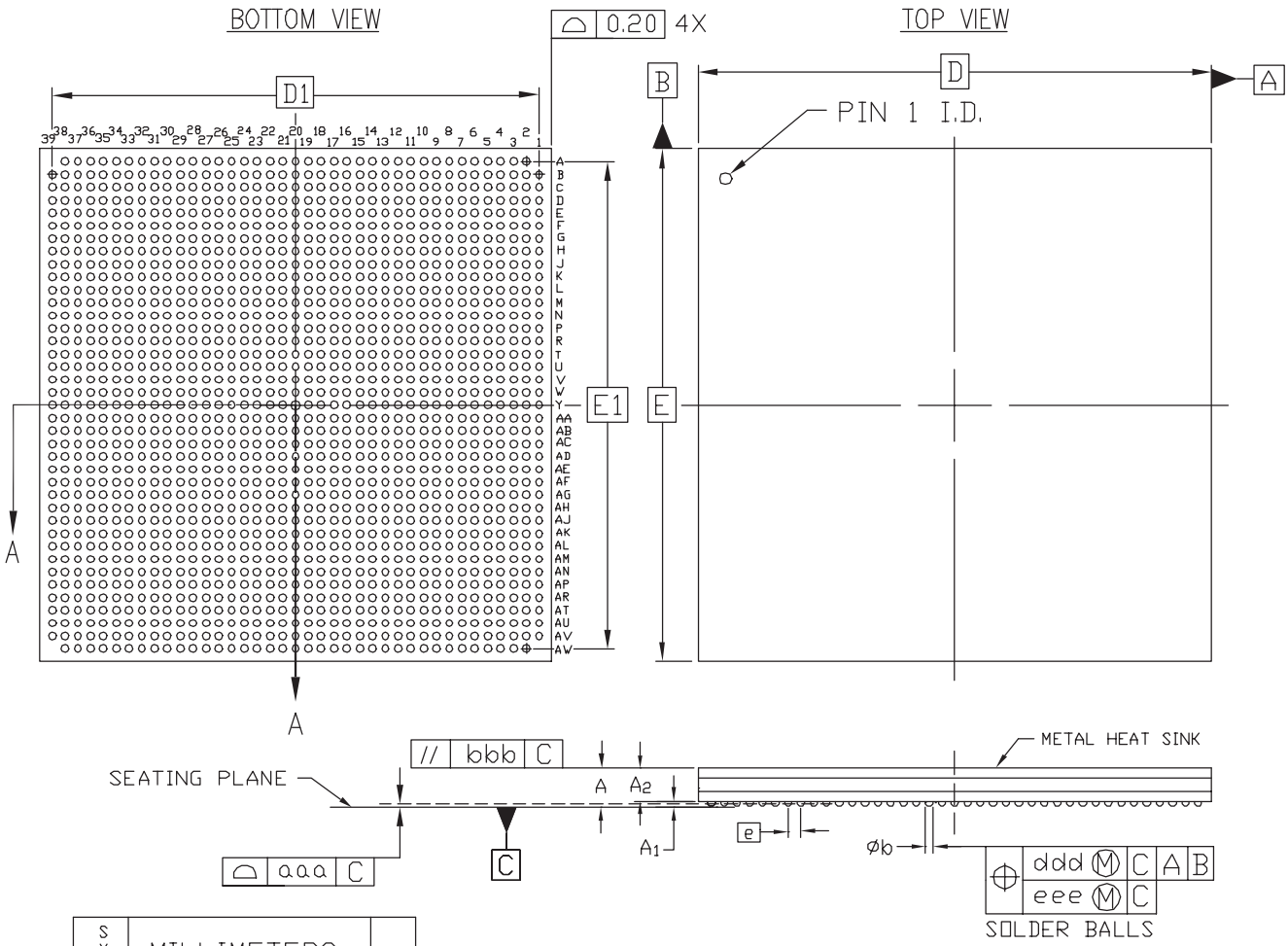
SYMBOL	MILLIMETERS			NOTE
	MIN.	NOM.	MAX.	
A	---	3.20	3.40	
A_1	0.40	0.50	0.60	
A_2	---	---	2.80	
D/E	35.00 BASIC			
D_1/E_1	33.00 REF			
e	1.00 BASIC			
øb	0.50	0.60	0.70	
aaa	---	---	0.20	
ccc	---	---	0.35	
ddd	---	---	0.30	
eee	---	---	0.10	
M	34			2

NOTES:

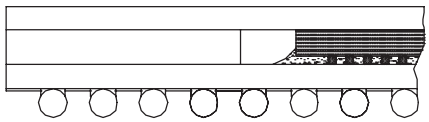
1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M–1994
2. SYMBOL 'M' IS THE BALL MATRIX SIZE.
3. CONFORMS TO JEDEC MS–034–AAR–1 (DEPOPULATED)

Figure 5-38: **FF1152 Flip-Chip Fine-Pitch BGA Package**

FF1517 Flip-Chip Fine-Pitch BGA Package (1.00 mm Pitch)



SYMBOL	MILLIMETERS			NOTE
	MIN.	NOM.	MAX.	
A		3.20	3.40	
A_1	0.40	0.50	0.60	
A_2			2.80	
D/E	40.00 BASIC			
D_1/E_1	38.00 REF			
e	1.00 BASIC			
øb	0.50	0.60	0.70	
aaa			0.20	
bbb			0.25	
ddd			0.25	
eee			0.10	
M		39		2



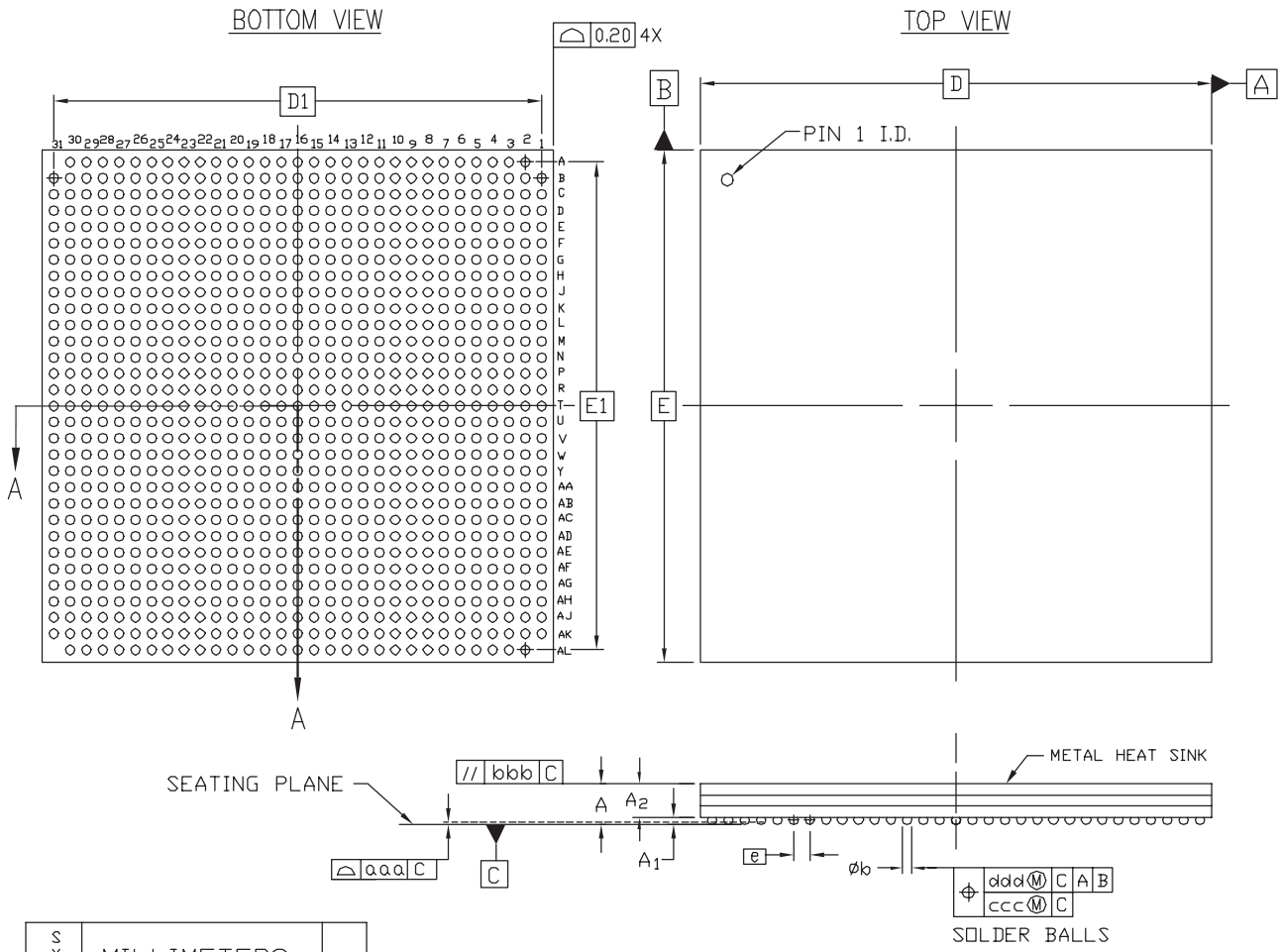
SECTION A–A
(NOT TO SCALE)

NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ASME Y14.5M–1994
2. SYMBOL "M" IS THE PIN MATRIX SIZE.
3. CONFORMS TO JEDEC MS–034–AAU–1 (DEPOPULATED)

Figure 5-39: **FF1517 Flip-Chip Fine-Pitch BGA Package**

BF957 Flip-Chip BGA Package (1.27 mm Pitch)



SYMBOL	MILLIMETERS			NOTE
	MIN.	NOM.	MAX.	
A		3.25	3.50	
A_1	0.50	0.60	0.70	
A_2			2.80	
A_3	0.20			
D/E	40.00 BSC			
D_1/E_1	38.10 REF.			
e	1.27 BSC			
øb	0.60	0.75	0.90	
aaa			0.20	
bbb			0.25	
ccc			0.15	
ddd			0.30	
M	31			

NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M–1994
2. SYMBOL "M" IS THE PIN MATRIX SIZE.
3. CONFORMS TO JEDEC MS–034–BAU–1 (DEPOPULATED)

Figure 5-40: **BF957 Flip-Chip BGA Package**

Flip-Chip Packages

As silicon devices become more integrated with smaller feature sizes as well as increased functionality and performance, packaging technology is also evolving to take advantage of these silicon advancements. Flip-chip packaging is the latest packaging option introduced by Xilinx to meet the demand for high I/O count and high performance required by today's advanced applications.

Flip-chip packaging interconnect technology replaces peripheral bond pads of traditional wire-bond interconnect technology with area array interconnect at the die/substrate interface.

The area array pads contain wettable metallization for solders (either eutectic or high-lead), where a controlled amount of solder is deposited either by plating or screen-printing. These parts are then reflowed to yield bumped dies with relatively uniform solder bumps spread over the surface of the device. Unlike traditional packaging in which the die is attached to the substrate face up and the connection is made by using wire, the bumped die in a flip-chip package is flipped over and placed face down, with the conductive bumps connecting directly to the matching metal pads on the ceramic or organic laminate substrate. The solder material at molten stage is self-aligning and produces good joints even if the chip is placed offset on the substrate.

Flip-chip packages are assembled on high-density, multi-layer ceramic or organic laminate substrates. Since flip-chip bump pads are in area array configuration, very fine lines and geometry on the substrates are required to be able to successfully route the signals from the die to the periphery of the substrates. Multi-layer build-up structures offer this layout flexibility on flip-chip packages, and they provide improvements in power distribution and signal transmission characteristics.

Advantages of Flip-Chip Technology

Flip-chip interconnections in combination with the advanced multi-layer laminated substrates provide superior performance over traditional wire-bond packaging. Benefits include:

- Easy access to core power/ground and shorter interconnects, resulting in better electrical performance
- Better noise control since the inductance of flip-chip interconnect is lower
- Excellent thermal performance due to direct heatsinking to backside of the die
- Higher I/O density since bond pads are in area array format
- Smaller size

Thermal Data

Thermal Considerations

Due to the variety of applications in which Virtex-II FPGA devices are likely to be used, it is traditionally a challenge to predict power requirements, and thus the thermal management needs, of a particular application. Virtex-II devices in general are characterized by high I/O counts and very high user gate counts. The attributes that make the devices popular with users also give the devices the potential of being clocked fast, which results in high power consumption. Because of this high heat-generating potential, the Virtex-II package offering includes medium and high power capable packaging options (see Table 5-3).

Table 5-3 shows junction-to-ambient, junction-to-case, and junction-to-board thermal resistance parameters and estimated power consumption for Virtex-II packages. These values were derived using typical thermal management assumptions, stated in the table. Table 5-3 provides only an example and is not intended as a maximum power dissipation specification.

Table 5-3: **Thermal Data for Virtex-II Packages**

Package	Lead Pitch (mm)	Junction to Ambient Theta-J_A Range °C/Watt in Air	Junction to Case Theta-J_C Typical °C/Watt	Junction to Board Psi-J_B ("Theta-J_B") Typical °C/Watt	Max Power Bare Pkg (Watts) T_A = 50 °C T_{JMAX} = 100 °C	Power With Heatsink (Watts) Theta-SA = 1.5 °C/Watt Theta-cs = 0.1 °C/Watt T_A = 50° C T_J = 100° C
CS144 Flex Based 12x12	0.8	32 - 36	1	20	1.5	N/A(1)
FG256 2- 4L PCB 17x17	1.0	30 -35	3.5	19	1.5	N/A(1)
FG456 4L PCB 23x23	1.0	15 - 28	2.0	11	2.4	N/A(1)
FG676 4L PCB 27x27	1.0	14 -22	1.8	9	2.8	15
BG575 4L PCB 31x31	1.27	13 - 20	1.6	7	3.1	16
BG728 4L PCB 35x35	1.27	12 -20	1.5	6	3.3	16
BF957 40x40 Flip-Chip	1.27	8 - 13	0.7	3	5.0	22
FF896 31x31 Flip-Chip	1.0	9 - 14	0.5	4	4.5	21
FF1152 35x35 Flip-Chip	1.0	8 - 13	0.5	4	4.5	21
FF1517 40x40 Flip-Chip	1.0	8 - 12	0.5	3	5.0	22

Notes:

1. The heat sink used in this example is not mechanically compatible with the CS144, FG256, and FG456 packages.

Virtex-II packages can be grouped into three broad performance categories: low, medium, and high, based on their power handling capabilities. All of the packages can use external thermal enhancements, which can range from simple airflow to schemes that can include passive as well as active heatsinks. This is particularly true for high-performance flip-chip packages where system designers have the option to further enhance the packages to handle in excess of 25 watts, with arrangements that take system physical constraints into consideration. Table 5-4 shows simple but incremental power management schemes that can be brought to bear on flip-chip packages.

Table 5-4: **Virtex-II Flip-Chip Thermal Management**

Power	Technique	Description
Low End (1 - 6 watts)	Bare package with moderate air 8 - 12 °C/Watt	Bare package. Package can be used with moderate airflow within a system.
Mid Range (4 - 10 watts)	Passive heatsink with air 5 - 10 °C/Watt	Package is used with various forms of passive heatsinks and heat spreader techniques.
High End (8 - 25 watts)	Active heatsink 2 - 3 °C/Watt or better	Package is used with active heatsinks, TEC, and board-level heat spreader techniques

Thermal Management Options

The following are thermal management options to consider:

- For moderate power dissipation (2 to 6 watts), the use of passive heatsinks and heatspreaders attached with thermally conductive double-sided tapes or retainers can offer quick thermal solutions.
- The use of lightweight finned external passive heatsinks can be effective for dissipating up to 10 watts. If implemented with forced air as well, the benefit can be a 40% to 50% increase in heat handling efficiency over bare packages. The more efficient external heatsinks tend to be tall and heavy. To help protect component joints from bulky heatsink induced stresses, the use of spring loaded pins or clips that transfer the mounting stress to a circuit board is advisable. The diagonals of some of these heatsinks can be designed with extensions to allow direct connections to the board.
- Flip-chip packages: All flip-chip packages are thermally enhanced BGAs with die facing down. They are offered with exposed metal heatsink at the top. These high-end thermal packages lend themselves to the application of external heatsinks (passive or active) for further heat removal efficiency. Again, precaution should be taken to prevent component damage when a bulky heatsink is attached.
- Active heatsinks can include a simple heatsink incorporating a mini fan or even a Peltier Thermoelectric Cooler (TECs) with a fan to blow away any heat generated. Any considerations to apply TEC in heat management should require consultation with experts in using the device, since these devices can be reversed and cause damage to the components. Also, condensation can be an issue.
- Molded packages (FG456, FG676, BG575, BG728, and so forth) with or without exposed metal at the top can also use heatsinks at the top for further heat removal. These BGA packages are similar in construction to those used in Graphics cards in PC applications, and heatsinks used for those applications can easily be used for these packages, as well. In this case, the Junction-to-Case resistance is the limiting consideration.
- Outside the package itself, the board on which the package sits can have a significant impact on thermal performance. Board designs can be implemented to take advantage of a board's ability to spread heat. The effect of the board is dependent on its size and how it conducts heat. Board size, the level of copper traces on it, and the number of buried copper planes all lower the junction-to-ambient thermal resistance for packages mounted on the board.

  The junction-to-board thermal resistance for Virtex-II packages are given in Table 5-3. A standard JEDEC type board was used for obtaining the data. Users need to be aware that a direct heat path to the board from a component also exposes the component to the effect of other heat sources - particularly if the board is not cooled effectively. An otherwise cooler component might be heated by other heat contributing components on the board.

Printed Circuit Board Considerations

Layout Considerations

The PC board is no longer just a means to hold ICs in place. At today's high clock rates and fast signal transitions, the PC board performs a vital function in feeding stable supply voltages to the IC and in maintaining signal integrity between devices.

VCC and Ground Planes

Since CMOS power consumption is dynamic, it is a non-trivial task to assure stable supply voltages at the device pins and to minimize ground differentials. A multi-layer PC board is a must, with four layers for the simplest circuits, 6 to 12 layers for typical boards. Ground and V_{CC} must each be distributed in complete layers with few holes. Slots in these layers would cause an unacceptable inductive voltage drop, when the supply current changes at a rate of 1 A/ns, or even faster. Besides an uninterrupted ground plane, Virtex-II devices require one plane for V_{CCINT} (1.5 V) plus one plane for V_{CCAUX} (3.3 V). V_{CCO} can be distributed on wide signal traces with sufficient bypass capacitors.

Beyond low resistance and inductance, ground and V_{CC} planes combined can also provide a small degree of V_{CC} decoupling. The capacitance between two planes is ~100 pF/inch^2 or ~15 pF/cm^2, assuming 10 mil (0.25 mm) spacing with FR4 epoxy.

V_{CC} Decoupling

Fast changing Icc transitions must be supplied by local decoupling capacitors, placed very closely to the V_{CC} device pins or balls. These capacitors must have sufficient capacitance to supply Icc for a few ns and must have low intrinsic resistance and inductance. X7R or NPO ceramic surface-mounted capacitors of 0.01 to 0.1 µF, one per V_{CC} device pin, are appropriate. 0.1 µF can supply 1A for 2ns with a 20 mV voltage droop.

1A • 2ns = 2 nanocoulomb = 100 nF • 0.02 V

Low impedance at >100 MHz is important, but capacitance variation with temperature is acceptable. These small capacitors are the first-line source for Icc, and they must be placed very close to the V_{CC} pins. A half-inch or 10 mm trace represents an inductance of several nanohenries, defeating the purpose of the decoupling capacitor. Backing up this local decoupling is one tantalum capacitor of 10 to 100 µF, able to supply multiple amperes for about 100 ns.

Finally, each board needs a power-supply decoupling electrolytic capacitor of 1000 to 10,000 µF able to supply even more current for a portion of the supply switching period. As described below, larger capacitors inevitably have higher series resistance and inductance, which is the reason for the above-mentioned hierarchy of supply decoupling. As a general rule, multiple capacitors in parallel always offer lower resistance and inductance than any single capacitor.

Decoupling Capacitors

The ideal decoupling capacitor would present a short circuit to ground for all ac signals. A real capacitor combines a given amount of capacitance with unavoidable parasitics, a small series resistance and inductance. At low frequencies, the composite impedance is capacitive, i.e., it decreases with increasing frequency. At high frequencies, it is inductive and increases with frequency, making the decoupling ineffective. In-between, there is the LC resonant frequency, where the capacitor looks like a small resistor.

Different technologies provide different trade-offs between desirable features like small size and high capacitance, and undesirable features like series resistance and inductance. Electrolytic and tantalum capacitors offer the largest capacitance in a given physical size, but also have the highest inductance. This makes them useful for decoupling low frequencies and storing large amounts of charge, but useless for high frequency decoupling. Surface-mount ceramic capacitors, on the other hand, offer the lowest

inductance and the best high-frequency performance, but offer only a small amount of capacitance, less than a microfarad.

Figure 5-41 shows the frequency-dependent impedance and resistance of a typical electrolytic capacitor of 1500 µF, while Figure 5-42 and Figure 5-43 show the equivalent data for ceramic bypass capacitors of 33,000 and 3,300 pF, respectively. Note that the resonant frequency for the small ceramic bypass capacitor at 100 MHz is 10,000 times higher than the resonance frequency of the large electrolytic capacitor at 10 KHz. For more technical information on decoupling capacitors, see the manufacturers' websites.

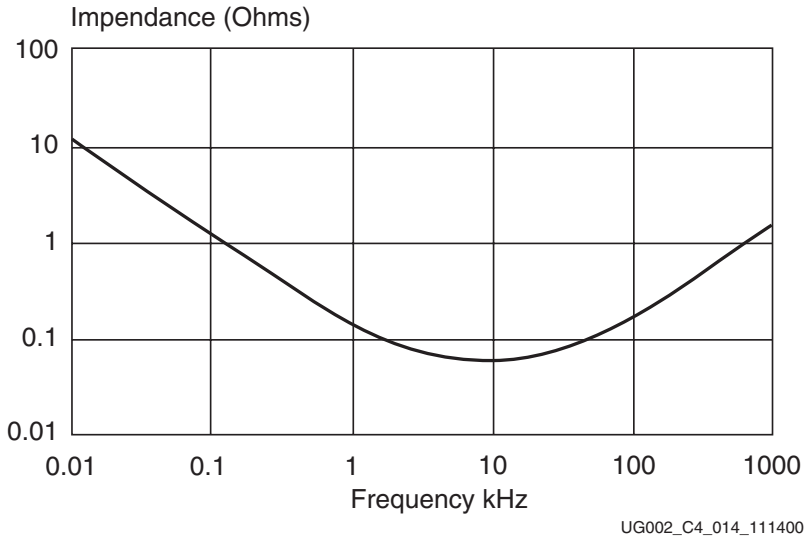


Figure 5-41: **1500 µF Electrolytic Capacitor Frequency Response Curve**

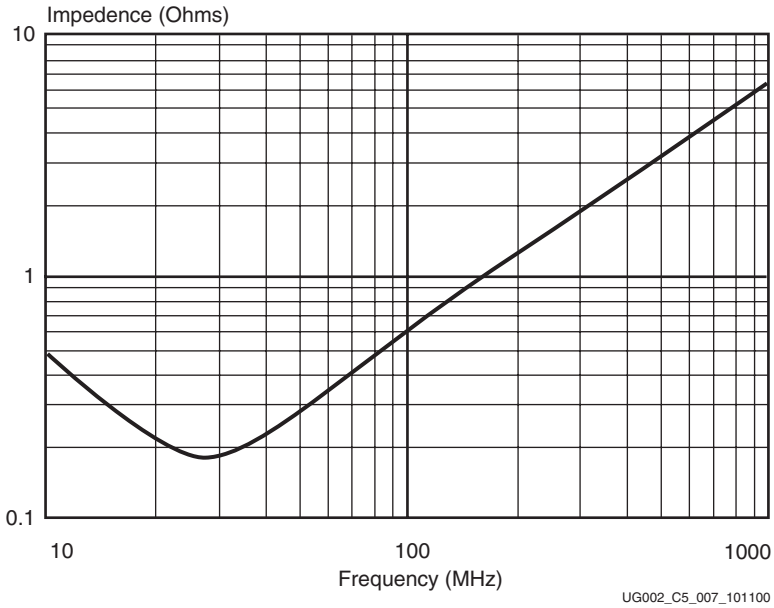


Figure 5-42: **33000 pF X7R Component Frequency Response Curve**

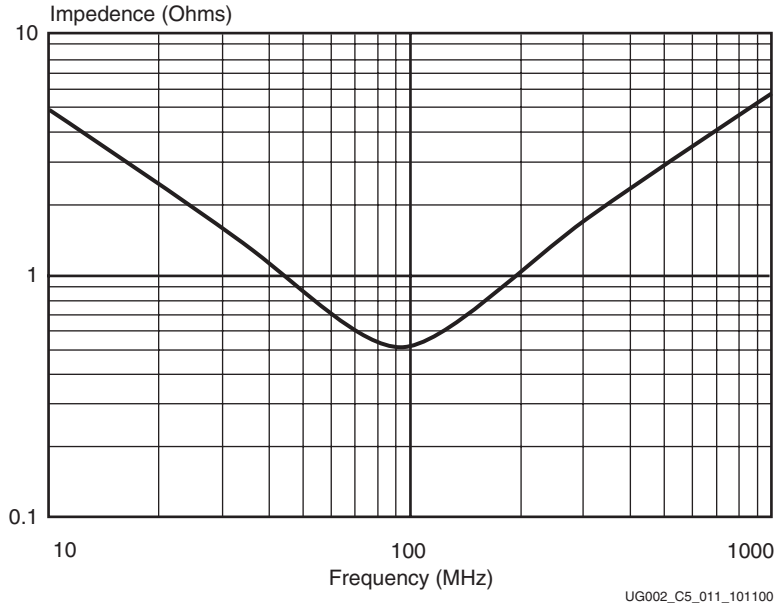


Figure 5-43: **3300 pF X7R Component Frequency Response Curve**

Transmission Line Reflections and Terminations

A PC board trace must be analyzed as a transmission line. Its series resistance and parallel conductance can generally be ignored, but series inductance and parallel capacitance per unit length are important parameters. Any signal transition (rising or falling edge) travels along the trace at a speed determined by the incremental inductance and capacitance.

For an outer-layer trace (air on one side) the propagation delay is 140 ps/inch, or 55 ps/cm. For an inner-layer trace (FR4 with ε=4.5 on both sides), the propagation delay is 180 ps/inch, or 70 ps/cm.

The voltage-to-current ratio at any point along the transmission line is called the characteristic impedance Z_0. It is determined by w/d, the ratio of trace width w to the distance d above the ground or V_{CC} plane.

For an outer layer trace (microstrip),

Z_0=50 Ω when w = 2d (e.g., w = 12 mil, d = 6 mil),

Z_0=75 Ω when w = d (e.g., both 6 mil = 0.15 mm).

For an inner layer trace between two ground or V_{CC} planes (stripline),

Z_0=50 Ω when w = 0.6•d (e.g., w = 5 mil, d = 8 mil),

Z_0=75 Ω when w = 0.25•d (impractical).

Most signal traces fall into the range of 40 to 80 Ω.

A slow transition treats a short narrow trace as a lumped capacitance of about 2 pF per inch (0.8 pF per cm). However, if the trace is so long, or the signal transition is so fast that the potential echo from the far end arrives after the end of the transition, then the trace must be analyzed as a transmission line.

In this case, the driver sees the trace not as a lumped capacitance, but rather as a pure resistance of Z_0. The signal transition then travels along the trace at the speed mentioned above. At any trace-impedance discontinuity all or part of the signal is reflected back to the origin. If the far end is resistively terminated with R=Z_0, then there is no reflection. If, however, the end is open, or loaded with only a CMOS input, then the transition doubles in amplitude, and this new wave travels back to the driver, where it may be reflected again, resulting in the familiar ringing. Such ringing has a serious impact on signal integrity, reduces noise margins, and can lead to malfunction, especially if an asynchronous signal or

a clock signal crosses the input threshold voltage unpredictably. Two alternate ways to avoid reflections and ensure signal integrity are parallel termination and series termination.

Parallel Termination

Reflections from the far end of the transmission line are avoided if the far end is loaded with a resistor equal to Z_0. A popular variation uses two resistors, one to V_{CC}, one to ground, as the Thevenin equivalent of Z_0. This reduces the load current for one signal level, while increasing it for the other. Parallel termination causes dc power consumption which can be eliminated by inserting a capacitor between the terminating resistor and ground. The value of this capacitor is determined as follows:

Signal transition time << RC << signal level duration

For example, 50 Ω • 120 pF for a 2 ns transition every 20 ns. See Figure 5-44.

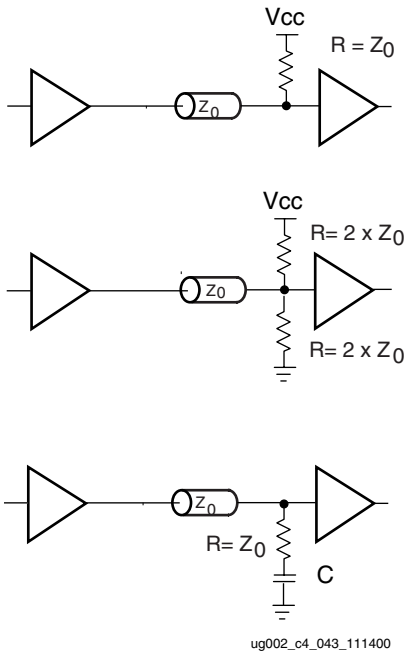


Figure 5-44: **Parallel Termination**

Series Termination

While parallel termination eliminates reflections, series termination relies on the reflection from the far end to achieve a full-amplitude signal. For series termination, the driver impedance is adjusted to equal Z_0, thus driving a half-amplitude signal onto the transmission line. At the unterminated far end, the reflection creates a full-amplitude signal, which then travels back to the driver where it gets absorbed, since the output impedance equals Z_0. See Figure 5-45.

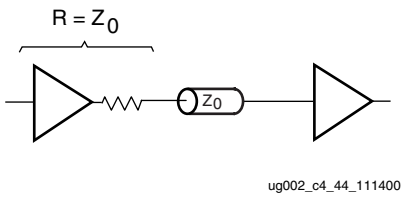


Figure 5-45: **Series Termination**

Series termination dissipates no dc power, but the half-amplitude round-trip delay signal means that there must be no additional loads along the line. Series termination is ideal (and only meaningful) for single-source-single-destination interconnects.

Virtex-II devices offer digitally controlled output impedance drivers and digitally-controlled input termination, thus eliminating the need for any external termination resistors. This feature is extremely valuable with high pin-count, high density packages.

These PC board considerations apply to all modern systems with fast current and voltage transitions, irrespective of the actual clock frequency. The designer of relatively slow systems is more likely caught off-guard by the inherent speed of modern CMOS ICs, where di/dt is measured in A/ns, dV/dt is measured in V/ns, and input flip-flops can react to 1 ns pulses, that are invisible on mid-range oscilloscopes. Powerful tools like HyperLynx can analyze signal integrity on the PC board and can often be amortized by one eliminated board-respin.

JTAG Configuration and Test Signals

Poor signal integrity and limitations of devices in a JTAG scan chain can reduce the maximum JTAG test clock (TCK) rate and reliability of JTAG-based configuration and test procedures. The JTAG TCK and test mode (TMS) signals must be buffered, distributed, and routed with the same care as any clock signal especially for long JTAG scan chains. The devices in a JTAG scan chain should be ordered such that the connections from the TDO of one device to the TDI of the next device are minimized. When high-speed JTAG-based configuration for the Virtex-II devices is required, devices with lower-specified maximum TCK rates can be placed in a separate JTAG scan chain.

Crosstalk

Crosstalk can happen when two signals are routed closely together. Current through one of the traces creates a magnetic field that induces current on the neighboring trace, or the voltage on the trace couples capacitively to its neighbor. Crosstalk can be accurately modeled with signal integrity software, but two easy to remember rules of thumb are:

- Crosstalk falls off with the square of increasing distance between the traces.
- Crosstalk also falls off with the square of decreasing distance to a ground plane.

$$\text{Peak Crosstalk Voltage} = \frac{\Delta V}{1 + (D/H)^2}$$

where

ΔV is the voltage swing

D is the distance between traces (center to center)

H is the spacing above the ground plane

Example:

3.3V swing, and two stripline traces 50 mils apart and 50 mils above the ground plane.

Peak Crosstalk Voltage = (3.3 V)/(1 + (0.05/0.05)^2) = 1.65 V

This can cause a false transition on the neighboring trace. Separating the trace by an additional 50 mils is significantly better:

Peak Crosstalk Voltage = (3.3 V)/(1 + (0.1/0.05)^2) = 0.66 V

Signal Routing to and from Package Pins

Signal escaping (traces leaving the pin/ball area) can be quite difficult for the large FG and flip-chip packages. The number of signal layers required to escape all the pins depends on the PCB design rules. The thinner the traces, the more signals per layer can be routed, and the fewer layers are needed. The thinner traces have higher characteristic impedance, so

choose an impedance plan that makes sense, and then be consistent. Traces from 40 to 80 ohms are common.

If only one signal can be escaped between two pads, only two rows of pins can be escaped per layer. For FG packages (1.0mm pitch) one signal of width 5 mils (0.13mm) can be escaped between two pads, assuming a space constraint equal to the trace width. For a discussion of signal routing specific to Virtex-II devices, see www.xilinx.com for currently available application notes.

As packages are able to handle more I/Os with a minimum increase in size, the signal integrity of those signals must be considered, regardless of clock frequency. Especially with the largest packages, precise PCB layer stackup is required. Parameters such as board material, trace width, pad type, and stackup must be defined based on simulation, and the fabrication drawings must be marked with "precise layer stackup" and the stackup specified. A number of board-level signal integrity simulators exist, and careful attention to PCB design rules creates a robust design with low EMI and high signal reliability.

Board Routability Guidelines

Board-Level BGA Routing Challenges

Xilinx ball grid array (BGA) wire-bond and flip-chip packages contain a matrix of solder balls (see Figure 5-46). These packages are made of multilayer BT substrates. Signal balls are in a perimeter format. Power and ground pins are grouped together appropriately.

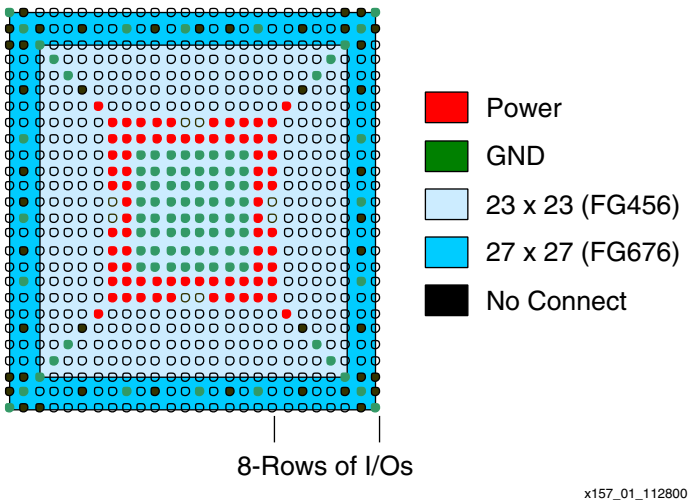


Figure 5-46: **Fine-Pitch BGA Pin Assignments**

The number of layers required for effective routing of these packages is dictated by the layout of pins in each package. If several other technologies and components are already present on the board, the system cost is factored with every added board layer. The intent of a board designer is to optimize the number of layers required to route these packages, considering both cost and performance. This section provides guidelines for minimizing required board layers for routing BGA products using standard PCB technologies (5 mils-wide lines and spaces or 6 mils-wide lines and spaces).

For high performance and other system needs, designers can use premium technologies with finer lines/spaces on the board. The pin assignment and pin grouping scheme in BGA packages enables efficient routing of the board with an optimum number of required board layers.

Board Routing Strategy

The diameter of a land pad on the component side is provided by Xilinx. This information is required prior to the start of board layout when designing the board pads to match component-side land geometry. Typical values for these land pads are described in Figure 5-47 and summarized in Table 5-5.

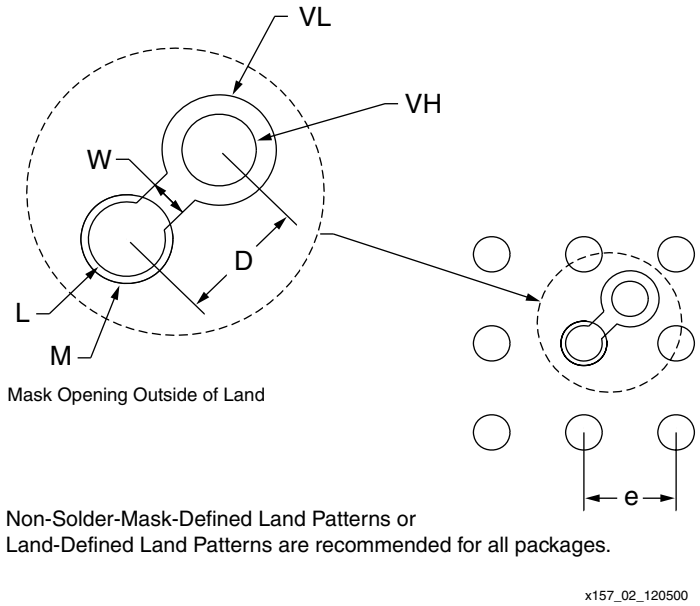


Figure 5-47: **Suggested Board Layout of Soldered Pads for BGA Packages**

Table 5-5: **Summary of Typical Land Pad Values (mm)**

Land Pad Characteristics	CS144	FG256	FG456	FG676	BG575	BG728	FF896	FF1152	FF1517	BF957
Component Land Pad Diameter (SMD)[4]	0.35	0.45	0.45	0.45	0.61	0.61	0.48	0.48	0.48	0.61
Solder Land (L) Diameter	0.33	0.40	0.40	0.40	0.56	0.56	0.45	0.45	0.45	0.56
Opening in Solder Mask (M) Diameter	0.44	0.50	0.50	0.50	0.66	0.66	0.55	0.55	0.55	0.66
Solder (Ball) Land Pitch (e)	0.80	1.00	1.00	1.00	1.27	1.27	1.00	1.00	1.00	1.27
Line Width Between Via and Land (w)	0.130	0.130	0.130	0.130	0.203	0.203	0.130	0.130	0.130	0.203
Distance Between Via and Land (D)	0.56	0.70	0.70	0.70	0.90	0.90	0.70	0.70	0.70	0.90
Via Land (VL) Diameter	0.51	0.61	0.61	0.61	0.65	0.65	0.61	0.61	0.61	0.65
Through Hole (VH), Diameter	0.250	0.300	0.300	0.300	0.356	0.356	0.300	0.300	0.300	0.356
Pad Array	-	Full	Full	Full	Full	Full	Full	Full	Full	Full
Matrix or External Row	13 x 13	16 x 16	22 x 22	26 x 26	24 x 24	27 x 27	30 x 30	34 x 34	39 x 39	31 x 31
Periphery Rows	4	-	7[3]	-	-	-	-	-	-	-

Notes:
1. Dimension in millimeters.
2. 3 x 3 matrix for illustration only, one land pad shown with via connection.
3. FG456 package has solder balls in the center in addition to the periphery rows of balls.
4. Component land pad diameter refers to the pad opening on the component side (solder-mask defined).

For Xilinx BGA packages, non-solder-mask defined (NSMD) pads on the board are suggested. This allows a clearance between the land metal (diameter L) and the solder mask opening (diameter M) as shown in Figure 5-47. The space between the NSMD pad and the solder mask, as well as the actual signal trace widths, depend on the capability of the PCB vendor. The cost of the PCB is higher when the line width and spaces are smaller.

Selection of pad types and sizes determines the available space between adjacent balls for signal escape. Based on PCB capability, the number of lines that can share the available space is described in Figure 5-48. Based on geometrical considerations, if one signal escapes between adjacent balls, then two signal rows can be routed on a single metal layer. This is illustrated in Figure 5-48 as routing with one line/channel, either at 6 mils-wide lines and spaces or 5 mils-wide lines and spaces. Using this suggested routing scheme, a minimum of eight PCB layers are required to route 10 signal rows in a package.

A slightly lower trace width can be used by the inner signal rows routed in internal layers than the width used in top and bottom external or exposed traces. Depending on the signal being handled, the practice of "necking down" a trace in the critical space between the BGA balls is allowable. Changes in width over very short distances can cause small impedance changes. Validate these issues with the board vendor and signal integrity engineers responsible for the design.

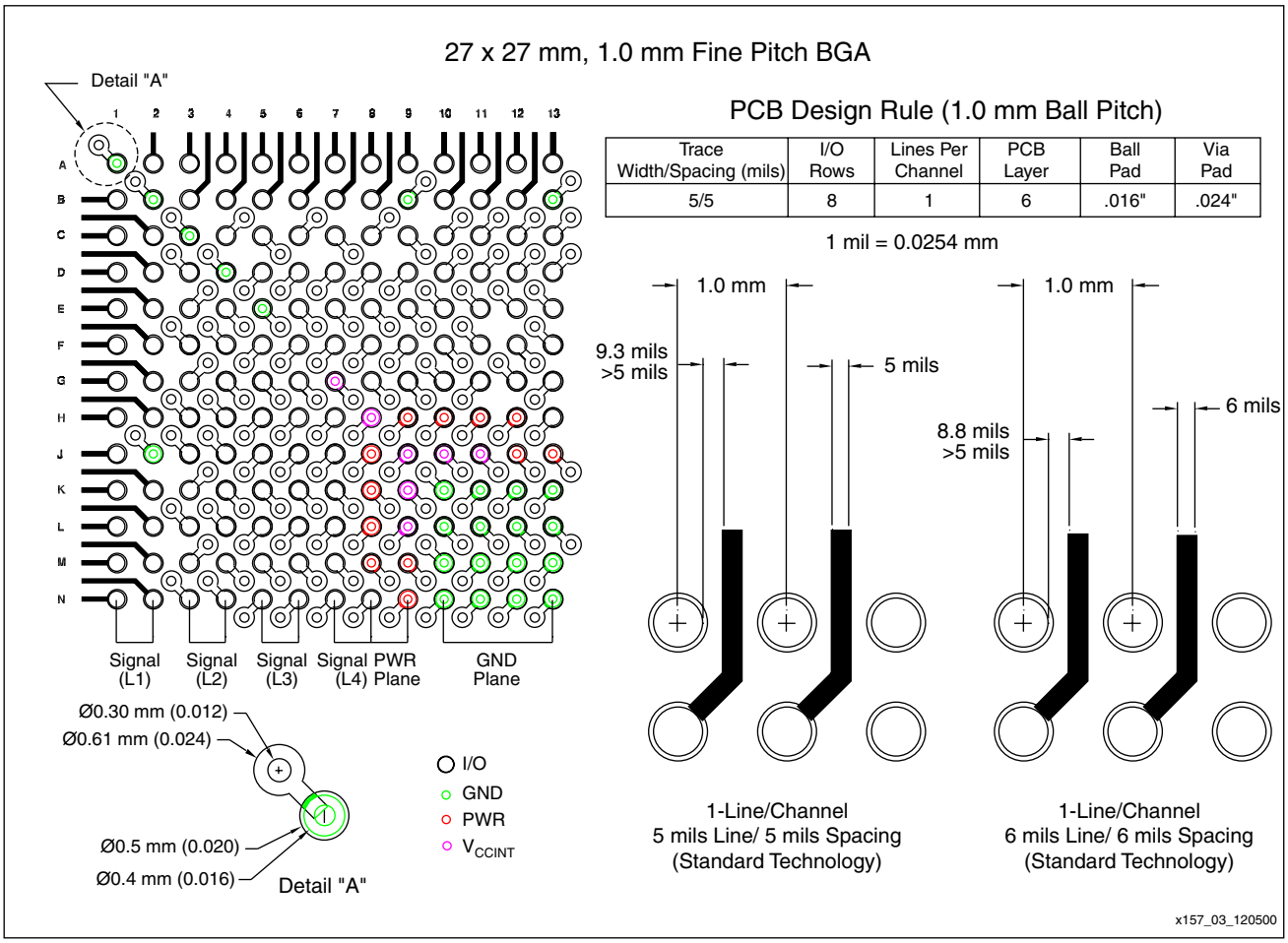


Figure 5-48: **FG676 PC Board Layout/Land Pattern**

Figure 5-48 describes a board-level layout strategy for a Xilinx 1.0 mm pitch FG676 package. Detail A in Figure 5-48 describes the opening geometry for the Land Pad and the Solder Mask. Routing with 5 mils-wide lines or spaces allows one signal per channel (between the balls). For successful routing, eight-row deep signal traces require six PCB layers. Figure 5-49 shows the suggested schematic of layers for the six-layer routing scheme.

Using premium board technology, such as Microvia Technology (allowing up to 4 mils-wide lines and spaces), efficient routing is possible with a reduced number of board layers. A grouping scheme for power, ground, control, and I/O pins, might also enable efficient routing.

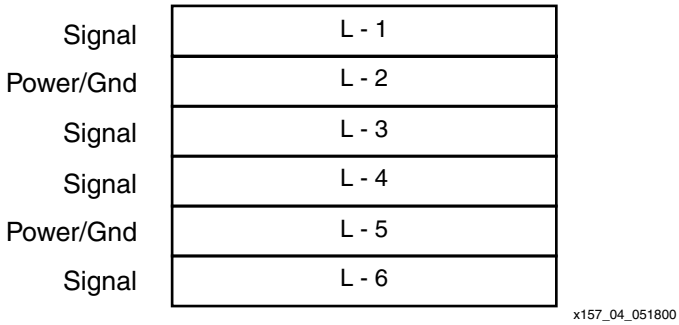


Figure 5-49: **Six-Layer Routing Scheme**

Figure 5-50 through Figure 5-67 show suggested layer-by-layer board routing for each Virtex-II package, including flip-chip packages. These drawings assume a standard PCB technology of 5 mils-wide lines and spaces. Table 5-6 lists the layer-by-layer routing examples provided. More details are contained in **XAPP157**, which is available on the web at **www.xilinx.com/bvdocs/appnotes/xapp157.pdf**, as is a full-color (PDF) version of this document.

Table 5-6: **Layer-By-Layer Board Routing Examples**

Package	Standard Routing	Routing With LVDS Pairs
FG256	Top and bottom layers	Top and bottom layers
FG456	Top, 2nd, and bottom layers	Top, 2nd, and bottom layers
FG676	Top, 2nd, 3rd, and bottom layers	Top, 2nd, 3rd, and bottom layers
BG575	Top, 2nd, and bottom layers	Top, 2nd, and bottom layers
BG728	Top, 2nd, 3rd, and bottom layers	Top, 2nd, 3rd, and bottom layers
FF896	Top, 2nd, 3rd, and bottom layers	Top, 2nd, 3rd, and bottom layers
FF1152	Top, 2nd, 3rd, 4th, and bottom layers	Top, 2nd, 3rd, 4th, and bottom layers
FF1517	Top, 2nd, 3rd, 4th, 5th, and bottom layers	Top, 2nd, 3rd, 4th, 5th, and bottom layers
BF957	Top, 2nd, 3rd, and bottom layers	Top, 2nd, 3rd, and bottom layers

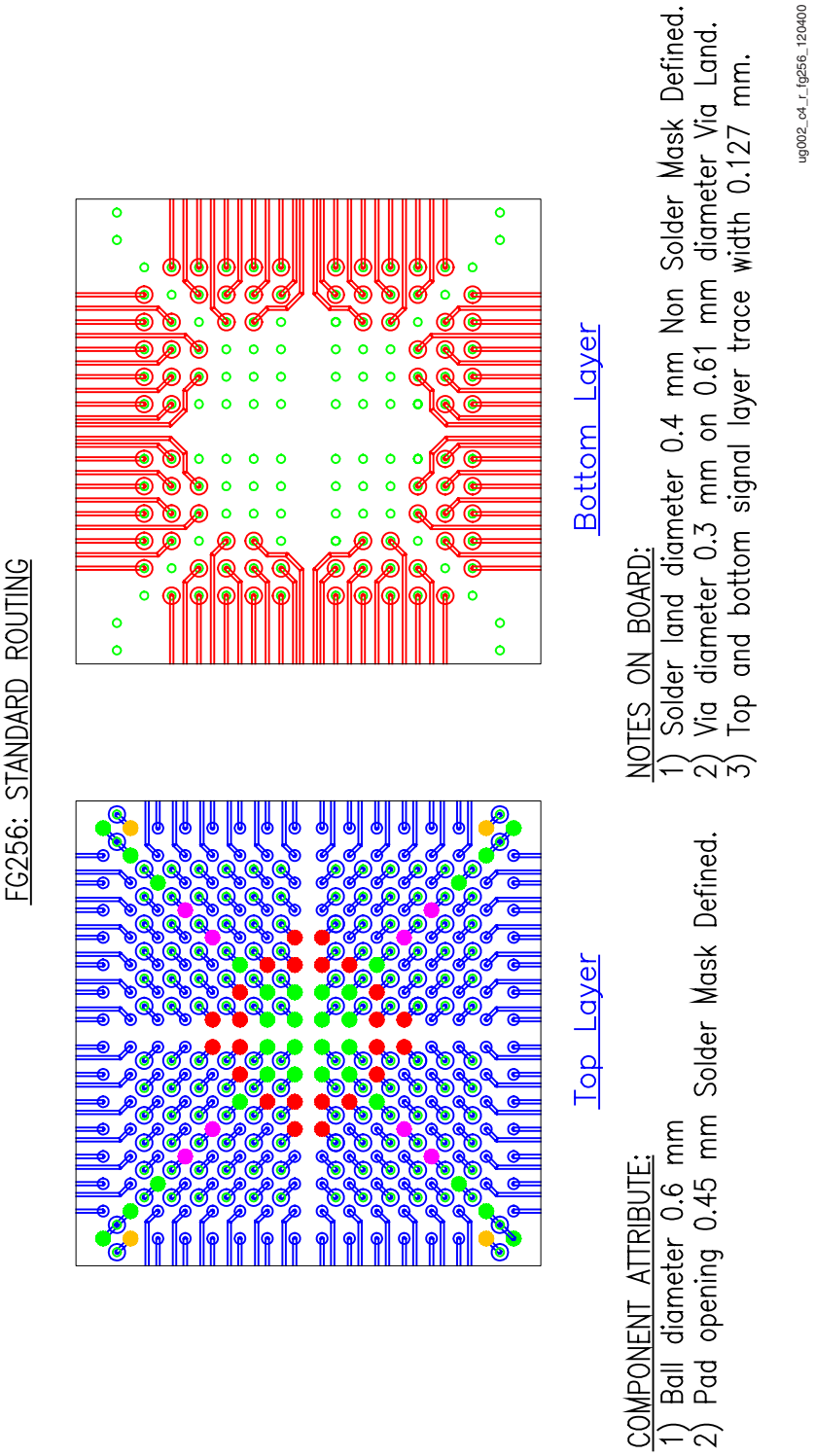


Figure 5-50: **FG256 Standard Routing**

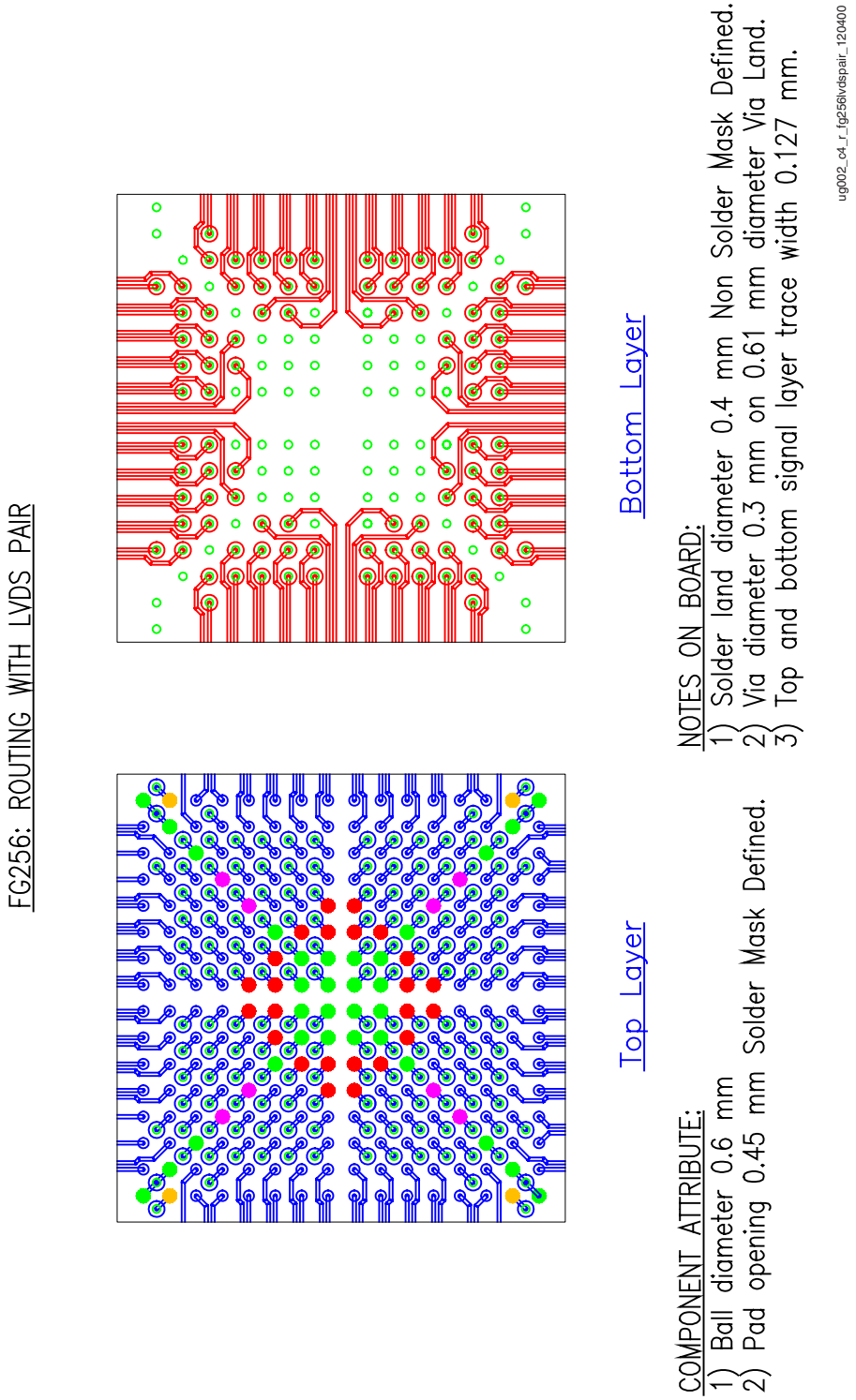


Figure 5-51: **FG256 Routing With LVDS Pairs**

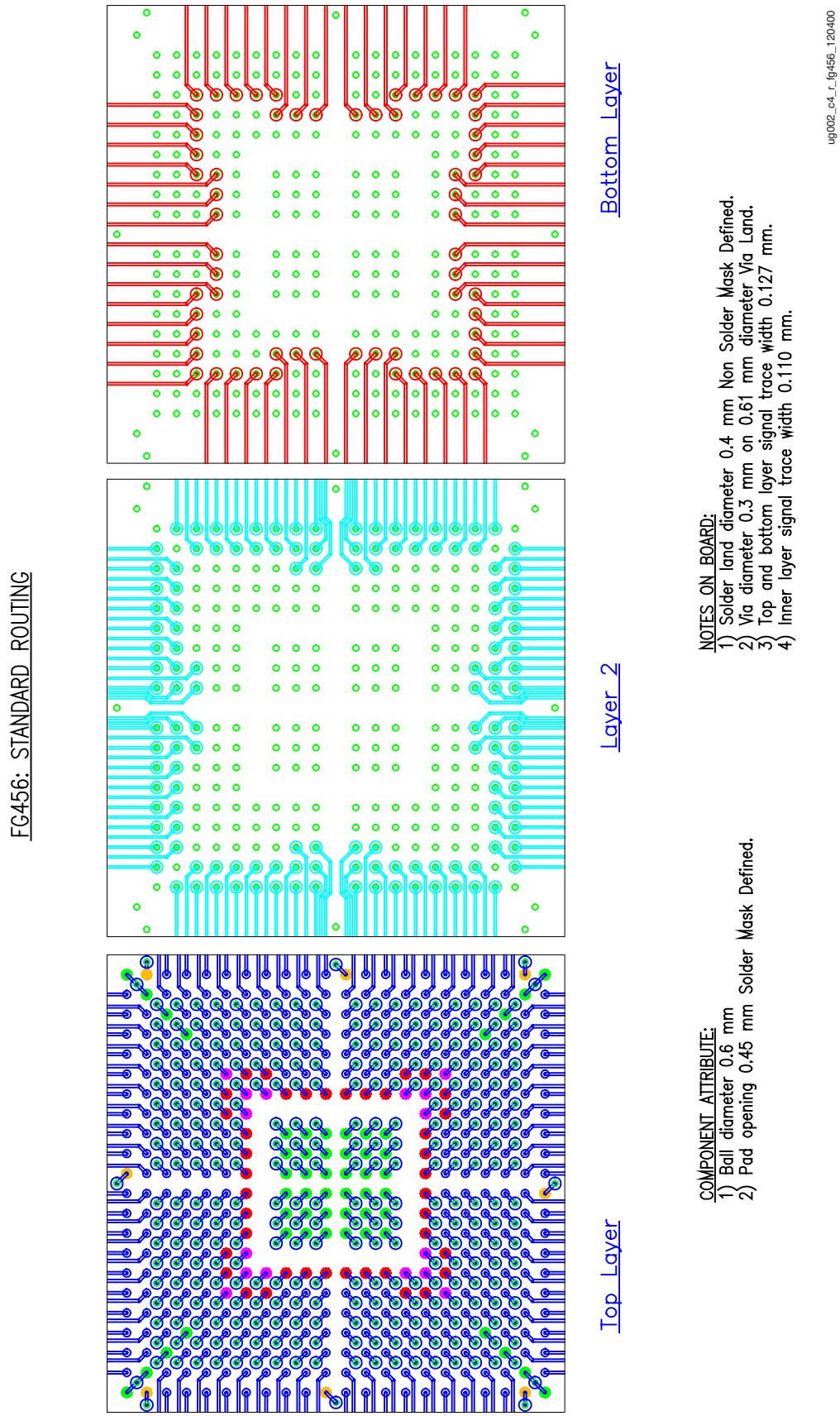


Figure 5-52: **FG456 Standard Routing**

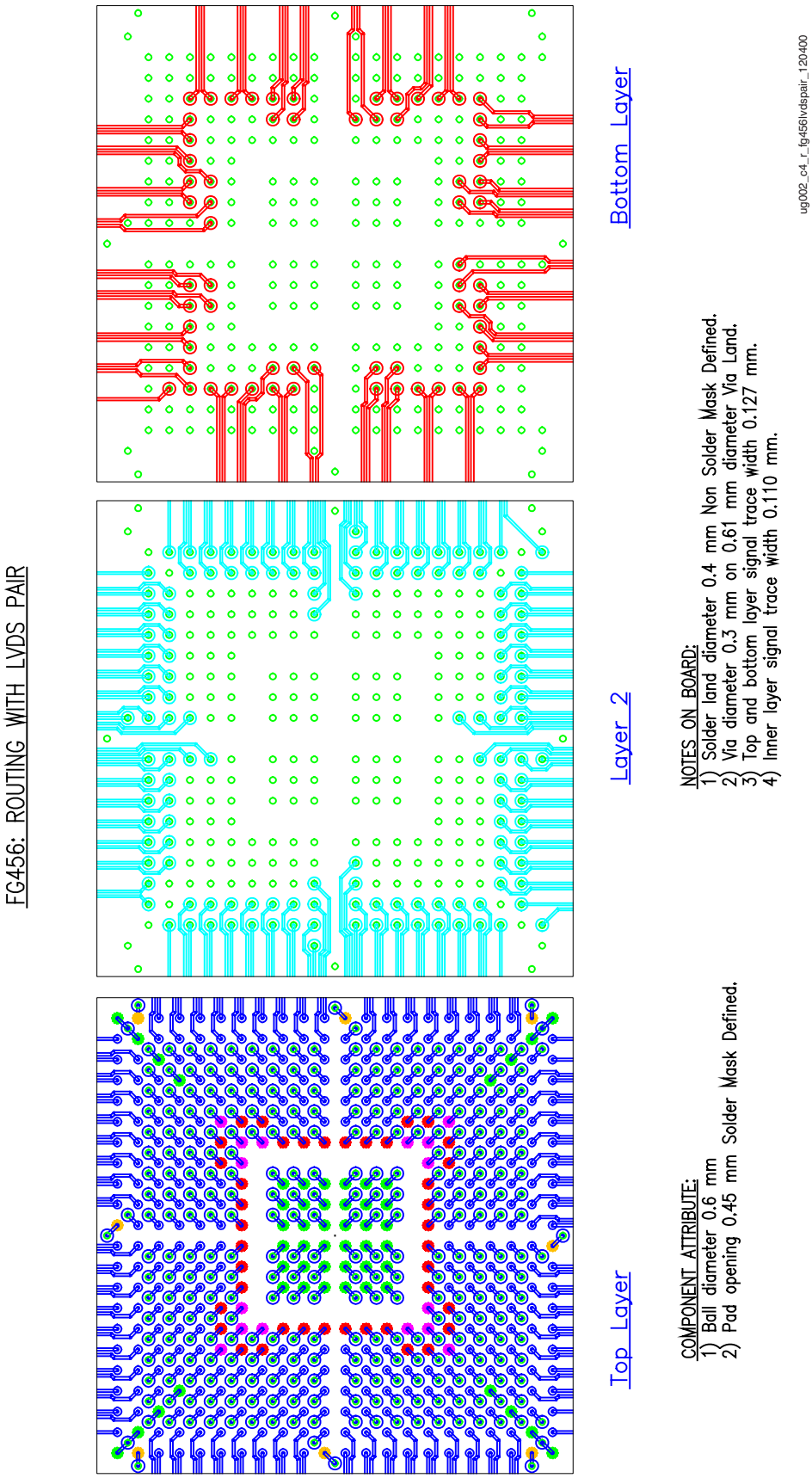


Figure 5-53: **FG456 Routing With LVDS Pairs**

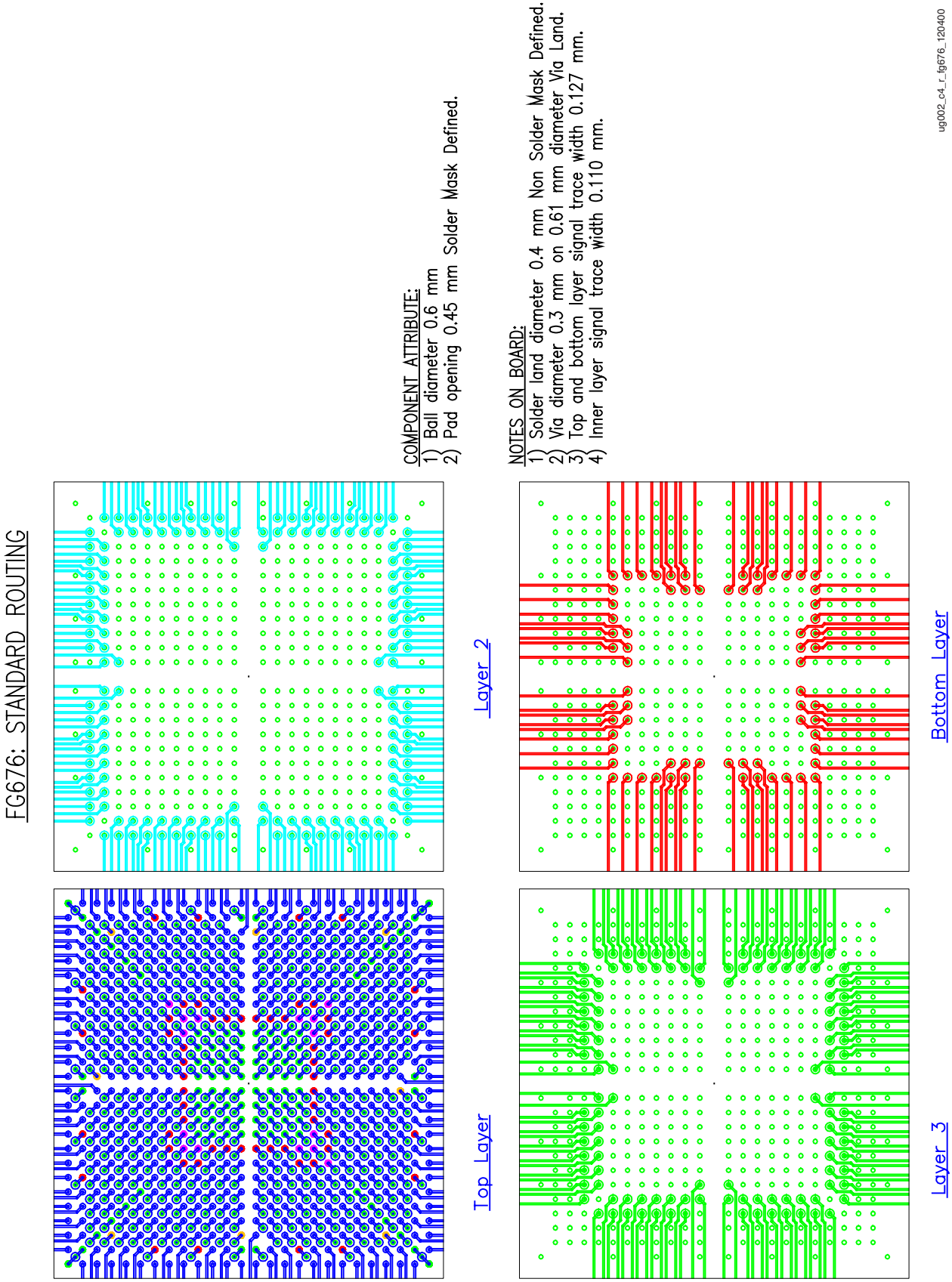


Figure 5-54: **FG676 Standard Routing**

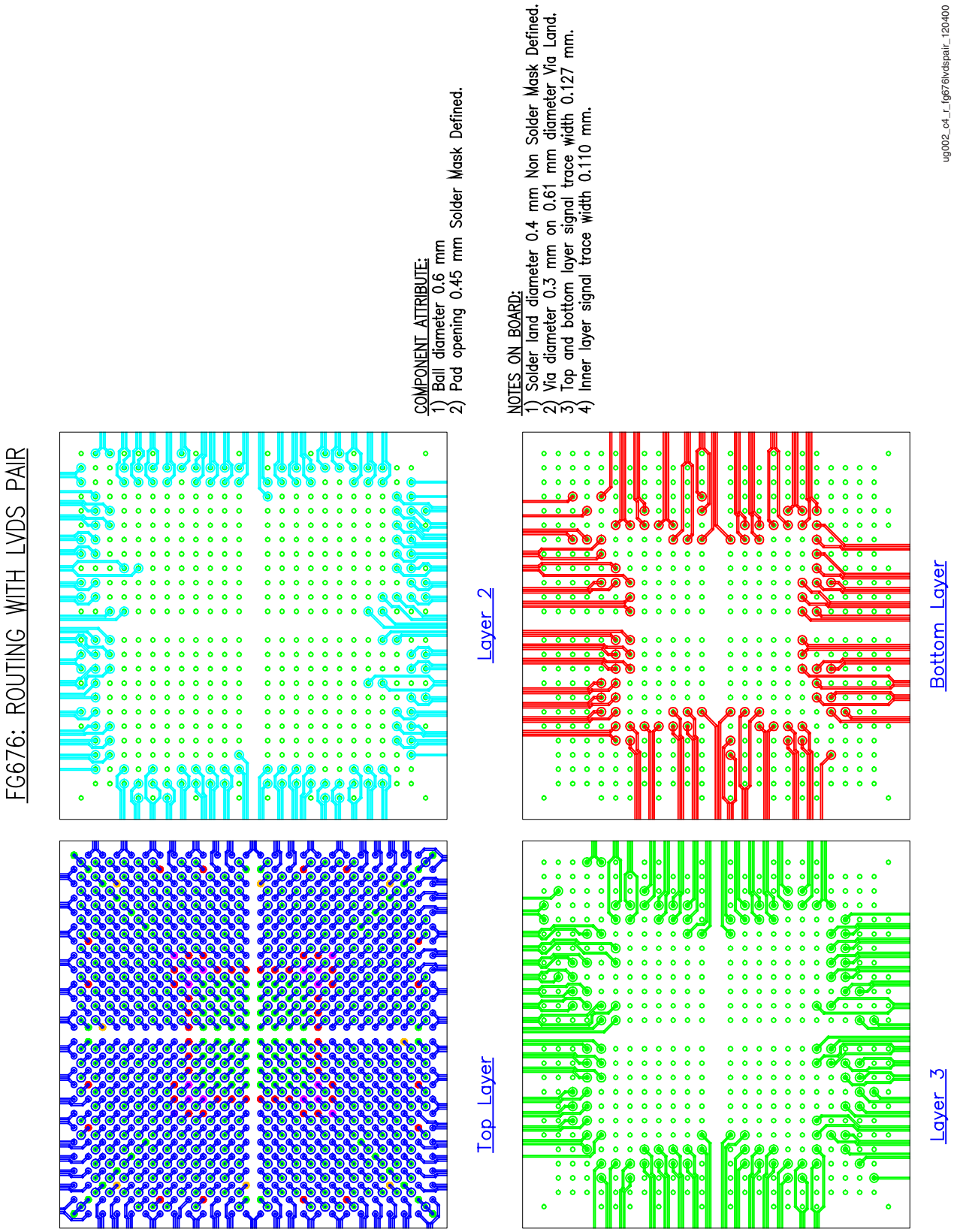


Figure 5-55: **FG676 Routing With LVDS Pairs**

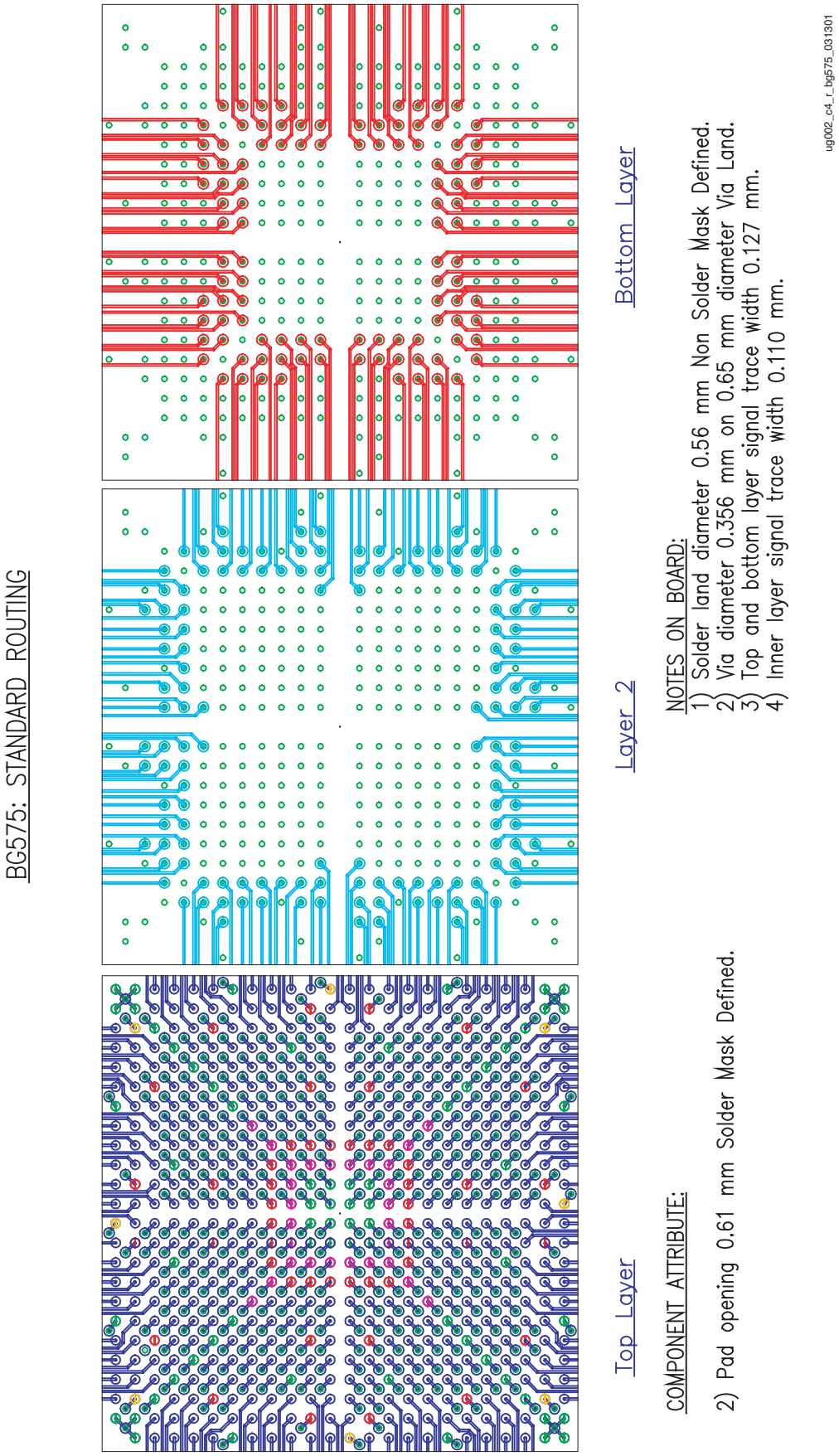


Figure 5-56: **BG575 Standard Routing**

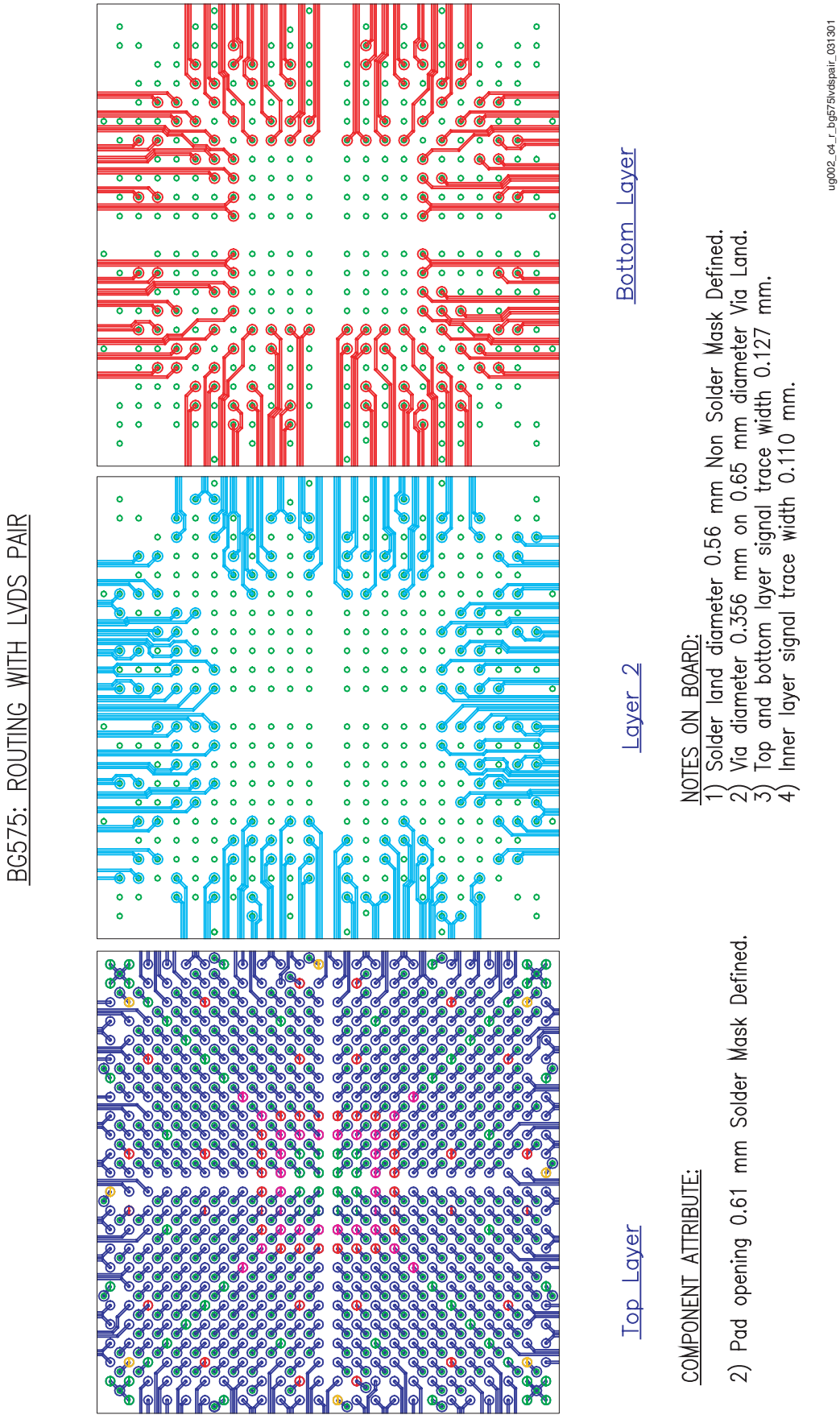


Figure 5-57: **BG575 Routing With LVDS Pairs**

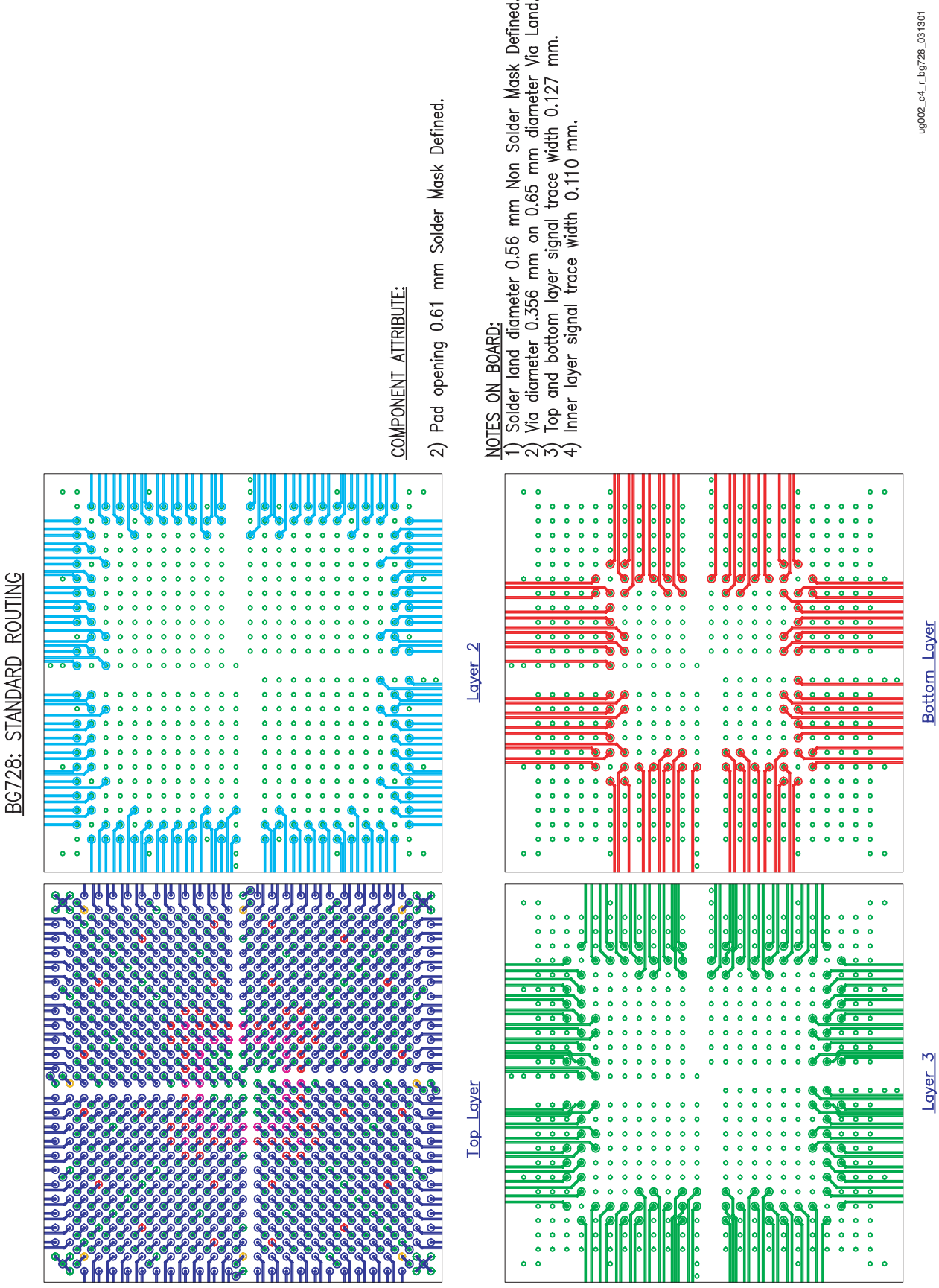


Figure 5-58: **BG728 Standard Routing**

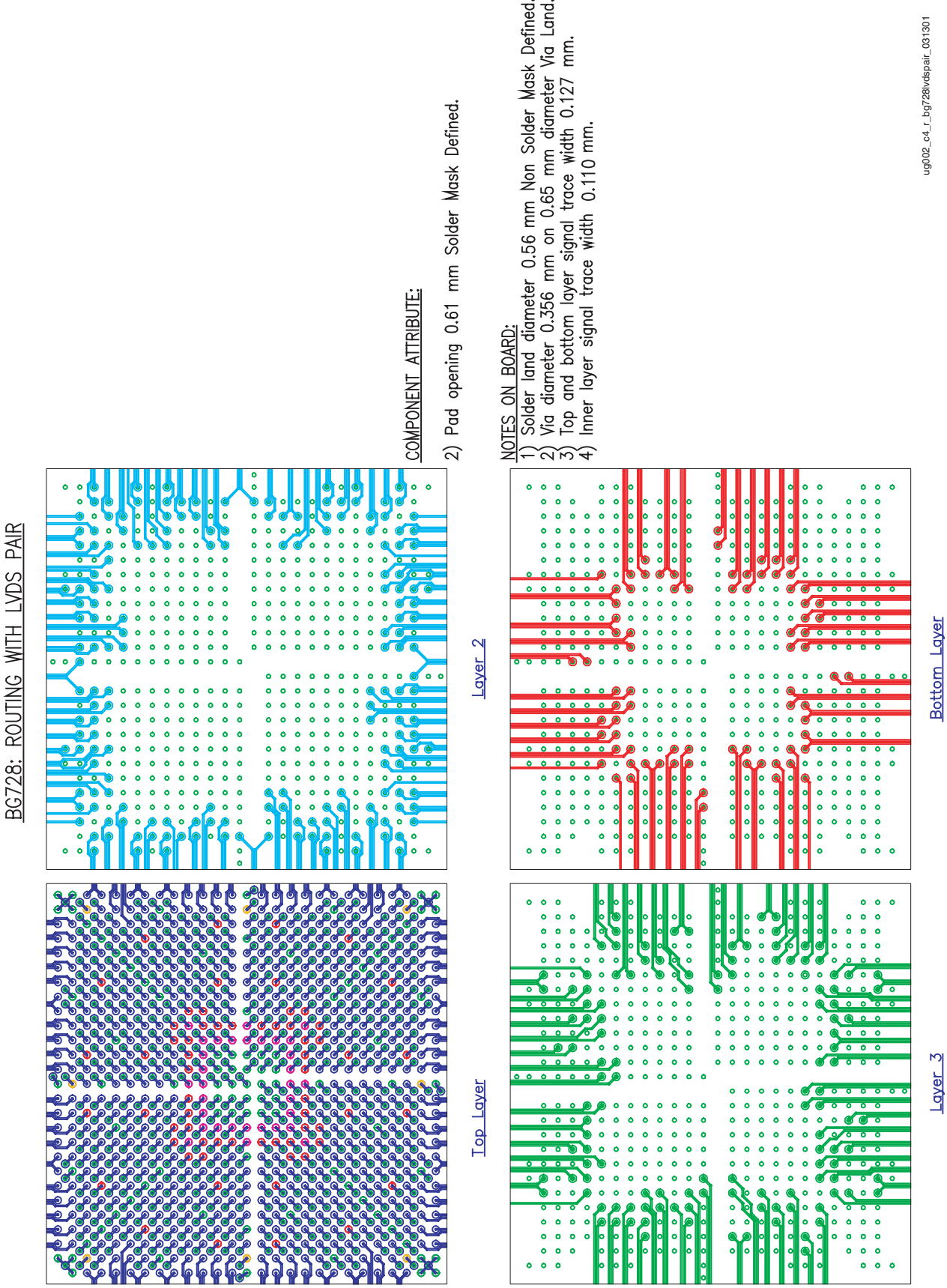


Figure 5-59: **BG728 Routing With LVDS Pairs**

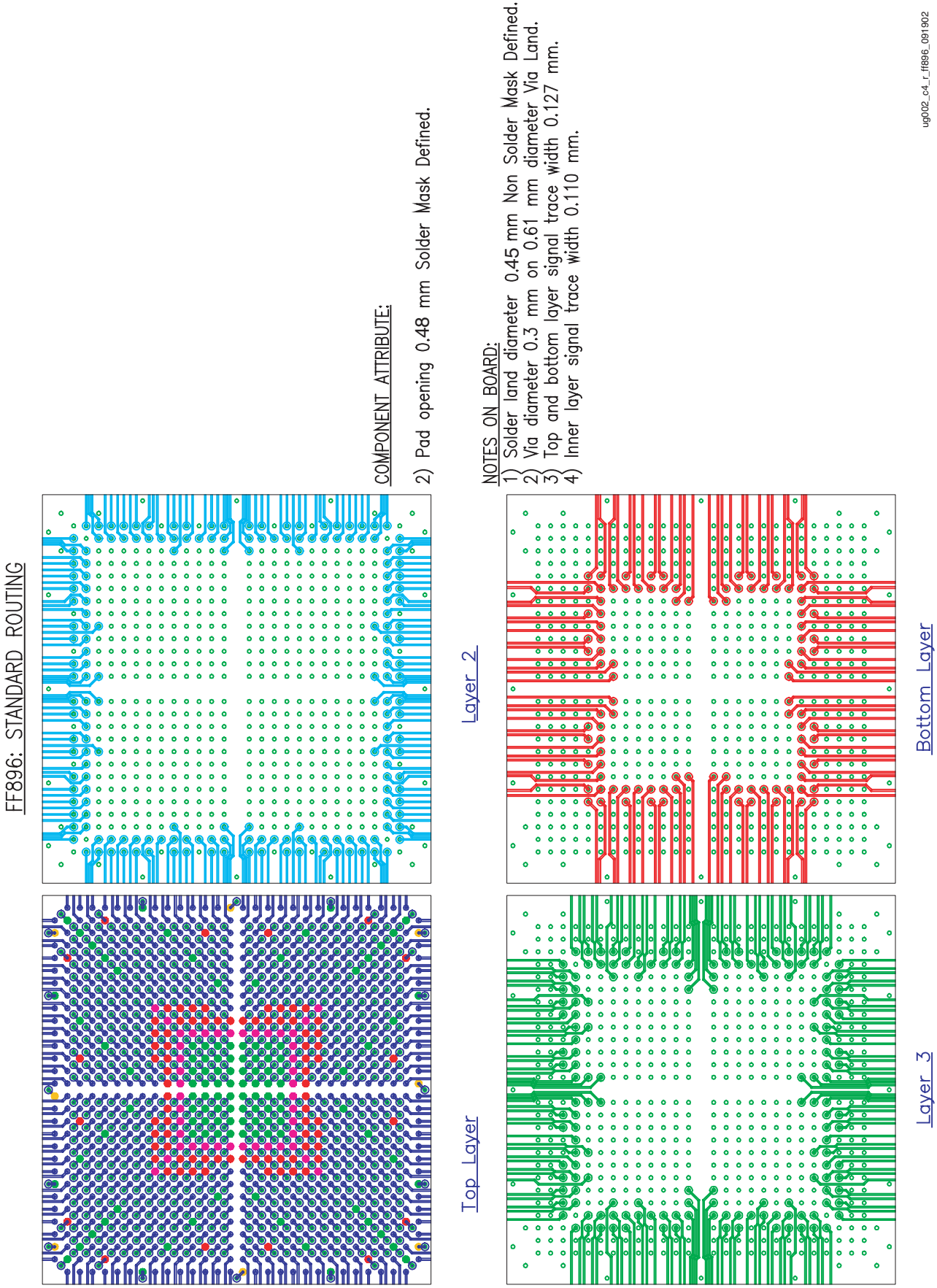


Figure 5-60: **FF896 Standard Routing**

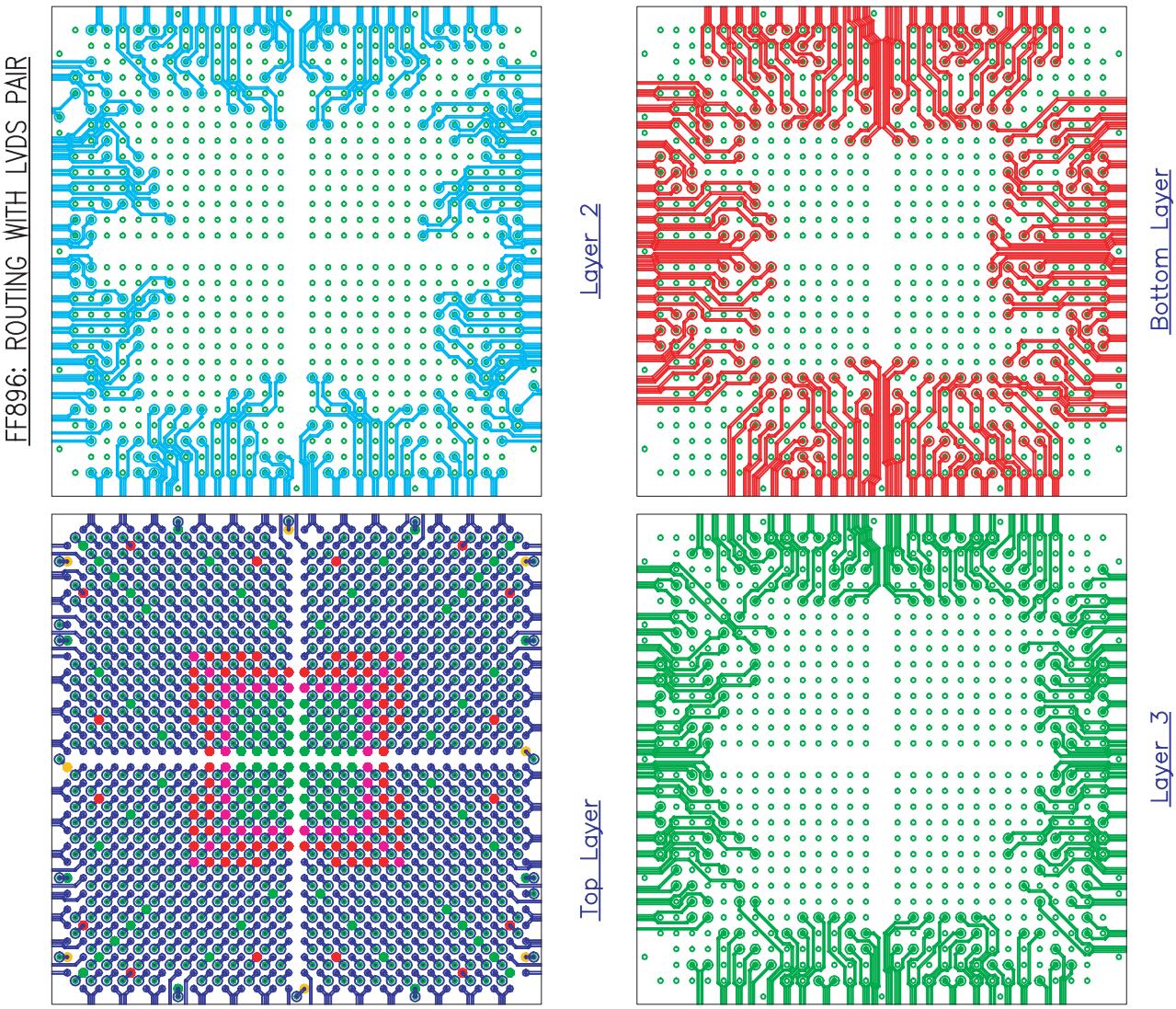


Figure 5-61: **FF896 Routing With LVDS Pairs**

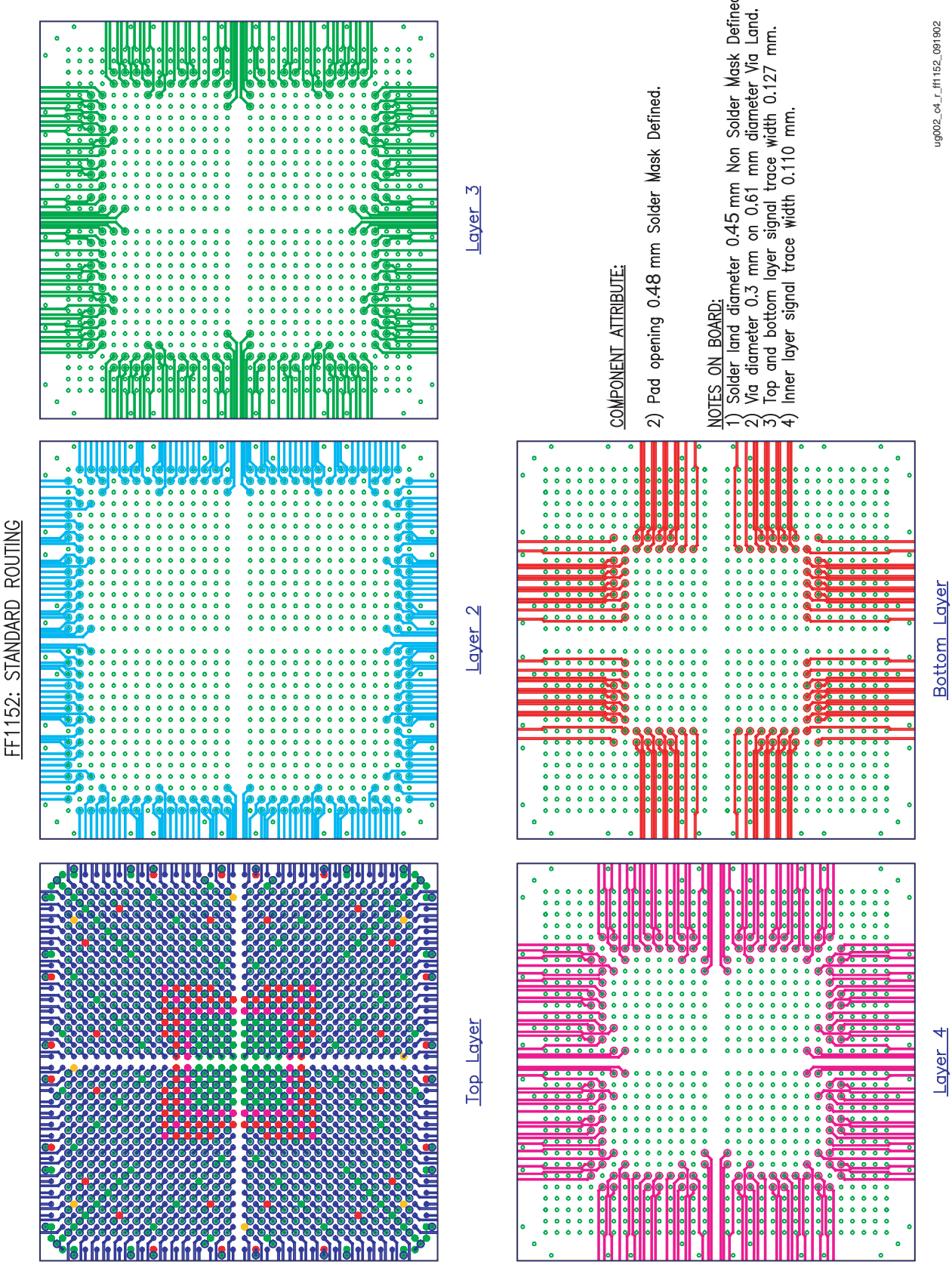


Figure 5-62: **FF1152 Standard Routing**

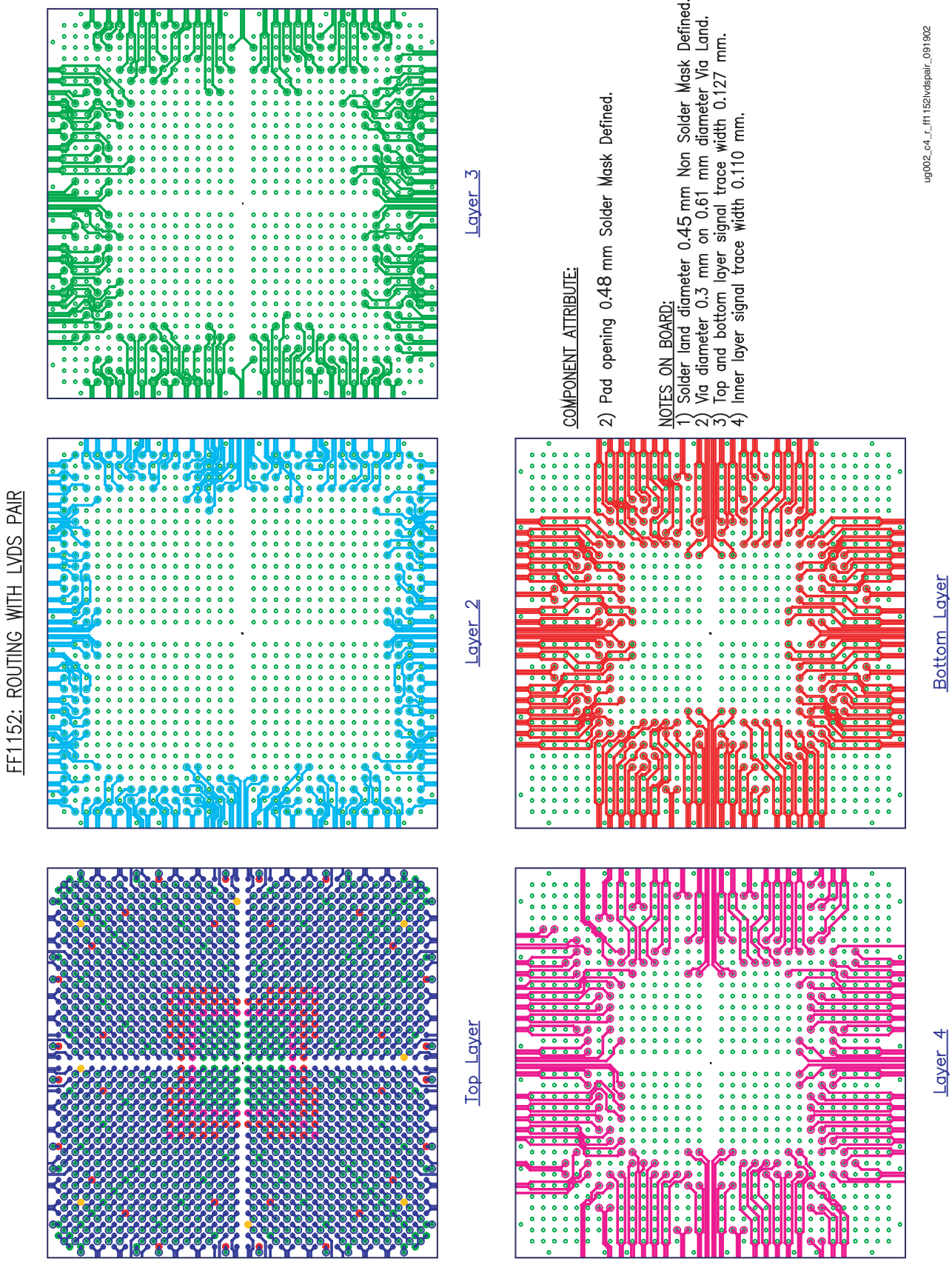


Figure 5-63: **FF1152 Routing With LVDS Pairs**

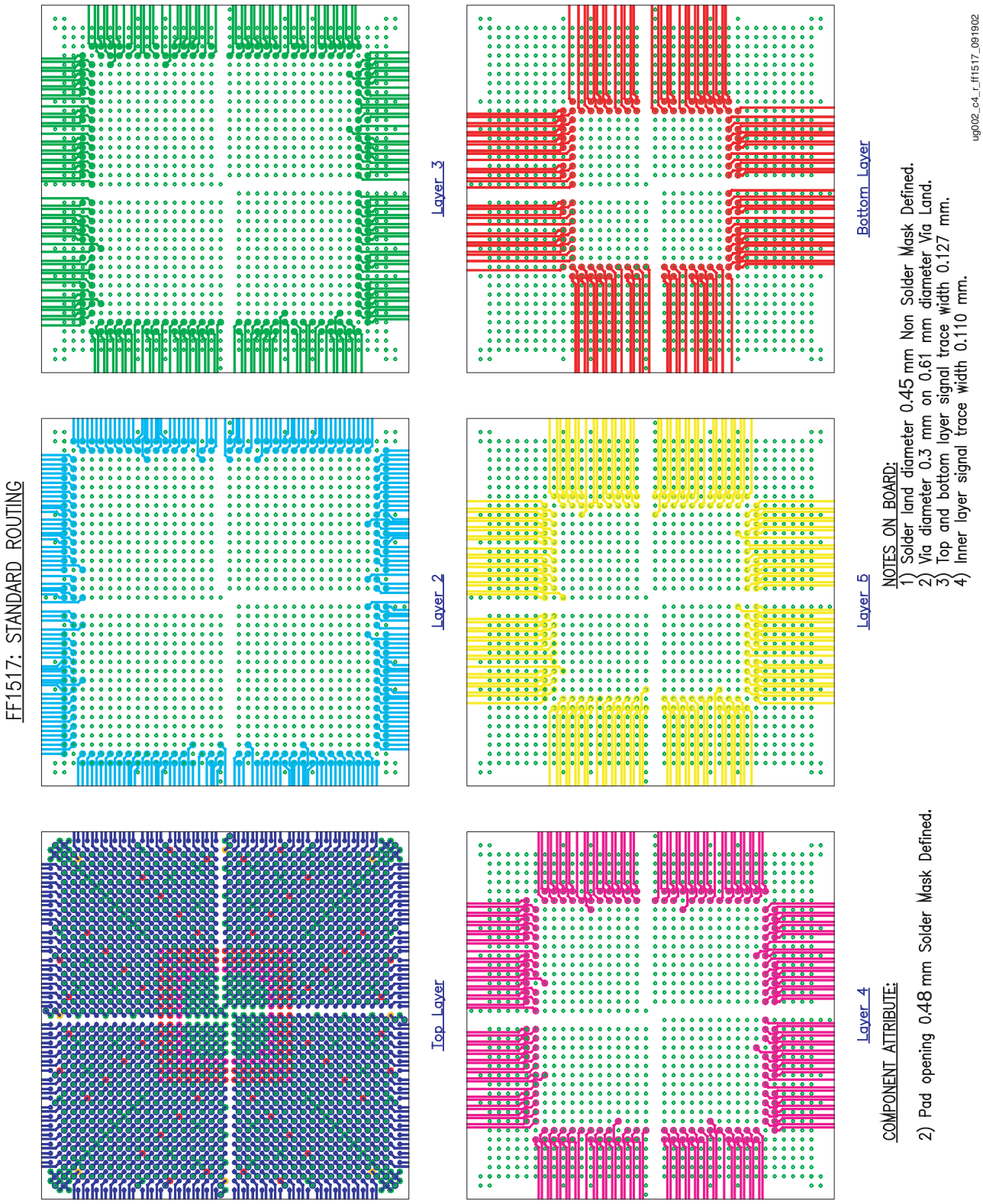


Figure 5-64: **FF1517 Standard Routing**

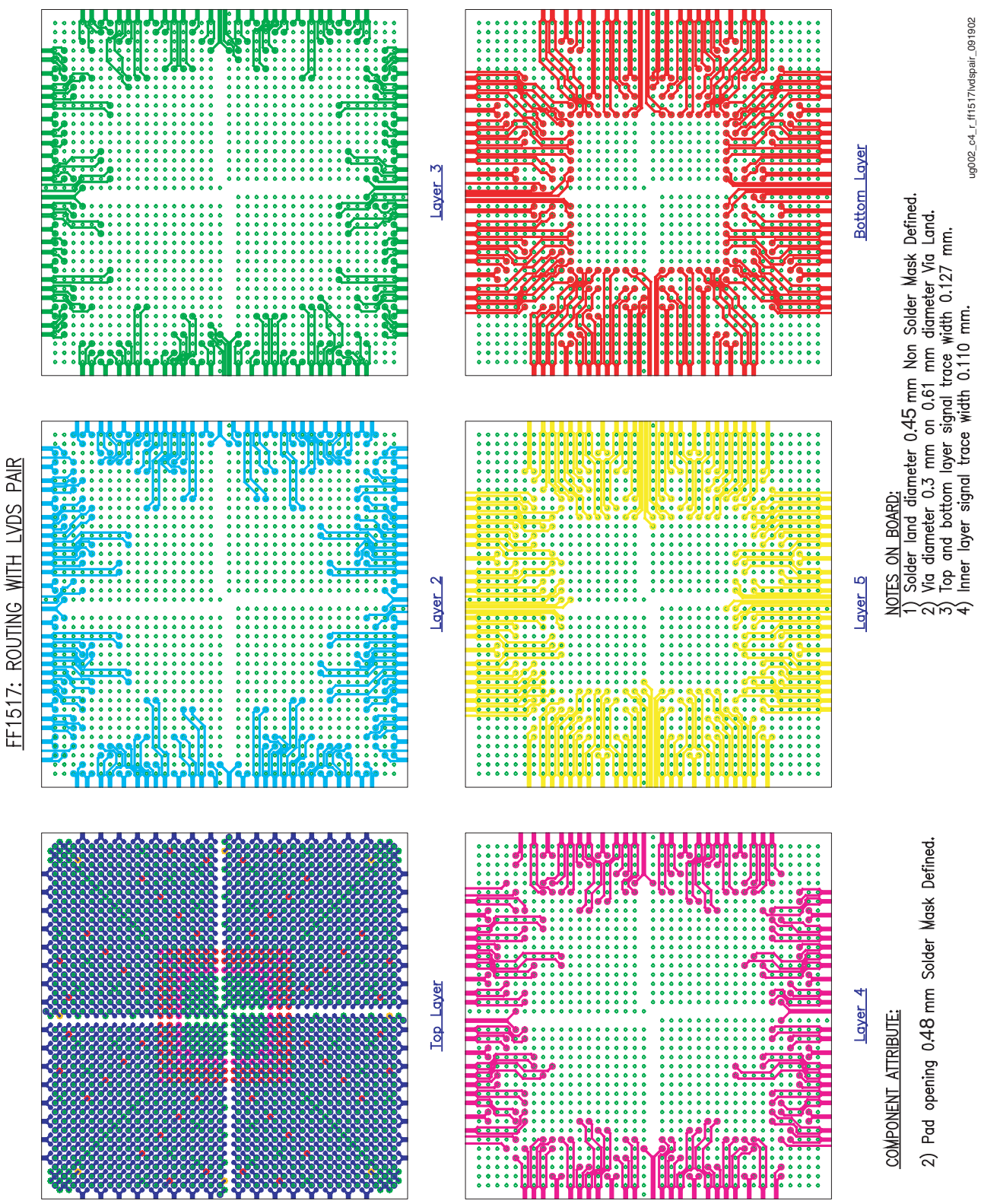


Figure 5-65: **FF1517 Routing With LVDS Pairs**

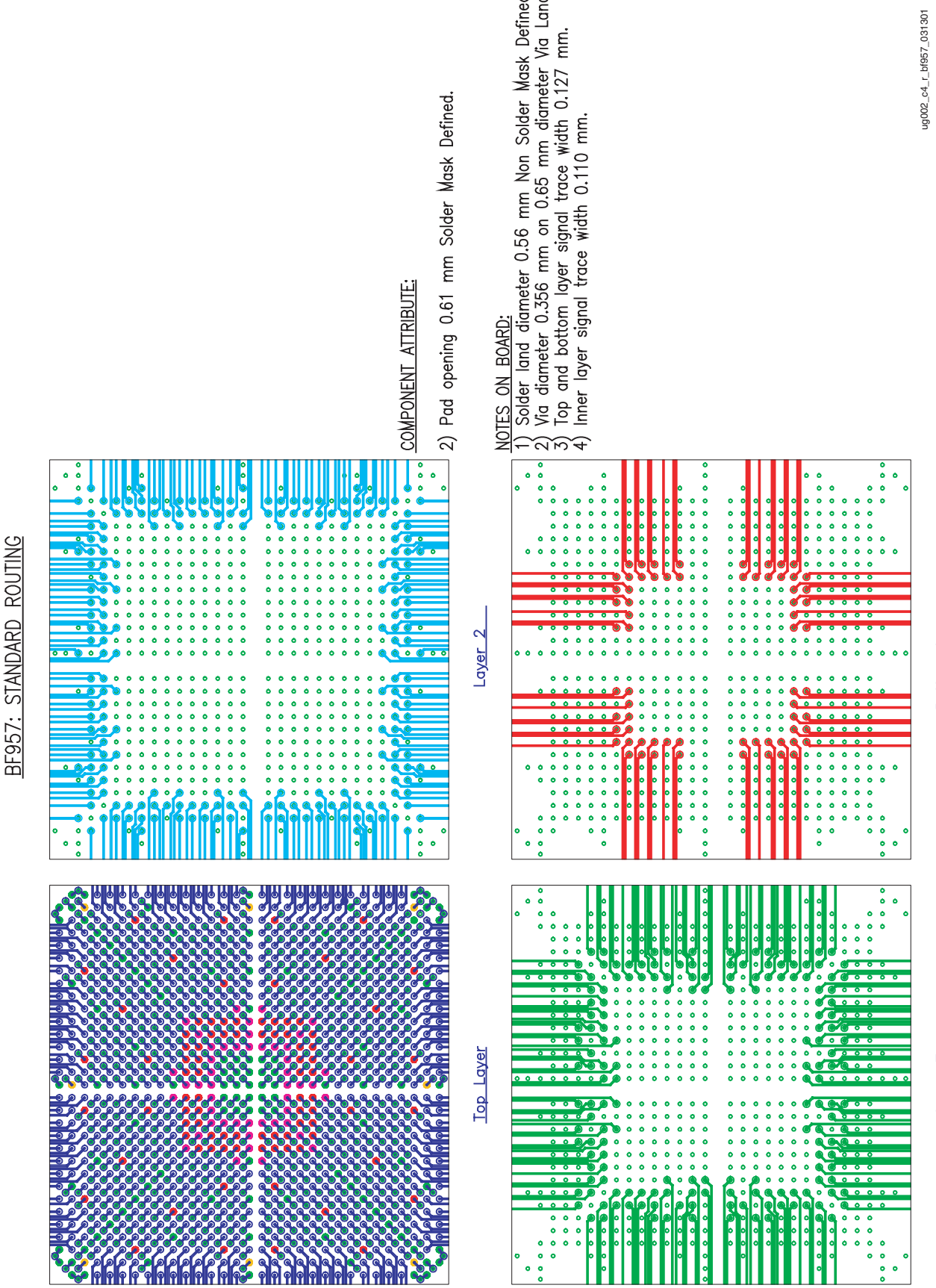


Figure 5-66: **BF957 Standard Routing**

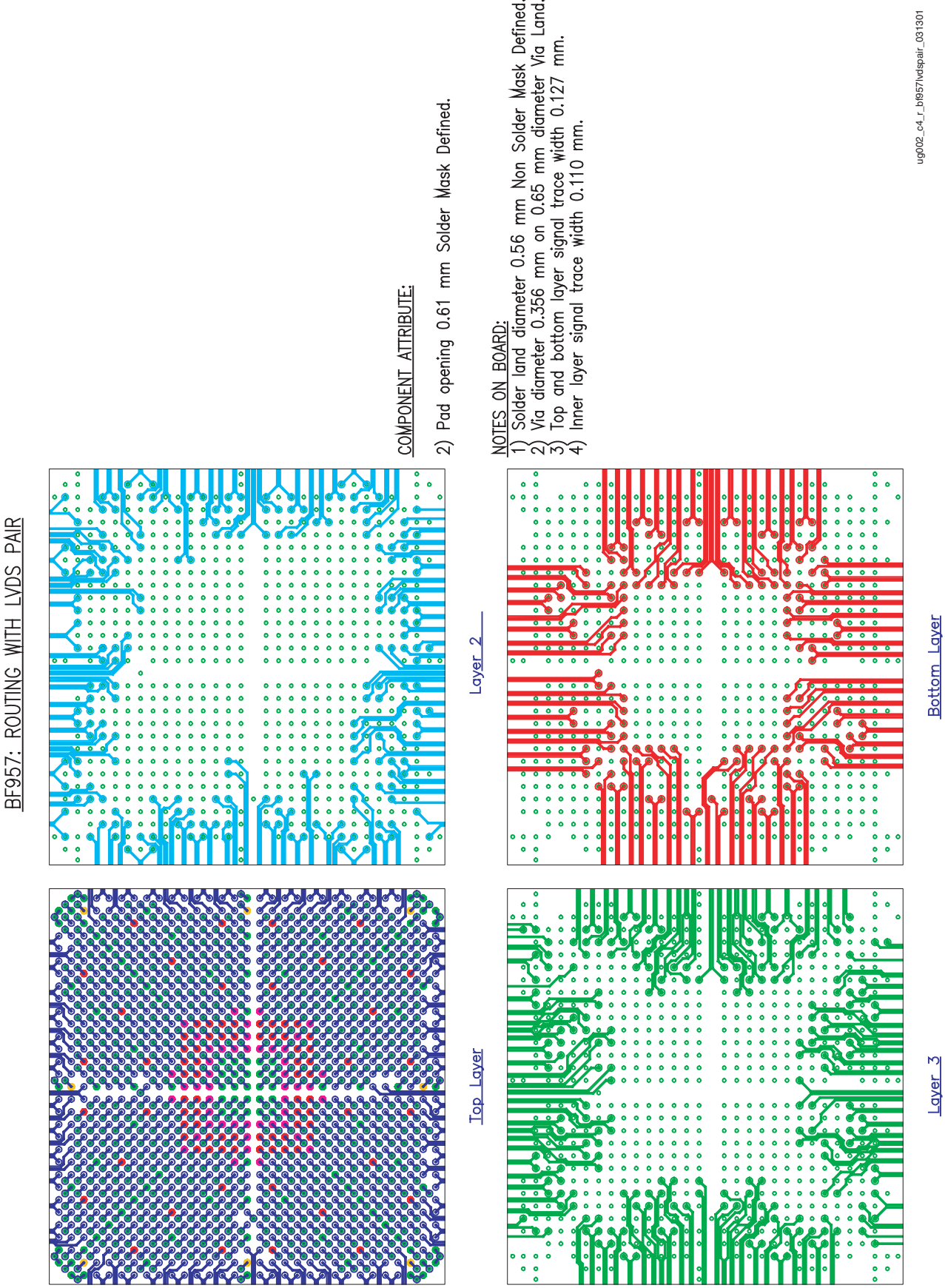


Figure 5-67: **BF957 Routing With LVDS Pairs**

Power Consumption

The Virtex-II power estimator worksheet estimates power consumption for a Virtex-II design before it is downloaded. It considers the design resource usage, toggle rates, I/O power, and many other factors in the estimation. The formulas used for calculations in the program are based on test design measurements.

Xilinx provides two versions of the power estimator, an Excel 97 version that works with Microsoft Office 97 software, and a CGI version for use with web browsers. They are identical in terms of estimations and data entries.

This section explains how to use the Power Estimator Worksheet to calculate estimated power consumption for Virtex-II designs. Since this is an estimation tool, results may not match precisely with what is measured on the board.

The power estimator consists of six categories: CLB (configurable logic block) logic power, dedicated non-multiplier power, dedicated registered multiplier power, block SelectRAM power, DCM (digital clock management), input/output power, and the results. To estimate power with the worksheet, a designer must determine how to group portions of the design into modules, what resources each module contains, the respective clock frequencies, and average toggle rates.

Note:

1. The Virtex-II power estimation is still under development. The table entries in this section may be different from the entries in the released version of the power estimation tool,

CLB Logic Power

Table 5-7 shows the data entries required for the CLB Logic Power section in the Power Estimator. This section estimates the power consumption of the CLBs for a Virtex-II design. In this section, users need to partition designs into modules, specify area utilization, and toggle rates.

Table 5-7: **CLB Logic Power**

Module	Frequency (MHz)	CLB Slices	Flip-Flops/ Latches	LUT		Average Toggle Rate (%)	Routing Amount
				Shift Register	SelectRAM		
User Module 1	0	0	0	0	0	0%	Medium
User Module 2	0	0	0	0	0	0%	Medium
User Module 3	0	0	0	0	0	0%	Medium
User Module 4	0	0	0	0	0	0%	Medium
User Module 5	0	0	0	0	0	0%	Medium
User Module 6	0	0	0	0	0	0%	Medium
User Module 7	0	0	0	0	0	0%	Medium
User Module 8	0	0	0	0	0	0%	Medium

Modules

Modules are portions of a design. A designer could treat the entire design as one module and calculate its toggle rate. However, estimating power this way is not as accurate as when the design is divided into multiple modules. Generally, with more modules the estimate is better.

The Virtex-II power estimator allows designs to be partitioned into a maximum of eight modules. Determining how to partition the design into modules depends on user preference. Three partitioning approaches are presented below as guidelines.

Grouping by Hierarchy

If a design contains hierarchical components at the top level, these components may be separated or grouped together to represent modules.

Grouping by Clocks

If a design has several different clocks, the logic associated with each clock should be treated as a module. For accuracy, it is recommended that each module contains only one clock.

Grouping by Functionality

For a design with sub-components that perform different functions, each sub-component can be considered as a module. For example, a microprocessor can be thought of as three main modules: an ALU, a Register File, and a Control System.

Frequency (MHz)

Frequency is the clock speed for the module. Again, it is strongly recommended that each module contains only one clock.

CLB Slices

This involves the total CLB usage of a module. This number is available from the synthesis report in a specific synthesis tool. For a more accurate result, MAP only this module in Xilinx Foundation software, and take the numbers from the map.mrp file. The map.mrp file is the output resource usage file produced by running the MAP program in the Xilinx Foundation software.

For schematic-based designs, obtaining this number is slightly more difficult. Designers can either estimate CLB usage based on the design structure or MAP the module and read the numbers from the map.mrp file.

Flip Flops or Latches

The total number of flip-flop and latch elements used for each module can be obtained from the synthesis report, the map.mrp file, or by adding up the registers from the schematics.

Shift Register LUTs

This is the total number of SRL16 elements used in each module.

SelectRAM LUTs

This is the total number of LUTs used as Distributed Select RAM components. For Virtex-II devices, one 16 x 1 synchronous RAM is equivalent to one LUT, and one 16 x 1 dual-port RAM is equivalent to two LUTs (split between two slices).

Average Toggle Rate (%)

The toggle rate describes how often the output changes with respect to the input clock, usually between 6% and 12% for a typical module. Functional simulation is required to accurately calculate the toggle rate. Designers need to simulate all the flip-flop outputs in each module with regard to the clock, and calculate how often the flip-flop outputs change in relation to the clock.

Measuring the toggle rate becomes a more complex and a time-consuming process as module size increases. A toggle flip-flop has a 100% toggle rate, an 8-bit counter has 28%, and 16-bit counter has 14%.

Figure 5-68 is an example of how to calculate the toggle rate for a 4-bit counter.

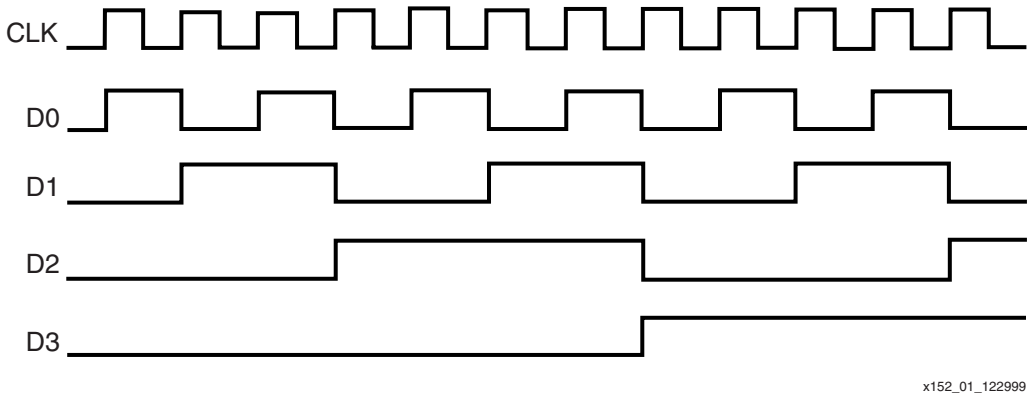


Figure 5-68: **Output Waveform of a 4-bit Counter**

Figure 5-68 shows the simulation wave form of a 4-bit counter. D0 stands for the LSB of the count, and D3 stands for the MSB. The toggle rate of D0 is 100% because D0 changes after every clock cycle. The toggle rate of D1 is 50% because D1 changes after every two clock cycles. The toggle rate of D2 is 25% because D2 changes after every four clock cycles. The toggle rate of D3 is 12.5% because D3 changes after every eight clock cycles. In this example, the average toggle rate of a 4-bit counter derived in the following equation is 46.875%.

$$\frac{(100 + 50 + 25 + 12.5)}{4} = 46.875$$

Routing Amount

There are three levels concerning the amount of routing to be used: low, medium, and high. The routing level is determined by the primary logic type of the module. Typical data path logic typically requires a low routing usage, random logic calls for a medium level, and control logic needs a high level.

Each designer needs to determine the routing that is most appropriate for each module.

Routing, which is determined by the type of logic in the module, is divided into three levels: low, medium, and high. Each designer needs to determine the routing that is most appropriate for each module.

1. Typical data path logic, which uses combinatorial logic such as multiplexers, adders, AND gates, and OR gates, usually requires a low routing usage. This also applies to any other signals that have one or two fanouts between structures.
2. Random logic, such as decoders, encoders, or any logic that has three to five fanouts, calls for a medium level of routing usage.
3. Control logic is typically logic with high fanout signals (excluding clocks) such as clock enables or reset signals. Control logic used in state machines also belongs to this category.

Block SelectRAM Power

Table 5-8 shows the data entries required for the Block SelectRAM Power section. This section is used to specify how many block RAMs are used and to determine their estimated power consumption. Before doing the calculation, designers can either treat all the RAMB16 cells as one module or break them down into smaller modules. RAMB16 is the base name for the Virtex-II Block SelectRAM component.

RAMB16 Cells

This is total number of Block Select RAMs (RAMB16 cells) used in each module.

Port A Frequency (MHz)

This is the frequency on the CLKA pin.

Port A Width

This is data width of DIA and DOA busses.

Port A Enable Rate (%)

This specifies how often ENA is enabled with respect to the clock. For a typical design, the rate may be 100% because the enable could be enabled all the time. For a FIFO design, the rate could be approximately 50% due to bursting of data into and out of the RAM.

Port B Frequency (MHz)

This is the frequency on the CLKB pin.

Port B Width

This is the data width of DIB and DOB busses.

Port B Enable Rate (%)

This specifies how often ENB is enabled with respect to the clock.

Table 5-8: **Block SelectRAM Power**

Module	RAMB16 Cells	Port A			Port B		
		Frequency (Mhz)	Width	Enable Rate (%)	Frequency (MHz)	Width	Enable Rate (%)
User Module 1	0	0	0	0	0%	0	0%
User Module 2	0	0	0	0	0%	0	0%
User Module 3	0	0	0	0	0%	0	0%
User Module 4	0	0	0	0	0%	0	0%
User Module 5	0	0	0	0	0%	0	0%
User Module 6	0	0	0	0	0%	0	0%
User Module 7	0	0	0	0	0%	0	0%
User Module 8	0	0	0	0	0%	0	0%

Digital Clock Management Power

Table 5-9 shows the data entries required for the DCM Power section and is used to estimate how much power DCMs consume. Only the clock input frequencies to the CLKIN pin needs to be entered.

Table 5-9: **Clock Delay Locked Loop Power**

Module	Clock Input Frequency (MHz)
User DCM 1	0
User DCM 2	0
User DCM 3	0
User DCM 4	0
User DCM 5	0
User DCM 6	0
User DCM 7	0
User DCM 8	0
User DCM 9	0
User DCM 10	0
User DCM 11	0
User DCM 12	0

Non-Registered Multiplier Power

The data entries for the Non-Registered Multiplier Power section are shown in Table 5-10. These entries are used to estimate Non-Registered Multiplier power consumption.

Table 5-10: **Data Entries for Non-Registered Multiplier Power**

Module	Mult18x18 Cell	Port A Width	Port B Width
User Module 1	0	0	0
User Module 2	0	0	0
User Module 3	0	0	0
User Module 4	0	0	0
User Module 5	0	0	0
User Module 6	0	0	0
User Module 7	0	0	0
User Module 8	0	0	0

Multi18x18 Cell

Multi18x18 cell is the total number of Multipliers used in each module.

Port A Width

Port A width is the data width of A busses.

Port B Width

Port B width is the data width of B busses.

Registered Multiplier Power

Data entries for the Registered Multiplier Power section are shown in Table 5-11. They are used to estimate Registered Multiplier power consumption.

Frequency

This is the frequency that the Multipliers operate at.

Multi18x18 Cell

Multi18x18 cell is the total number of Multipliers used in each module.

Port A Width

Port A width is the data width of A busses.

Port B Width

Port B width is the data width of B busses.

Average Toggle Rate

This is the toggle rate for the multiplier modules. This number can be obtained in the same way as obtaining the Average Toggle Rate in the CLB logic power section.

Table 5-11: **Data Entries for Registered Multiplier Power**

Module	Frequency (MHz)	Mult18x18 Cell	Port A Width	Port B Width	Average Toggle Rate
User Module 1	0	0	0	0	0
User Module 2	0	0	0	0	0
User Module 3	0	0	0	0	0
User Module 4	0	0	0	0	0
User Module 5	0	0	0	0	0
User Module 6	0	0	0	0	0
User Module 7	0	0	0	0	0
User Module 8	0	0	0	0	0

Input/Output Power

Table 5-12 shows the data entries for the Input/Output Power section used to estimate the power dissipation of the Inputs and Outputs. I/Os should be grouped into modules based on their I/O standard type. If the entire design has only one I/O standard type, all of the I/Os can be treated as one module. However, separating the I/Os into smaller modules makes it easier to obtain more accurate results.

Frequency (MHz)

This is the frequency of the module.

I/O Standard Type

This is the type of I/Os used in the module. Each module can have only one I/O standard type. I/O power is strongly influenced by the I/O standard used.

Inputs

This is the total number of the input buffers in each module.

Outputs

This is the total number of the output buffers in each module.

Average Output Toggle Rate (%)

This number can be obtained in the same way as obtaining the Average Toggle rate in the CLB Logic Power section.

Average Output Load (pF)

This specifies the average capacitive load on the outputs.

Table 5-12: **Data Entries for Input/Output Power**

Module	Frequency (MHz)	I/O Standard Type	Inputs	Outputs	Average Output Toggle Rate (%)	Average Output Load (pF)
User Module 1	0	LVTTL_12	0	0	0%	0
User Module 2	0	LVTTL_12	0	0	0%	0
User Module 3	0	LVTTL_12	0	0	0%	0
User Module 4	0	LVTTL_12	0	0	0%	0
User Module 5	0	LVTTL_12	0	0	0%	0
User Module 6	0	LVTTL_12	0	0	0%	0
User Module 7	0	LVTTL_12	0	0	0%	0
User Module 8	0	LVTTL_12	0	0	0%	0

Results

The results section of the power estimator are shown in Table 5-13. The four sections of the power estimator program independently estimate power consumption, and the results are displayed at the end of each section.

The total design power consumption is the summation of those, and is displayed at the very top of the program.

Table 5-13: **Power Estimator Results**

Target		Estimated Design Power Values (mW)					
Device	Package	Total Power	V_{CCINT} 1.5 V	V_{CCO} 3.3 V	V_{CCO} 2.5 V	V_{CCO} 1.5 V	Output Sink Power
XC2V500	FG256	0	0	0	0	0	0

Target Device

This refers to the target Virtex-II device size.

Note: No checking is done to verify that the module entries fit into the amount of resources available in the selected devices.

Target Package

This refers to the package of the device.

Note: No checking is done to verify that the selected device-package combination is valid.

Estimated Total Power

This section displays the total power consumption of the design. It is the summation of CLB Logic power, Block Select RAM power, Multiplier power, DCM power, and Input/Output power.

Estimated V_{CCINT} 1.5V Power

This section displays the total power consumption from the core supply voltage (V_{CCINT}). It does not include the power consumption from the input and output source voltage (V_{CCO}).

Estimated V_{CCAUX} 3.3V Power

This section displays the power consumption from auxiliary circuits.

Estimated V_{CCO} 3.3V Power

This section displays the V_{CCO} power consumption of 3.3 V applications. The I/O standards that use 3.3V V_{CCO} are LVTTL, LVCMOS33 PCI, SSTL3 Class I and II, and AGP2X.

Estimated V_{CCO} 2.5V Power

This section displays the V_{CCO} power consumption of 2.5 V applications. The supported I/O standards are LVCMOS25 and SSTL2 Class I and II.

Estimated V_{CCO} 1.5V Power

This section displays the V_{CCO} power consumption of 1.5 V applications. The supported I/O standards are LVCMOS15, and HSTL Class I, II, III, and IV.

Estimated Output Sink Power

This section displays the power consumption when sinking current to ground. The supported I/O standards are GTL and GTL+.

IBIS Models

The need for higher system performance leads to faster output transitions. Signals with fast transitions cannot be considered purely digital; it is important to understand their analog behavior for signal integrity analysis.

To simulate the signal integrity on printed circuit boards (PCB) accurately and solve design problems before the PCB is fabricated, models of the I/O characteristics are required. SPICE models are most frequently used for this purpose. A manufacturer's SPICE models, however, contain proprietary circuit-level information. Therefore, simpler models are devised to extract SPICE parameters for the proprietary information to remain protected. One such standard is the I/O Buffer Information Specification (IBIS) format originally suggested by Intel.

In the early 1990's, the IBIS Open Forum was formed and the first IBIS specification was written to promote tool independent I/O models for system signal integrity analysis.

IBIS is now the ANSI/EIA-656 and IEC 62014-1 standard. IBIS accurately describes the signal behavior of the interconnections without disclosing the actual technology and circuitry used to implement the I/O. The standard is basically a black-box approach to protecting proprietary information.

Using IBIS Models

IBIS models are used by designers for system-level analysis of signal integrity issues, such as the evaluation and matching of loads to drivers for ringing and ground bounce, examining effects of cross talk, and predicting RFI/EMI. It is useful in that complete designs can be simulated and evaluated before additional costs are incurred for PCB fabrication and assembly time.

IBIS models consist of look-up tables that predict the I/V characteristics and dV/dt of integrated circuit inputs and outputs when combined with the PCB wiring. The predictions are performed for the typical case, minimum case (weak transistors, low V_{CC}, hot temperatures), and maximum case (strong transistors, high V_{CC}, cold temperatures).

IBIS models have limitations in that they do not contain internal delay modeling and are limited in package modeling. IBIS models contain package parasitic information for simulation of ground bounce. Although the data is available within the model file, not all simulators are able to use the data to simulate ground bounce. Simulation results may not agree with the actual results due to package, die, and PCB ground plane modeling problems. Similarly, because simultaneous switching outputs (SSOs) are also difficult to model, only a first approximation is provided to the designer.

IBIS Generation

IBIS is generated either from SPICE simulations, or actual measurements of final devices. IBIS models that are derived from measurements do not have process corner information, unlike IBIS models that are derived from SPICE simulations. The measurements are of only a few parts, and the extremes of production are not represented by such a method.

SPICE is a transistor model based on detailed equations using device geometry, and properties of materials. A SPICE netlist of the CMOS buffer is required for V/I and dV/dt curve simulations. These SPICE simulations are then converted to IBIS format/syntax.

Advantages of IBIS

SPICE requires a greater knowledge of the internal workings of the circuits being modeled, and as such, errors may be made in simulation indicating a problem when there is none. IBIS models are easy to use, and because many of the decisions required for simulation parameters have been organized. IBIS simulations are faster compared to SPICE simulations, because IBIS does not contain circuit details. The voltage/current/time information provided in the IBIS model is only for the external nodes of the building block, making IBIS ideal for system-level interconnects design. Although IBIS models are not as accurate as SPICE models, they are entirely adequate for system-level analysis.

IBIS File Structure

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 contents 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
- V/I Tables for:
  - Pullup
  - Pulldown
  - Power Clamp
  - Ground Clamp
- Rise and Fall dV/dt for minimum, typical, and maximum conditions (driving 50 ohms)
- Package Model (optional) XXXX.pkg with RLC sections.

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 ($2 \times V_{CC}$). 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 ($2 \times V_{CC}$).

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 0 V 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} or V_{DD}).
- The ground clamp I/V curve contains receive (3-state) mode data, with the origin of the curve at 0 V for CMOS buffers.

- The power clamp I/V curve contains receive (3-state) mode data, with the origin of the curve at the supply voltage (V_{CC} or V_{DD}). For 3.3 V buffers that are 5 V tolerant, the power clamp is referenced to 5 V while the pullup is referenced to 3.3 V.

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 is 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.

IBIS Simulations

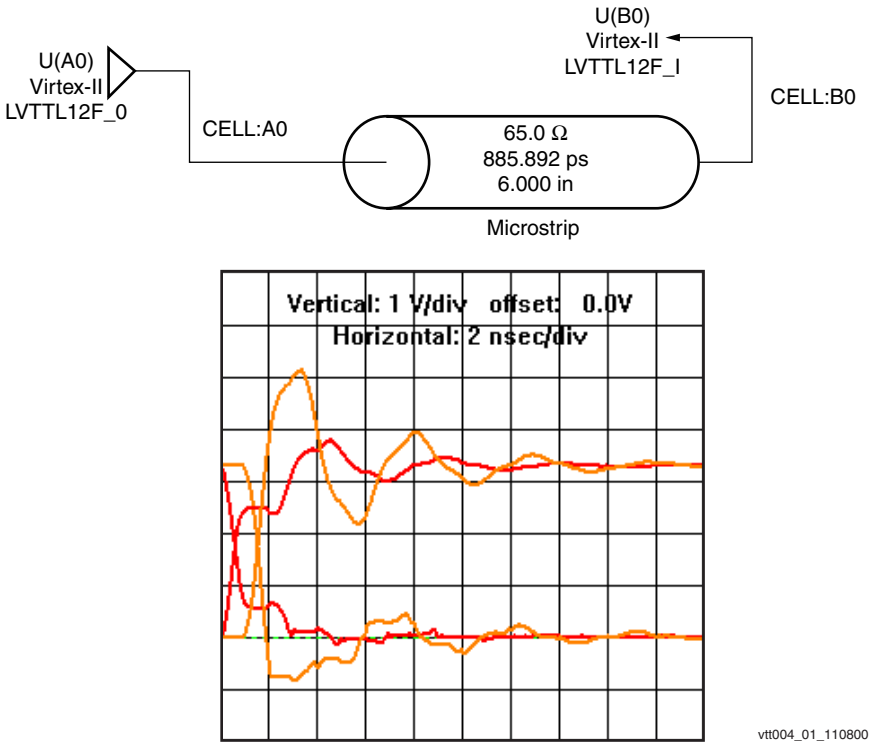


Figure 5-69: **Unterminated Example**

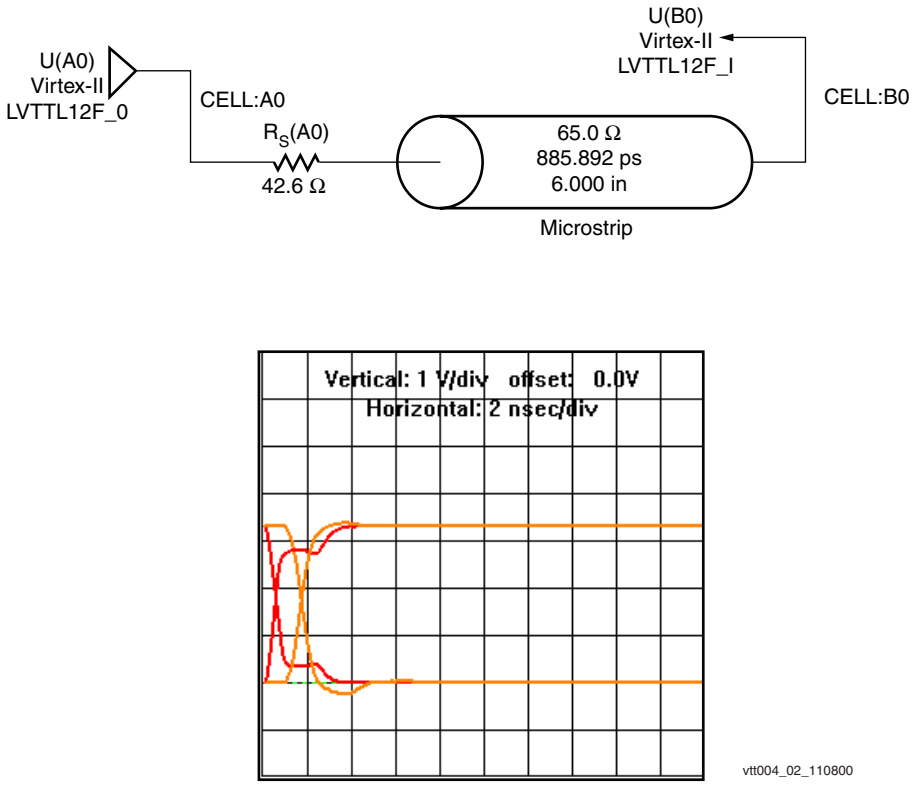


Figure 5-70: **Series Termination Example**

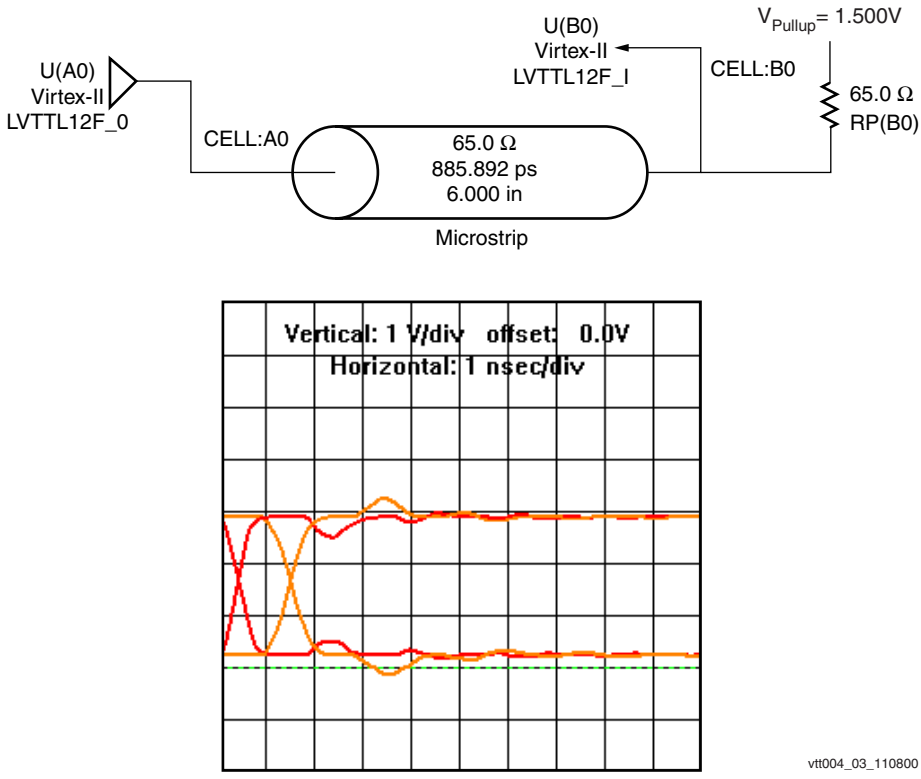


Figure 5-71: **Parallel Termination Example**

IBIS Simulators

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 the following vendors:

- Cadence
- Avanti Corporation
- Hyperlynx
- Mentor
- Microsim
- Intusoft
- Veribest
- Viewlogic

Xilinx IBIS Advantages

Xilinx provides preliminary IBIS files before working silicon has been verified (before tape out), as well as updated versions of IBIS files after the ICs are verified. Preliminary IBIS files are generated from SPICE models before working silicon has been verified. After the IC (device) is verified, appropriate changes are made to the existing IBIS files. These IBIS files are available at the following web site:

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

IBIS Reference Web Site

http://www.eia.org/eig/ibis/ibis.htm

BSDL and Boundary Scan Models

Boundary scan is a technique that is used to improve the testability of ICs. With Virtex-II devices, registers are placed on I/Os that are connected together as a long shift register. Each register can be used to either save or force the state of the I/O. There are additional registers for accessing test modes.

The most common application for boundary scan is testing for continuity of the IC to the board. Some packages make visual inspection of solder joints impossible, e.g. BGA. The large number of I/Os available requires the use of such packages, and also increases the importance of testing. A large number of I/Os also means a long scan chain.

Test software is available to support testing with boundary scan. The software requires a description of the boundary scan implementation of the IC. The IEEE 1149.1 specification provides a language description for Boundary Scan Description Language (BSDL). Boundary scan test software accepts BSDL descriptions.

The IEEE 1149.1 spec also defines a 4 to 5 pin interface known as the JTAG interface. IEEE 1532 is a capability extension of IEEE 1149.1.

BSDL Files

Preliminary BSDL files are provided from the IC Design Process. Final BSDL files have been verified by an external third party test and verification vendor. The following are Virtex-II BDSL file names.

Virtex-II BSDL File Names	
XC2V40_CS144.BSD	XC2V2000_FG676.BSD
XC2V40_FG256.BSD	XC2V2000_FF896.BSD
XC2V80_CS144.BSD	XC2V2000_BG575.BSD
XC2V80_FG256.BSD	XC2V2000_BF957.BSD
XC2V250_CS144.BSD	XC2V3000_FG676.BSD
XC2V250_FG256.BSD	XC2V3000_FF1152.BSD
XC2V250_FG456.BSD	XC2V3000_BG728.BSD
XC2V500_FG256.BSD	XC2V3000_BF957.BSD
XC2V500_FG456.BSD	XC2V4000_FF1152.BSD
XC2V1000_FG256.BSD	XC2V4000_FF1517.BSD
XC2V1000_FG456.BSD	XC2V4000_BF957.BSD
XC2V1000_FF896.BSD	XC2V6000_FF1152.BSD
XC2V1000_BG575.BSD	XC2V6000_FF1517.BSD
XC2V1500_FG676.BSD	XC2V6000 _BF957.BSD
XC2V1500_FF896.BSD	XC2V8000_FF1152.BSD
XC2V1500_BG575.BSD	XC2V8000_FF1517.BSD

Appendix A

BitGen and PROMGen Switches and Options

Using BitGen

BitGen produces a bitstream for Xilinx device configuration. After the design has been completely routed, it is necessary to configure the device so that it can execute the desired function. The Xilinx bitstream necessary to configure the device is generated with BitGen. BitGen takes a fully routed NCD (Circuit Description) file as its 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 defining 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 can then be downloaded into the FPGA memory cells, or it can be used to create a PROM file (see Figure A-1).

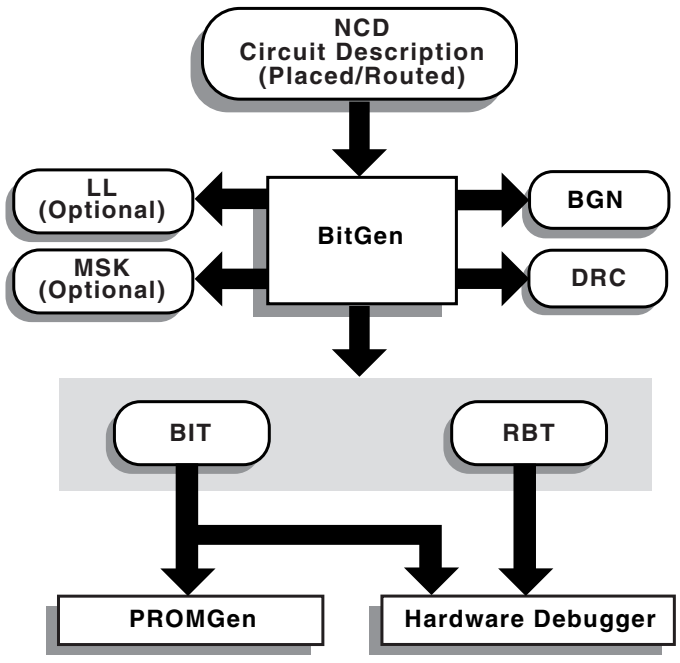


Figure A-1: **BitGen**

BitGen Syntax

The following syntax creates a bitstream from your NCD file.

```
bitgen [options] infile[.ncd] [outfile] [pcf_file]
```

options is one or more of the options listed in the "BitGen Options" on page 467.

Infile is the name of the NCD design for which you want to create the bitstream. You can specify only one design file, and it must be the first file specified on the command line.

You do not have to use an extension. If you do not, **.ncd** is assumed. If you do use an extension, it must be **.ncd**.

Outfile is the name of the output file. If you do not specify an output file name, BitGen creates one in the same directory as the input file. If you specify -l on the command line, the extension is .ll (see -l command line option). If you specify -m (see -m command line option), the extension is .msk. If you specify -b, the extension is .rbt. Otherwise the extension is .bit. If you do not specify an extension, BitGen appends one according to the aforementioned rules. If you do include an extension, it must also conform to the rules.

Pcf_file is the name of a physical constraints (PCF) file. BitGen uses this file to determine which nets in the design are critical for tiedown, which is not available for Virtex families. BitGen automatically reads the .pcf file by default. If the physical constraints file 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 read if that file exists.

A report file containing all BitGen's output is automatically created under the same directory as the output file. The report file has the same root name as the output file with a .bgn extension.

BitGen Files

This section describes input files that BitGen requires and output files that BitGen generates.

Input Files

Input to BitGen consists of 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. If you specify a PCF file on the BitGen command line, BitGen uses this file to determine which nets in the design are critical for tiedown (not used for Virtex families).

Output Files

Output from BitGen consists of the following files.

- BIT file—a binary file with a .bit extension. The BIT file contains all of the configuration information from the NCD file defining 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 can then be downloaded into the FPGA memory cells, or it can be used to create a PROM file (see "Using PROMGen" on page 473).
- RBT file—an optional "rawbits" file with an .rbt extension. The rawbits file is ASCII ones and zeros representing the data in the bitstream file. If you enter a -b option on the BitGen command line, an RBT file is produced in addition to the binary BIT file (see "-b (Create Rawbits File)" on page 467).
- LL file—an optional ASCII logic allocation file with a .ll extension. The logic allocation file indicates the bitstream position of latches, flip-flops, and IOB inputs and outputs. A .ll file is produced if you enter a -l option on the BitGen command line (see "-l

(Create a Logic Allocation File)" on page 473).

- MSK file—an optional mask file with an .msk extension. This file is used to compare relevant bit locations for executing a readback of configuration data contained in an operating FPGA. A MSK file is produced if you enter a -m option on the BitGen command line (see "-m (Generate a Mask File)" on page 473).
- BGN file—a report file containing information about the BitGen run.
- DRC file—a Design Rule Check (DRC) file for the design. A DRC runs and the DRC file is produced unless you enter a -d option on the BitGen command line (see "-d (Do Not Run DRC)" on page 467).

BitGen Options

Following is a description of command line options and how they affect BitGen behavior.

-b (Create Rawbits File)

Create a "rawbits" (*file_name*.rbt) file. 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.

-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 the "-j (No BIT File)" on page 472).

-f (Execute Commands File)

-f *command_file*

The -f option executes the command line arguments in the specified *command_file*.

-g (Set Configuration)

-g *option*:*setting*

The -g option specifies the startup timing and other bitstream options for Xilinx FPGAs. The settings for the -g option depend on the design's architecture. These options have the following syntax:

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 uses the multiple frame write feature in the bitstream to reduce 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.

ConfigRate

Virtex-II Pro uses 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: 4, 5, 7, 8, 9, 10, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60
Default: 4

DCMShutdown

When DCMShutdown is enabled, the DCM (Digital Clock Manager) 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: You should 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

DisableBandgap

Disables bandgap generator for DCMs to save power.

Settings: No, Yes
Default: No

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 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

Encrypt

Encrypts the bitstream. (For more information, see "Using Bitstream Encryption" on page 250.)

Settings: No, Yes
Default: No

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

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: Done, 1, 2, 3, 4, 5, 6, Keep
Default: 6

Key0, Key1, Key2, Key3, Key4, Key5

Sets Key*x* for bitstream encryption. The Pick option causes BitGen to select a random number for the value. (For more information, see "Using Bitstream Encryption" on page 250.)

Settings: Pick, <*hex_string*>
Default: Pick

KeyFile

Specifies the name of the input encryption file.

Settings: <*string*>

Keyseq0, Keyseq1, Keyseq2, Keyseq3, Keyseq4, Keyseq5

Sets the key sequence for key*x*. The settings are equal to the following:

- S = single

- F = first
- M = middle
- L = last

Settings: S, F, M, L
Default: S

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 is used to determine 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 is used to determine 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 this Startup cycle until DCI (Digitally Controlled Impedance) match signals are asserted.

Settings: NoWait, 0, 1, 2, 3, 4, 5, 6
Default: NoWait

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 IO. Refer to the data sheet for a description of SelectMAP mode and the associated pins.

Settings: No, Yes
Default: No

ProgPin

Adds an internal pull-up to the ProgPin pin. The Pullnone setting disables the pull-up. The pull-up affects the pin after configuration.

Settings: Pullup, Pullnone
Default: Pullnone

ReadBack

This option allows you to perform Readback by the creating the necessary bitstream. When specifying the -g Readback option, the .rba, .rbb, .rbd, and .msd file are created.

Security

Selecting Level1 disables Readback. Selecting Level2 disables Readback and Partial Reconfiguration.

Settings: None, Level1, Level2
Default: None

StartCBC

Sets the starting CBC (Cipher Block Chaining) value. The pick option causes BitGen to select a random number for the value.

Settings: Pick, *<hex_string>*
Default: Pick

StartKey

Sets the starting key number.

Settings: 0, 3
Default: 0

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 indicates 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 indicates 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 TdoPin pin, the serial data output for all JTAG instruction and data registers. Selecting one setting enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Settings: Pullup, Pulldown, Pullnone
Default: Pullup

TmsPin

This option selects an internal pullup or pulldown on the TMS (JTAG Mode Select) pin.

Settings: Pullnone, Pullup, Pulldown
Default: Pullup

UnusedPin

Adds a pull-up, a pull-down, or neither to the UnusedPin, the serial data output for all JTAG instruction and data registers. Selecting one setting enables it and disables the others. The Pullnone setting indicates 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.

-h or -help (Command Usage)

-h *architecture*

Displays a usage message for BitGen. The usage message displays all available options for BitGen operating on the specified *architecture*.

-j (No BIT File)

Do not create a bitstream file (.bit file). This option is generally 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 might 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 indicates the bitstream position of latches, flip-flops, and IOB inputs and outputs.

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 Hardware Debugger uses the **design.ll** file to locate signal values inside a readback bitstream.

-m (Generate a Mask File)

Creates a mask file. This file is used to compare relevant bit locations for executing a readback of configuration data contained in an operating FPGA.

-w (Overwrite Existing Output File)

Enables you to overwrite an existing BIT, LL, MSK, or RBT output file.

Using PROMGen

The PROMGen program is compatible with the following families.

- Virtex/Virtex-E/Virtex-II

PROMGen formats a BitGen-generated configuration bitstream (BIT) file into a PROM format file (Figure A-2).

The PROM file contains configuration data for the FPGA device. PROMGen converts a BIT file into one of three PROM formats: MCS-86 (Intel), EXORMAX (Motorola), or TEKHEX (Tektronix). It can also generate a Hex file format.

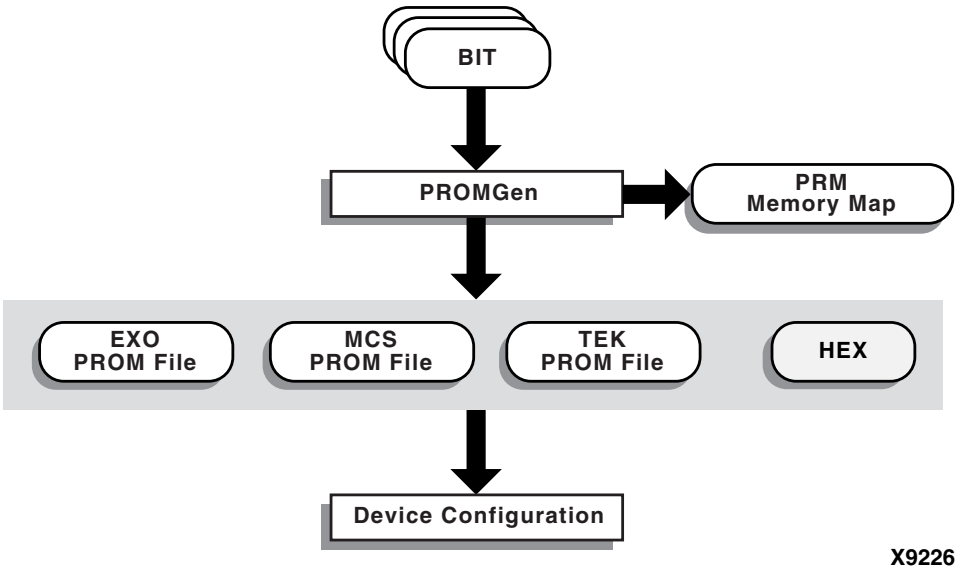


Figure A-2: **PROMGen**

There are two functionally equivalent versions of PROMGen. There is a stand-alone version you can access from an operating system prompt. You can also access an interactive version, called the PROM File Formatter, from inside the Design Manager for Alliance or the Project Manager in Foundation. This chapter describes the stand-alone version; the interactive version is described in the *PROM File Formatter Guide*.

You can also use PROMGen to concatenate bitstream files to daisy-chain FPGAs.

Note: If the destination PROM is one of the Xilinx Serial PROMs, you are using a Xilinx PROM Programmer, and the FPGAs are not being daisy-chained, it is not necessary to make a PROM file. See the *Hardware User Guide* for more information about daisy-chained designs

PROMGen Syntax

Use the following syntax to start PROMGen from the operating system prompt:

```
promgen [options]
```

Options can be any number of the options listed in"PROMGen Options" on page 475. Separate multiple options with spaces.

PROMGen Files

This section describes the PROMGen input and output files.

Input Files

The input to PROMGEN consists of BIT files— one or more bitstream files. BIT files contain configuration data for an FPGA design.

Output Files

Output from PROMGEN consists of the following files.

- PROM files—The file or files containing the PROM configuration information. Depending on the PROM file format used by the PROM programmer, you can output a TEK, MCS, or EXO file. If you are using a microprocessor to configure your devices, you can output a HEX file, containing a hexadecimal representation of the bitstream.
- PRM file—The PRM file is a PROM image file. It contains a memory map of the output PROM file. The file has a **.prm** extension.

Bit Swapping in PROM Files

PROMGen produces a PROM file in which the bits within a byte are swapped compared to the bits in the input BIT file. Bit swapping (also called "bit mirroring") reverses the bits within each byte, as shown in Figure A-3.

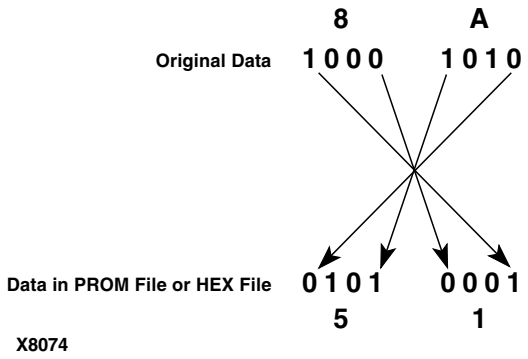


Figure A-3: **Bit Swapping**

In a bitstream contained in a BIT file, the Least Significant Bit (LSB) is always on the left side of a byte. But when a PROM programmer or a microprocessor reads a data byte, it identifies the LSB on the right side of the byte. In order for the PROM programmer or microprocessor to read the bitstream correctly, the bits in each byte must first be swapped so they are read in the correct order.

In this release of the Xilinx Development System, the bits are swapped for all of the PROM formats: MCS, EXO, and TEK. For a HEX file output, bit swapping is on by default, but it can be turned off by entering a -b PROMGen option that is available only for HEX file format.

PROMGen Options

This section describes the options that are available for the PROMGen command.

-b (Disable Bit Swapping—HEX Format Only)

This option only applies if the -p option specifies a HEX file for the output of PROMGen. By default (no -b option), bits in the HEX file are swapped compared to bits in the input BIT files. If you enter a -b option, the bits are not swapped. Bit swapping is described in "Bit Swapping in PROM Files" on page 474.

-c (Checksum)

```
promgen -c
```

The -c option generates a checksum value appearing in the .prm file. This value should match the checksum in the prom programmer. Use this option to verify that correct data was programmed into the prom.

-d (Load Downward)

```
promgen -d hexaddress0 filename filename...
```

This option loads one or more BIT files from the starting address in a downward direction. Specifying several files after this option causes the files to be concatenated in a daisy chain. You can specify multiple -d options to load files at different addresses. You must specify this option immediately before the input bitstream file.

The multiple file syntax is as follows:

```
promgen -d hexaddress0 filename filename...
```

The multiple **-d** options syntax is as follows:

```
promgen -d hexaddress1 filename -d hexaddress2 filename...
```

-f (Execute Commands File)

```
-f command_file
```

The -f option executes the command line arguments in the specified *command_file*.

-help (Command Help)

This option displays help that describes the PROMGen options.

-l option (Disable Length Count)

```
promgen -l
```

The -l option disables the length counter in the FPGA bitstream. It is valid only for 4000EX, 4000XL, 4000XLA, 4000XV, and SpartanXL Devices. Use this option when chaining together bitstreams exceeding the 24 bit limit imposed by the length counter.

-n (Add BIT FIles)

```
-n file1[.bit] file2[.bit]...
```

This option loads one or more BIT files up or down from the next available address following the previous load. The first -n option *must* follow a -u or -d option because -n does not establish a direction. Files specified with this option are not daisy-chained to previous files. Files are loaded in the direction established by the nearest prior -u, -d, or -n option.

The following syntax shows how to specify multiple files. When you specify multiple files, PROMGen daisy-chains the files.

```
promgen -d hexaddress file0 -n file1 file2...
```

The following syntax when using multiple -n options prevents the files from being daisy-chained:

```
promgen -d hexaddress file0 -n file1 -n file2...
```

-o (Output File Name)

```
-o file1[.ext] file2[.ext]...
```

This option specifies the output file name of a PROM if it is different from the default. If you do not specify an output file name, the PROM file has the same name as the first BIT file loaded.

ext is the extension for the applicable PROM format.

Multiple file names may be specified to split the information into multiple files. If only one name is supplied for split PROM files (by you or by default), the output PROM files are named *file*_#.ext, where *file* is the base name, # is 0, 1, etc., and *ext* is the extension for the applicable PROM format.

```
promgen -d hexaddress file0 -o filename
```

-p (PROM Format)

```
-p {mcs | exo | tek | hex}
```

This option sets the PROM format to one of the following: MCS (Intel MCS86), EXO (Motorola EXORMAX), TEK (Tektronix TEKHEX). The option may also produce a HEX file, which is a hexadecimal representation of the configuration bitstream used for microprocessor downloads. If specified, the -p option must precede any -u, -d, or -n options. The default format is MCS.

-r (Load PROM File)

```
-r promfile
```

This option reads an existing PROM file as input instead of a BIT file. All of the PROMGen output options may be used, so the -r option can be used for splitting an existing PROM file into multiple PROM files or for converting an existing PROM file to another format.

-s (PROM Size)

```
-s promsize1 promsize2...
```

This option sets the PROM size in kilobytes. The PROM size must be a power of 2. The default value is 64 kilobytes. The -s option must precede any -u, -d, or -n options.

Multiple *promsize* entries for the -s option indicates the PROM will be split into multiple PROM files.

Note: PROMGen PROM sizes are specified in bytes. *The Programmable Logic Data Book* specifies PROM sizes in bits for Xilinx serial PROMs (see -x option).

-u (Load Upward)

```
-u hexaddress0 filename1 filename2...
```

This option loads one or more BIT files from the starting address in an upward direction. When you specify several files after this option, PROMGen concatenates the files in a daisy chain. You can load files at different addresses by specifying multiple -u options.

This option must be specified immediately before the input bitstream file.

-x (Specify Xilinx PROM)

```
-x xilinx_prom1 xilinx_prom2...
```

The -x option specifies one or more Xilinx serial PROMs for which the PROM files are targeted. Use this option instead of the -s option if you know the Xilinx PROMs to use.

Multiple xilinx_*prom* entries for the -x option indicates the PROM will be split into multiple PROM files.

Examples

To load the file test.bit up from address 0x0000 in MCS format, enter the following information at the command line.

```
promgen -u 0 test
```

To daisy-chain the files test1.bit and test2.bit up from address 0x0000 and the files test3.bit and test4.bit from address 0x4000 while using a 32K PROM and the Motorola EXORmax format, enter the following information at the command line.

```
promgen -s 32 -p exo -u 00 test1 test2 -u 4000 test3 test4
```

To load the file test.bit into the PROM programmer in a downward direction starting at address 0x400, using a Xilinx XC1718D PROM, enter the following information at the command line.

```
promgen -x xc1718d -d 0x400 test
```

To specify a PROM file name that is different from the default file name enter the following information at the command line.

```
promgen options filename -o newfilename
```

Appendix B

Platform Flash Family PROMs

This appendix contains package specifications for the Platform Flash Family of In-System Programmable Configuration PROMs. The latest version of this information is available online at www.xilinx.com.

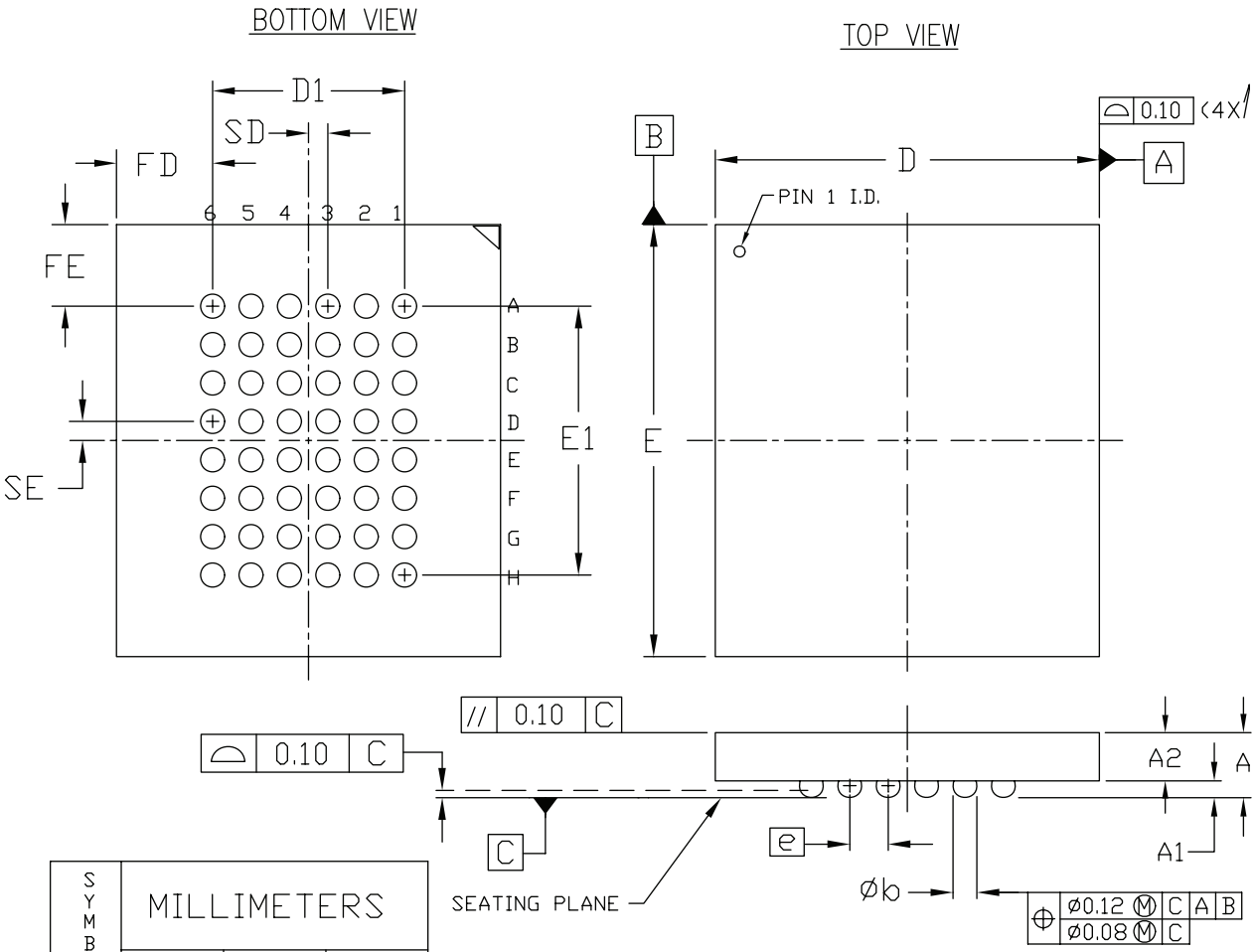
The data sheet for the Platform Flash Family of In-System Programmable Configuration PROMs can be accessed at www.xilinx.com/bvdocs/publications/ds123.pdf.

PROM Package Specifications

This section contains specifications for the following Platform Flash Family PROM packages:

- FS48 Package Specification
- VO48 Package Specification
- VO20 Package Specification

FS48 Package Specification



SYMBOL	MILLIMETERS		
	MIN.	NOM.	MAX.
A			1.20
A1	0.26	0.30	0.34
A2	0.75		0.86
D	7.90	8.00	8.10
D1		4.00	
e	0.80 BSC		
E	8.90	9.00	9.10
E1		5.60	
FD	2.00 REF.		
FE	1.70 REF.		
SD		0.40	
SE		0.40	
øb	0.35	0.40	0.45
M	6 X 8		

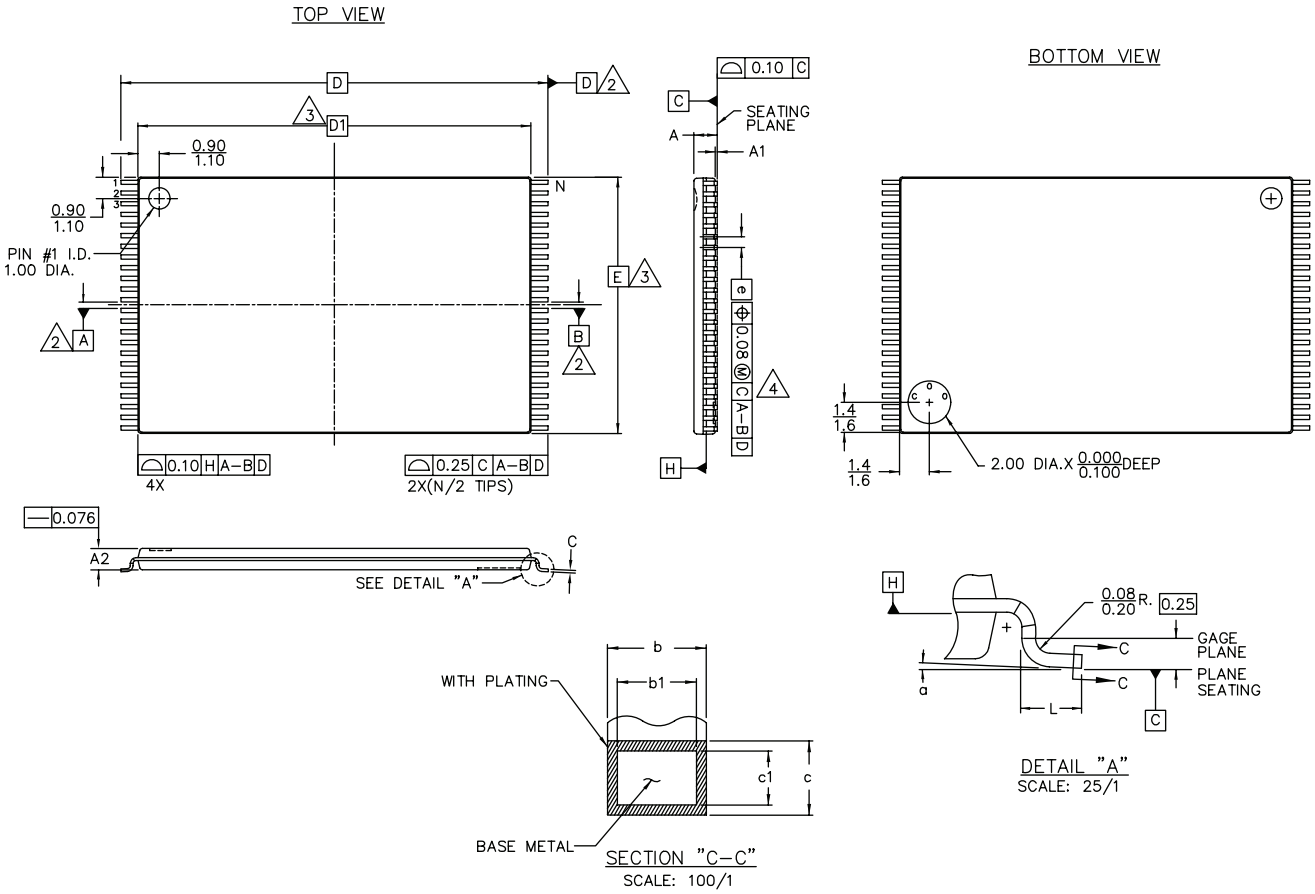
NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M–1994
2. SYMBOL "M" IS THE PIN MATRIX SIZE.

ug002_appB_fs48

Figure B-1: **FS48 Package Specification**

VO48 Package Specification



SYMBOL	DIMENSIONS IN MILLIMETERS			NOTE
	MIN.	NOM.	MAX.	
A	–	–	1.20	
A_1	0.05	–	0.15	
A_2	0.95	1.00	1.05	
b	0.17	0.22	0.27	
b1	0.17	0.20	0.23	
c	0.10	–	0.21	
c1	0.10	–	0.16	
D	19.80	–	20.20	2
D1	18.40	–	18.50	3
E	11.90	–	12.10	3
e	0.50 BSC			
L	0.50	0.60	0.70	
N	48			
α	0°	3°	5°	

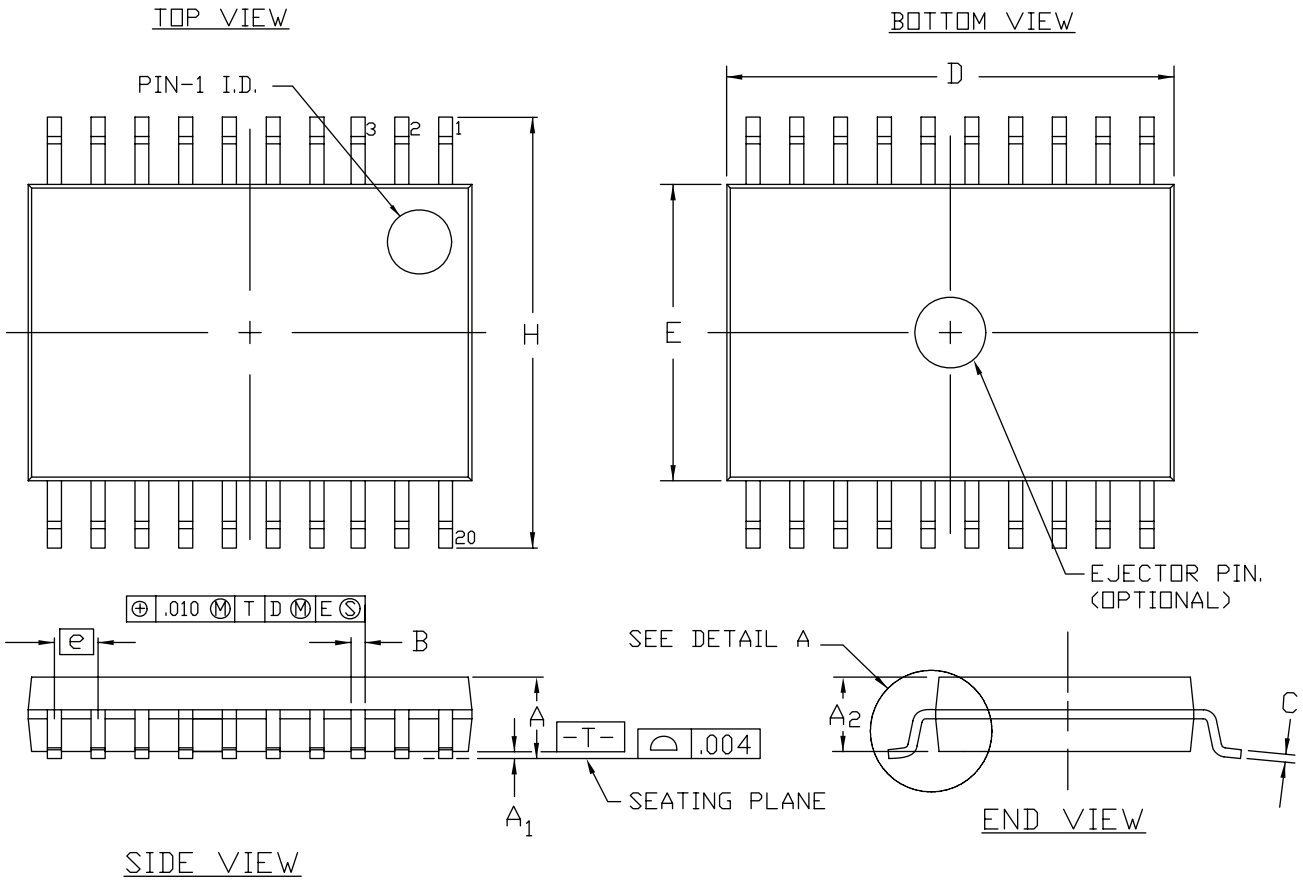
NOTES:

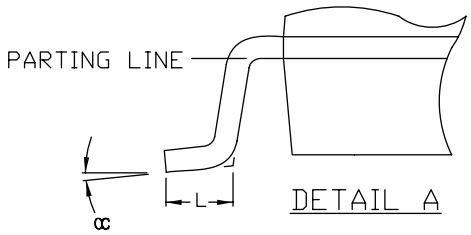
1. DIMENSIONS & TOLERANCES PER ASME Y14.5M. – 1994.
2. DATUM A–B AND D TO BE DETERMINED WHERE CENTERLINE BETWEEN LEADS EXIST PLASTIC BODY AT DATUM PLANE H.
3. "D1" & "E" DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS, AND ARE MEASURED AT THE PARTING LINE, MOLD FLASH OR PROTRUSIONS SHALL NOT EXCEED 0.15mm ON "E" AND 0.25 ON "D1" PER SIDE.
4. THE LEAD WIDTH DIMENSION DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.08mm TOTAL IN EXCESS OF THE LEAD WIDTH DIMENSION AT MAXIMUM MATERIAL CONDITION. DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OR THE FOOT. MINIMUM SPACE BETWEEN PROTRUSIONS AND AN ADJACENT LEAD TO BE 0.07mm. SEE SECTION "C–C".
5. LEAD FINISH: VO48 – 85/15 (Sn/Pb)
   VOG48 – 100% Matte Sn
6. CONTROLLING DIMENSION: MILLIMETERS.
7. THIS PART IS COMPLIANT WITH JEDEC SPECIFICATION MO–142 VARIATION DD.

ug002_appB_vo48

Figure B-2: **VO48 Package Specification**

VO20 Package Specification





SYMBOL	INCHES		
	MIN.	NOM.	MAX.
A	--	--	.047
A_1	.002	.004	.006
A_2	.031	--	.041
B	.007	--	.012
C	.004	--	.0079
D	.252	.256	.260
E	.169	.173	.176
e	.0256 BSC		
H	.244	.252	.260
L	.0197	.0236	.0295
α	0°	5°	8°

NOTES:

1. ALL DIMENSIONS AND TOLERANCES CONFORM TO ANSI Y14.5M-1982.
2. DIMENSION "D" DOES NOT INCLUDE MOLD PROTRUSION. ALLOWABLE MOLD PROTRUSION SHALL NOT EXCEED .006" PER SIDE.
3. DIMENSION "E" DOES NOT INCLUDE MOLD PROTRUSION. ALLOWABLE MOLD PROTRUSION SHALL NOT EXCEED .010" PER SIDE.
4. LEAD FINISH: VO20 – 85/15 (Sn/Pb)
   VOG20 – 100% Matte Sn
5. CONFORMS TO JEDEC MO-153-AC

ug002_appB_vo20

Figure B-3: **VO20 Package Specification**

Appendix C

Choosing the Battery for V_{BATT}

As noted in section "V_{BATT}" in Chapter 3, the battery backed-up RAM (BBRAM) array in the FPGA requires either V_{CCAUX} or V_{BATT} to keep the key programmed. Please reference DC and Switching Characteristics datasheet for V_{BATT} and I_{BATT} specifications. Once the Virtex-II Pro V_{CCAUX} power is applied, the BBRAM is no longer powered from the V_{BATT} pin. However, the leakage specification (I_{BATT}) still applies, as leakage results primarily from secondary structures rather than from the BBRAM itself.

Depending on the application, the BBRAM may need to be powered from time of shipment until used by a customer. This time period may be long, and involve storage at temperature extremes that shorten battery life. In the case of a primary (non-rechargeable) battery, it is safe to assume is that the battery must supply current to the system for the entire product life.

Battery Types and Chemistry Choices

In the engineering of a battery back-up system, designers need to decide if the battery system is to be primary (non-rechargeable) or secondary (rechargeable), and what cell voltage and chemistry are appropriate.

In the primary battery category, there are alkaline, lithium, and silver-oxide batteries. Their cell voltages are 1.5V, 3.0V, and 1.55V DC respectively at the beginning of service. Alkaline batteries are a good choice for operation below 54°C (130°F), and above –4°C (–20°F). Alkaline cells have a lifetime of less than one or two years without a load. Lithium batteries that are designed for high and low temperature operation may be used up to 125°C and down to –40°C. Lithium cells have a 15-year shelf or storage life, and if sized correctly may exhibit the same life in a system. Silver oxide cells are designed to operate from –10°C to +60°C, and have a less than 5% per year self-discharge rate at 21°C.

In the secondary battery category, there are nickel-cadmium, nickel metal hydride, lithium, and lead-acid batteries. Their cell voltages are 1.2V, 1.2V, 3.6V, and 2.0V DC respectively after charge under nominal load. The self-discharge rates of all of these batteries are such that they will typically need to be recharged within 30 days. The maximum number of deep (> 80%) charge/recharge cycles is usually less than 200 for these types of batteries.

Primary or Secondary

Primary batteries simplify the design as they cannot be recharged, and therefore require no extra circuitry. Secondary batteries require a method of charging, and can be damaged if charging currents are not controlled. Primary batteries can last up to 15 years (shelf life) which means that a battery could be placed in circuit, and forgotten about for the life of the product. Secondary batteries can provide significant current for a month or more, but then require recharging.

If the design goal is to configure the keys at the factory, and then ship the product, lithium primary cells are the first choice, followed by silver oxide as a second choice. These two examples will be detailed below.

If the design goal is to install keys in the field, through a system level secure key management system, then rechargeable batteries should be considered as they may outlast the product life as long as the product remains operational. A rechargeable example is also given.

Battery Cost and Care

Lithium batteries cost from USD1.00 to USD2.00 for 50 mAh to 1000 mAh capacity. For the rechargeable option, the battery cost is similar, but the extra cost of the charge controller IC must be added (USD2.50 to USD5.00).

Nickel cadmium batteries must be disposed of properly due to the cadmium metal. Lead acid batteries also have to be disposed of properly and silver is also considered a hazardous waste. Most batteries contain highly corrosive base or acid electrolytes, and will seriously damage or destroy electrical components if they leak. Batteries will leak if they are overcharged, if a primary battery type is charged, or if the battery suffers physical damage (dented or punctured).

Not all battery types are available throughout the world. Standard nickel cadmium batteries are generally available, as are alkaline cells. If operation anywhere in the world is needed, the standard AA alkaline cell may be obtained just about everywhere.

All batteries may have a risk of explosion or fire if they are shorted. The use of a small $^1/_8$-watt surface mount resistor as a fuse is a necessary precaution.

Battery Summary

Table C-1: **Primary Cell Types and Characteristics**

	Lithium	Silver Oxide	Alkaline
Nominal Voltage:	3.0V	1.55V	1.5V
Temperature Range:	–40°C to +150°C	–10°C to +60°C	–20°C to +54°C
Lifetime:	> 15 years	> 10 years	> 1 year
Comments:	• Best choice for long life • Industrial grade • Safely disposable	• Used in medical systems • Hazardous waste	• Inexpensive • Safely disposable (except Minnesota)

Table C-2: **Secondary Cell Types and Characteristics**

	Lead Acid	Nickel Cadmium	Nickel Metal Hydride	Lithium Ion (Polymer)
Nominal Voltage:	2.0V	1.2V	1.2V	3.6V
Temperature Range:	0°C to +40°C	0°C to +45°C	0°C to +40°C	0°C to +45°C
Lifetime:	> 5 years	> 5 years	> 5 years	> 2 years
Comments:	• Heavy • High current • Hazardous waste	• Low self-discharge rate • Hazardous waste	• High self-discharge rate • Safely disposable	• Complex charger circuit • Lightweight • Safely disposable

Design Examples

Case #1:

A 15-year operational life is desired, using a primary extended-temperature-range lithium button or coin cell. The current required is for ten Virtex-II or Virtex-II Pro FPGAs, or 10 x 100 nA = 1 uA, continuous. Leakage currents from the printed circuit board (PCB) are about 1 uA. Thus, total current = 2 µA.

The smallest coin cell from Panasonic is the BR1225A 48 mAh (milliampere hour), size 0.49" diameter by 0.1" height. The capacity in mAh divided by the current is the number of hours to end-of-life voltage. In this case, the number of hours available is :

48mAh / .002 mA = 24,000 hours (~2.7 years)

To obtain 15 years, we need a larger-capacity battery.

Notice in this case that the PCB leakage is limiting battery life. If the PCB is specified with lower leakage (which typically requires de-ionized water wash cycles), the smallest coin cell is adequate for about 5 years of life.

If a 15-year life is required, a larger cell may be used. The cost of the larger-capacity battery might be offset by eliminating the de-ionized PCB water wash.

End-of-life battery voltage is 2.0V DC. Start-of-life battery voltage is 3.0V DC, nominal. There are no restrictions on disposal of this type of battery, so there is no need for hazard labels.

Adding a series diode is a good way to isolate the battery from the device. It allows for current testing by placing a current meter across the diode.

Case #2:

A Rayovac 675G is primary silver oxide cell, size 0.5" diameter by 0.2" thick. It provides 190 mAh starting at 1.55V and ending at 1.0V. In the previous case, where 1 uA operating current plus 1 uA PCB leakage yielded a total current of 2 µA, a single cell would reach only about 11 years of life rather than the 15 years required. Limiting the PCB leakage to less current is one option to achieve longer battery life. Another option is to use two cells in parallel to achieve the 15-year life requirement.

Because silver is considered a hazardous waste, there are restrictions on the disposal of a silver oxide battery. Customer instructions and labels regarding battery disposal will add to the cost of the system.

Case #3:

A system is designed to utilize a lithium polymer rechargeable battery. The batteries require a sophisticated charge controller IC to prevent damage and optimize performance. A National LM3621 charger IC is chosen to keep the cell charged, and requires only 1 µA from the cell to function. The total requirement is now 3 µA. There is a 5% charge loss per month, with no load utilizing a Battery Engineering 152-102-6.3 4000 mAh Polymer Rechargeable Lithium Battery, so the shelf life after first charge is ~20 months. The life with 3 µA load is 21 years, much longer than the self discharge of 5% per month. If no other use of the current is required, a smaller battery may be used to balance out the self-discharge life. If a goal of one year is desired, the smallest cell from the catalog (86-54-3.5, 700 mAh) may be used.

Start-of-cycle voltage may be as high as 4.6V, so it must be dropped to 3.6V for the Virtex-II and Virtex-II Pro BBRAM power pin. Use of 1N914 or 1N4148 diodes in series is suggested to drop the voltage. End-of-cycle voltage will not go below 3.0V. The voltage drop from the diodes reduces this to a lower number, but this is still well above the 1.0V required by the V_{BATT} pin.

In general, a 10 nA current is enough to drop the voltage 0.4 volts using two 1N914/1N4148 diodes. If the total current is less than the minimum forward current specified for the diode, this guideline can be used to estimate how many diodes are required in series to achieve a certain voltage drop. Double-check with the diode vendor and with lab measurements to confirm.

There is no disposal restriction on the lithium polymer battery, but the charging restrictions require a special integrated circuit, and there are many rules to be followed in the proper use of a rechargeable lithium cell.

Case #4:

The cost and care required for the lithium battery in Case #3 were cause for rejection in the design review process. The alternative is nickel metal hydride, which has no disposal concerns, and can be charged with a simple series resistor. A Panasonic HHR55AAA/FT (requiring a AAA cell holder) will provide 560 mAh capacity, more than enough for our application. For charging, the self-discharge rate of the battery needs only to be offset. Nickel Metal Hydride batteries lose ~1% capacity per day, so charging at C/100 (where C= capacity in mAh) would just offset the self-discharge. Since it is safe to charge a nickel metal hydride battery at C/20 forever, a good compromise choice would be C/50, or 11 mA.

If the local power supply is 3.3V and the terminal battery voltage when fully charged is 1.45V, a series diode is required to prevent the battery from discharging when the power is turned off. Using a 1N914 diode, the drop is 0.7V, and 3.3 – 1.45 – 0.7 = 1.15V. For 11 mA, the required resistor is therefore about 105Ω. A 110Ω $\frac{1}{8}$-watt 5% resistor is all that is needed.

In order to replace the battery (if needed), a capacitor is added in parallel to the battery to hold the voltage for the time it takes to replace a battery.

For 2 µA, we solve for capacitance in $I = C \bullet dV/dt$, or $C = I \bullet dt/dV$. For a dV of 0.4 volts, a dT of 100 seconds, and a current of 1 µA, the capacitance required is 250 µF. The next safe standard value would be 470 µF. A high-quality, low-leakage aluminum electrolytic or tantalum would be recommended.

Glossary

ASIC

Application-specific integrated circuit, also called a *gate array*.

asynchronous

Logic that is not synchronized by a clock. Asynchronous designs can be faster than synchronous ones, but are more sensitive to parametric changes, and are thus less robust.

ATM

Asynchronous transfer mode. A very-high-speed (megahertz to gigahertz) connection-oriented bit-serial protocol for transmitting data and real-time voice and video in fixed-length packets (480byte payload, 5-byte header).

back annotation

Automatically attaching timing values to the entered design format after the design has been placed and routed in a field-programmable gate array (FPGA).

behavioral language

Top-down description from an even higher level than VDHL.

bitstream

The bitstream is a binary representation of an implemented FPGA design. The bitstream is generated by Xilinx bit generation tools (BitGen and Makebits) and is denoted with the **.bit** extension. For information on creating BIT files, refer to the *Hardware Debugger Reference/User Guide*.

block RAM

An 18-Kbit block of random access memory (RAM) inside the Virtex-II device. Dual-port and synchronous operation are desirable.

block SelectRAM

Fully-synchronous, dual-port memories in the Virtex-II FPGAs. Each of these memories contain 18 x 1024 (18,432) bits. The organization of each memory is configurable. Block SelectRAM resources complement smaller, distributed, LUT-based SelectRAM resources.

Boundary Scan interface

One of the configuration interfaces on the Virtex device. This is a bit-serial interface. The Boundary Scan interface is also known as the JTAG port. Also see *SelectMAP interface*.

capture data

The flip-flop and pad data saved from the logic cells and I/O blocks into the bitstream for readback. Use the CAPTURE_VIRTEX primitive in your HDL code to specify the trigger and clock for the capture operation.

compiler

Software that converts a higher-language description into a lower-level representation. For FPGAs, the complete partition, place, and process.

configurable logic block (CLB)

Xilinx-specific name for a block of logic surrounded by routing resources. The functional elements for constructing logic circuits. The Virtex-II CLB is made up of four slices, and each slice contains two Logic Cells.

configuration file

The internally stored file that controls the FPGA sot hat it performs the desired logic function. Also, the act of loading an FPGA with that file. That is, the process of programming Xilinx SRAM-based FPGAs with a bitstream.

configuration bitstream

Configuration commands with configuration data.

configuration clock (CCLK)

During configuration, the configuration clock (CCLK) is an output in Master modes or in the Asynchronous Peripheral mode but is an input in Slave, Synchronous Peripheral, Express, and SelectMAP/Slave Serial modes. After configuration, CCLK has a weak pull-up and can be selected as the readback clock.

configuration commands

Instructions for the Virtex-II device. There are two classes of Configuration Command — Major and Minor. The Major Commands read and write data to configuration registers in the Virtex-II device. The Minor commands instruct the Virtex-II configuration logic to perform specific functions.

configuration data

Bits that directly define the state of programmable logic. These are written to a Virtex-II device in a configuration bitstream, and read as readback data from a Virtex-II device.

configuration frame

The configuration bits in a Virtex-II device are organized in columns. A column of CLBs with the I/O blocks above and below the CLBs contain 48 frames of configuration bits. The smallest number of bits that can be read or written through the configuration interfaces is one frame.

configuration interface

A logical interface on the Virtex-II device through which configuration commands and data can be read and written. A interface consists of one or more physical device pins.

configuration readback

The operation of reading configuration data (also known as readback data) from a Virtex-II device.

constraints

Performance requirements imposed on the design, usually in the form of maximum allowable delay, or the required operating frequency.

$\overline{CS}$ pin

The $\overline{CS}$ pin is the Chip Enable pin for Virtex-II devices. It is used only in SelectMAP mode. When $\overline{CS}$ is asserted (Low) the device examines data on the Data bus. When $\overline{CS}$ is de-asserted (High), all CCLK transitions are ignored.

DataFrame

A DataFrame is a block of configuration data. A configuration bit-stream contains many such frames, each with a start bit and stop bits. Also see *configuration frame*.

digital signal processing (DSP)

The manipulation of analog data that has been sampled and converted into a digital representation. Examples are filtering, convolution, Fast-Fourier-Transform, and so on.

DIN pin

During serial configuration, the DIN pin is the serial configuration data input receiving data on the rising edge of CCLK. During parallel configuration, DIN is the D0 input. After configuration, DIN is a user-programmable I/O pin.

DONE pin

The DONE pin on a Xilinx FPGA is a bidirectional signal with an optional internal pull-up resistor. As an output, it indicates the completion of the configuration process. As an input, a low level on DONE can be configured to delay the global logic initialization and the enabling of outputs.

DOUT pin

During configuration in any mode except Express and SelectMAP, the DOUT pin is the serial configuration data output that can drive the DIN pin of daisy-chained slave FPGAs. DOUT data changes on the falling edge of CCLK, one-and-a-half CCLK periods after it is received at the DIN pin (in Master Serial Mode only).

DOUT/BUSY pin

For Virtex-II devices, the DOUT/BUSY pin has a dual purpose, depending on device mode. When the device is in Serial mode, this pin functions as DOUT. When the device is in SelectMAP/Slave Parallel mode, this pin functions as a handshaking signal. If BUSY is asserted (High) on a rising edge of CCLK, the data is not seen on the data bus, and should be held until the data is accepted.

dynamic random access memory (DRAM)

A low-cost read-write memory where data is stored on capacitors and must be refreshed periodically. DRAMs are usually addressed by a sequence of two addresses, row address, and column address, which makes them slower and more difficult to use than SRAMs. Also see *SRAM*.

electronic data interchange format (EDIF)

Industry standard for specifying a logic design in text (ASCII) form.

electrostatic discharge (ESD)

High-voltage discharge can rupture the input transistor gate oxide. ESD-protection diodes divert the current to the supply leads.

failure in time (FIT)

Describes the number of device failures statistically expected for a certain number of device-hours. Expressed as failures per one billion (10^9) device hours. Device temperature must be specified. Mean time between failure (MTBF) can be calculated from FIT. 10 FITs are good; 100 FITs are bad.

first-in first-out (FIFO)

FIFO memory where data is stored in the incoming sequence and is read out in the same sequence. Input and output can be asynchronous to each other. A FIFO needs no external addresses, although all modern FIFOs are implemented internally with RAMs driven by circular read and write counters.

flash

Non-volatile programmable technology, and alternative to electrically-erasable programmable read-only memory (EEPROM) technology. The memory content can be erased by an electrical signal. This allows in-system programmability and eliminates the need for ultraviolet light and quartz windows in the package.

flip-flop

Single-bit storage cell that samples its data input at the active (rising or falling) clock edge, and then presents the new state on its Q output after that clock edge, holding it there until after the next active clock edge.

frame

Also see *configuration frame*.

field programmable gate array (FPGA)

An integrated circuit that contains configurable (programmable) logic blocks and configurable interconnect between these blocks. Xilinx FPGAs are SRAM- based programmable logic devices (PLDs).

function generator

Also called a look-up table (LUT), with N inputs and one output. Can implement any logic function of its N inputs. N can be between 3 and 6; 4-input function generators are most popular.

gate

Smallest logic element with several inputs and one output. The AND gate output is High when all inputs are High. The OR gate output is High when at least one input is High. The NAND gate output is Low when all inputs are High. A 2-input NAND gate is used as the measurement unit for gate array complexity.

gate array

ASIC where transistors are predefined, and only the interconnect pattern is customized for the individual application.

graphical user interface (GUI)

The way of representing the computer output on the screen as graphics, pictures, icons, and windows. Pioneered by Xerox and the Apple Macintosh, now universally adopted, e.g., by Windows95 and others.

HDL

Hardware Description Language.

HardWire

Xilinx name for a low-cost derivative of an FPGA, where the configuration is fixed, but functionality and footprint are identical with the original FPGA-based design.

hierarchical design

Design description in multiple layers, from the highest (overview) tot he lowest (circuit details). An alternative is flat design, where everything is described at the same level of detail.

$\overline{\text{INIT}}$ pin

The $\overline{\text{INIT}}$ pin is a quadruple function signal. Before and during configuration, $\overline{\text{INIT}}$ is a bidirectional signal. A 1 - 10 kΩ external pull-up resistor is recommended. As an active-Low open-drain output, $\overline{\text{INIT}}$ is held Low during power stabilization and internal clearing of the configuration memory. As an active-Low input, it can be used to hold the FPGA in the internal WAIT state before the start of configuration. During configuration, a Low on this output indicates that a configuration data error has occurred. After the I/O become active in the Startup sequence, $\overline{\text{INIT}}$ becomes a user-programmable I/O.

intellectual property (IP)

In the legal sense, patents, copyrights, and trade secrets. In integrated circuits (ICs), predefined large functions, called “cores,” that help the user complete a large design faster.

JTAG

Joint Test Action Group. Previous name for IEEE 1149.1 boundary scan, a method for testing boards and integrated circuits. Also see *Parallel Cable III*.

LogiBLOX

Library of logic modules, often with user-definable parameters, like data width. Similar to LPM.

logic cell (LC)

Metric for FPGA density. The basic building block of the Virtex-II CLB. An LC includes a 4-input function generator, carry logic, and a storage element.

LPM

Library of Parametrized Modules. Library of logic modules, often with user-definable parameters, like data width. Similar to LogiBLOX.

LUT

Look-up table, also called a function generator with N inputs and one output. Can implement any logic function of its N inputs. N is between 3 and 6; most popular are 4-input LUTs.

LUT SelectRAM

Shallow RAM structure implemented in CLB look-up tables (LUTs). Also see *block SelectRAM*.

mapping

Process of assigning portions of the logic design to the physical chip resources (CLBs). With FPGAs, mapping is more demanding and more important a process than with gate arrays. Also see *synthesis*.

MTBF

Mean Time Between Failure. The statistically relevant up-time between equipment failure. Also see *failure in time (FIT)*.

MultiLINX cable

The MultiLINX cable provides many complex functions and can be loaded with new firmware as it becomes available. It can be connected to the host computer in two ways: via a Serial port or a USB port. The MultiLINX cable is supported by the Hardware Debugger software for Slave Serial and SelectMAP/Slave Parallel programming (as appropriate), as well as readback/verify. It is also supported by the JTAG programmer software for JTAG programming of both CPLDs and FPGAs.

MultiPRO Desktop Tool

The MultiPRO Desktop Tool is a multi-function programming and configuration cable. Designed for use in a desktop environment, the MultiPRO Desktop Tool interfaces to a PC using the industry standard IEEE 1284 parallel port. When mated with MultiPRO device adapters, it can program individual Platform Flash PROMs, XC18V00 ISP PROMs and CoolRunner-II CPLDs.

netlist

Textual description of logic and interconnects. Also see *XNF file* and *electronic data interchange format (EDIF)*.

NRE

Non-Recurring Engineering charges. Start-up cost for the creation of an ASIC, gate array, or HardWire. Pays for layout, masks, and test development. FPGAs and CPLD do not require NRE.

optimization

Design change to improve performance. Also see *synthesis*.

pad

Pad bits are extra bits used to make the total number of bits in a frame an integral multiple of 32, the number of bits in a configuration word. A pad word is an extra word used at the end of a configuration frame for pipelining. A pad frame is an extra configuration frame

used at the beginning of a configuration readback and at the end of a configuration write for pipelining.

Parallel Cable III

The Xilinx Parallel Cable III (model DLC5) is a serial download cable. The Parallel cable uses a serial 25-pin interface to the parallel port of a host computer and two 6-pin headers for flying-wire connectors to a target board. The Parallel cable is supported by the Hardware Debugger software for performing Slave Serial configuration of FPGAs only. The Parallel cable is also supported by the JTAG Programmer software for performing Slave Serial and Boundary Scan configuration of FPGAs, and Boundary Scan programming of CPLDs. For more information on using the Parallel cable, refer to Chapter 8 or this guide, the Hardware Debugger Reference/Users Guide, and the JTAG Programmer Guide.

partitioning

In FPGAs, the process of dividing the logic into subfunctions that can later be placed into individual CLBs. Partitioning precedes placement.

PCI

Peripheral Component Interface. Synchronous bus standard characterized by short range, light loading, low cost, and high performance. __?-MHz PCI can support data byte transfers up to ___? megabytes per second (Mb/s) on __? parallel data lines (including parity) and a common clock.

PCMCIA

Personal Computer Memory Card Interface Association. Physical and electrical standard for small plug-in boards for portable computers.

pin-locking

Rigidly defining and maintaining the functionality and timing requirements of device pins while the internal logic is still being designed or modified. Pin-locking has become important, since circuit board fabrication times are longer than PLD design implementation times.

PIP

Programmable Interconnect Point. In Xilinx FPGAs, a point where two signal lines can be connected, as determined by the device configuration.

placement

In FPGAs, the process of assigning specific parts of the design to specific locations (CLBs) on the chip. Usually done automatically. Also see *partitioning*.

PLD

Programmable Logic Device. Generic name for all programmable logic: PALs, CPLDs, and FPGAs.

programmable interconnect point

See *PIP*.

PROGRAM pin

The PROGRAM pin is an active-Low input that forces clearing of the FPGA configuration memory and is used to initiate a configuration cycle. While PROGRAM is held Low, the FPGA drives INIT Low and continues to clear the configuration memory. When PROGRAM goes High, the FPGA finishes the current clear cycle, executes another complete clear cycle, goes into a WAIT state, and releases INIT.

readback

Initiating a readback causes the configuration memory to become accessible to be serially clocked out and read from the device, or (byte-wide in SelectMAP/Slave Parallel modes). The configuration memory contains the configuration data, facilitating a Read-Verification of the data. The configuration memory can also contain the CLB output logic states facilitating a Read-Capture of the internal logic states.Read-Verification and Read-Capture are used by the Hardware Debugger for hardware verification. For information on the readback specification and timing, refer to *The Programmable Logic Data Book*. For information on using the readback component in a design, refer to the *Libraries Guide*. For information on enabling the readback function in the Implementation Software, refer to the *Development System Reference Guide*. For information on using the Hardware Debugger refer to the *Hardware Debugger Reference/User Guide*. For information on connecting the XChecker cable for readback, refer to the *Hardware Users Guide*.

readback data

Configuration data read from a Virtex-II device. The data is organized as configuration frames.

routing

The interconnection or the process of creating the desired interconnection of logic cells to make them perform the desired function. Routing follows after partitioning and placement.

schematic

Graphic representation of a logic design in the form of interconnected gates, flip-flops, and larger blocks. Older and more visually intuitive alternative to the increasingly more popular equation-based or high-level language textual description of a logic design.

SelectMAP interface

One of the configuration interfaces on the Virtex-II device. This is a byte-serial interface. The pins in the SelectMAP interface can be used as user I/O after configuration has been completed or remain configured as a configuration interface.

SelectRAM

Xilinx-specific name for RAM implemented in CLBs.

simulation

Computer modeling of logic and (sometimes) timing behavior of logic driven by simulation inputs (stimuli or vectors).

slice

A subdivision of the Virtex-II CLB. There are four vertical slices in each Virtex-II CLB. Each slice contains two Logic Cells.

SRAM

Static random access memory. Read-Write memory with data stored in latches. Faster than DRAM and with simpler timing requirements, but smaller in size and about four times more expensive than DRAM of the same capacity.

static timing

Detailed description of on-chip logic and interconnect delays.

submicron

The smallest feature size is usually expressed in micron (µ = millionth of a meter, or a thousandth of a millimeter). The state of the art is moving from 0.35µ to 0.25µ and soon may reach 0.18µ. The wavelength of visible light is 0.4µ to 0.8µ. 25.4µ = 1 mil, a thousandth of an inch.

synchronous

Circuitry that changes state only in response to a common clock, as opposed to asynchronous circuitry that responds to a multitude of derived signals. Synchronous circuits are easier to design, debug, modify, and better tolerate parameter changes and speed upgrades than asynchronous circuits.

sync word

A 32-bit word with a value that is used to synchronize the configuration logic.

synthesis

Optimization process of adapting a logic design to the logic resources available on the chip, like look-up tables, Longline, and dedicated carry. Synthesis precedes mapping.

TBUFs

Buffers with a 3-state option, where the output can be made inactive. Used for multiplexing different data sources onto a common bus.

timing

Relating to delays, performance, or speed.

timing driven

A design or layout method that takes performance requirements into consideration.

UART

Universal asynchronous receiver/transmitter. An 8-bit parallel-to-serial and serial-to-parallel converter, combined with parity and start-detect circuitry, and sometimes even FIFO buffers. Used widely in asynchronous serial communications interface, e.g., modems.

USB

Universal Serial Bus, A low-cost, low-speed, self-clocking bit-serial bus (1.5 MHz and 12 MHz) using four wires (V_{CC}, ground, differential data) to daisy-chain up to 128 devices.

VME

Older bus standard, popular with MC68000-based industrial computers.

WRITE pin

The WRITE pin is an input to Virtex-II devices in the SelectMAP/Slave Parallel mode, indicating to the device which direction data is flowing on the Data bus. When WRITE is asserted (Low), data is entering the device (configuration). When WRITE is de-asserted (High), data is leaving the device (readback). If WRITE changes state when the device isn't expecting it, an abort occurs. For more information on the WRITE pin, refer to *The Programmable Logic Data Book* and "Design Considerations" on page 59.

XNF file

Xilinx-proprietary description format for a logic design. Alternative is EDIF.

Index

Numerics

3-state output buffer 181

A

additional resources 15
addressing scheme 119
AGP-2X 209
ASIC
  defined 487
asynchronous
  defined 487
asynchronous transfer mode 487
ATM
  defined 487

B

back annotation
  defined 487
banks
  I/O 360
behavioral language
  defined 487
BF957
  composite pinout diagram 403
  routing with LVDS pairs 449
  standard routing 448
BG575
  composite pinout diagram 383
  routing with LVDS pairs 439
  standard routing 438
BG728
  composite pinout diagram 387
  routing with LVDS pairs 441
  standard routing 440
BGN files 467
bidirectional LVDS 245
bidirectional signals 182
BIT files
  description 466
  disabling 472
  loading downward 475
  loading up or down 475
  loading upward 476
bit swapping
  description 474
  disabling 475
BitGen
  -b option 467
  -d option 467
  description 465
  disabling DRC 467
  DRC file 467
  encryption options 251
  -g option 467 to ??
  -h option 472
  input files 466
  -j option 472
  -l option 473
  -m option 473
  options 467
  output files 466
  PCF files 466
  syntax 466
  -w option 473
bitstreams
  defined 487
  encryption 250
  loading encrypted 253
block RAM
  defined 487
block SelectRAM
  timing model 30
  timing parameters 31
Block SelectRAM Power section 453
board routability 428
board-level layout strategy 430
Boundary Scan
  interface defined 487
  mode 270
  models 463
Boundary Scan Description Language (BSDL) 463
BSDL files 463
buffers
  3-state output 181
  bidirectional LVDS 245
  global clock 60
  LDT 246
  output 179
BUFGCE 73

C

capacitors
  decoupling 423
capture operation 488
cascadable shift registers 140
CCLK
  defined 488
characteristics
  land pads 429
checksum 475
chip enable pin 489
ChipScope ILA 357
classification and export considerations 251
CLB / slice timing model 22
CLB Logic Power section 450
CLBs
  defined 488
clearing configuration memory 273
CLK 99
CLK2X 91
CLKDV 91
CLKFB 86
CLKIN 85
clock de-skew 80
clocks 60
  buffer input 63
  distribution 60
  forwarding 235
  global buffers 60
  global networks 60
  input clock tolerances 54
  multiplexer waveform 72, 73
  multiplexers 60
  output clock precision 55
  phase shifting 93
  resources 60
commands
  file, executing 475
compiler
  defined 488
configurable logic block (CLB) 488
configuration 269
  bitstream 488
  Boundary Scan mode 270
  clearing memory 273
  commands 488
  data 488
  file 488
  frame 488
  -g option 467 to ??
  interface 488
  JTAG 427
  Master SelectMAP mode 270
  Master Serial mode 270
  mode pins 269
  modes 269, 270
  process 272
  readback 489
  Slave SelectMAP mode 270
  Slave Serial mode 270
conflict resolution 115
constraining placement 145
constraints 489
content-addressable memory (CAM) 139
control signals 80
crosstalk 427
CS pin 489
CS144
  composite pinout diagram 370

D

Data Encryption Standard (DES) 250
DataFrame
  defined 489
DCI 210
  I/O buffer library 216
  software support 216
DCM Power section 454
DCMs
  clock de-skew 80
  control signals 80
  frequency synthesis 80
  miscellaneous timing parameters 56
  operating frequency ranges 53
  phase shifting 80, 93
  port signals 85
  timing model 53
  timing parameters 53
  waveforms 109
DDR
  input 227
  output 229
  output with 3-state control 231
  SDRAM 235
debugging
  using ChipScope ILA 357
decoupling capacitors 423
dedicated pins 269, 361
DES 250
de-skew circuit 85
differential signaling 246
Digital Clock Manager (DCM) 80
Digital Controlled Impedance (DCI) 210
DIN pin 489
distributed SelectRAM 112
DLLs
  characteristics 85
  source clock input 85
DONE pin 489
double data rate (DDR) 227
DOUT/BUSY pin 489
DRC
  disabling for BitGen 467
DRC file 467
dynamic random access memory (DRAM) 489
dynamic read operations 141

E

EDIF
  defined 490
electronic data interchange format (EDIF) 490
electrostatic discharge (ESD) 490
embedded multipliers 167
  timing model 34
  timing parameters 34
encryption
  BitGen options 251
  bitstream 250
export considerations 251

F

failure in time (FIT) 490
FDDRCPE 234
FDDRRSE 234
FF1152
  composite pinout diagram 395
  pinout compatibility diagram 407
  routing with LVDS pairs 445
  standard routing 444
FF1517
  composite pinout diagram 399
  routing with LVDS pairs 447
  standard routing 446
FF896
  composite pinout diagram 391
  pinout compatibility diagram 407
  routing with LVDS pairs 443
  standard routing 442
FG256
  bank information 372
  composite pinout diagram 371
  pinouts 362
  routing with LVDS pairs 433
  standard routing 432
FG456
  composite pinout diagram 375
  pinout compatibility diagram 406
  routing with LVDS pairs 435
  standard routing 434
FG676
  composite pinout diagram 379
  pinout compatibility diagram 406
  routing with LVDS pairs 437
  standard routing 436
field programmable gate array (FPGA) 490
fine phase adjustment 94
FIT
  defined 490
flash
  defined 490
flip-chip advantages 420
flip-chip packages 420
flip-flops
  defined 490
FPGA
  defined 490
frame
  defined 490
frequency synthesis 80
fully synchronous shift registers 146

G

global clock buffers 60
global clock nets 60
global clocks
  input to output timing parameters 49
  setup and hold timing parameters 51
Glossary 487
graphical user interface (GUI) 491
GTL 192
GTL+ 193
GUI
  defined 491

H

Hardware Description Language (HDL) 491
hierarchical design
  defined 491
HSTL_I 194, 198
HSTL_II 195, 199
HSTL_III 196, 200
HSTL_IV 197, 201

I

I/O banks 360
I/O Buffer Information Specification (IBIS) 458
I/Os
  single-ended standards 174
IBIS 458
  advantages 458
  file structure 459
  generation 458
  I/V and dV/dt curves 459
  models 458
  ramp keyword 460
  simulations 460
  simulators 462
IEEE 1149.1 463
IEEE 1532 463
INIT pin 491
input clock tolerances
  timing parameters 54
input DDR 227
input files
  BitGen 466
  PROMGen 474
Input/Output Power section 455
intellectual property (IP) 491
IOBs
  3-state timing parameters 44
  input timing parameters 37
  timing model 36
IOBUF 182
IOSTANDARD attribute 218
IP
  defined 491

J

JTAG 427
  defined 491

K

keys 253
  creating 251

L

land pad characteristics 429
land pads 429
layout strategy 430
LDT 246
  buffers 246
legacy support 87
Lightning Data Transport (LDT) 246
LL files 466, 473
loading 253
locked output 86
LogiBLOX
  defined 491
logic
  allocation file 473
logic cell (LC) 491
look-up table (LUT) 492
low voltage differential signaling (LVDS) 241
low voltage positive emitter-coupled logic (LVPECL) 247
LPM
  defined 491
LUTs 139
  defined 492
LVCMOS15 207
LVCMOS18 207
LVCMOS25 208
LVCMOS33 208
LVDS 241
  3-state buffer termination 245
  bidirectional 245
  primitives 241
  receiver termination 242
  transmitter termination 243
LVPECL 247 to ??
  primitives 247
  receiver termination 248
  transmitter termination 249
LVTTL 206

M

mapping 492
mask file 473
Master SelectMAP mode 270
Master Serial mode 270
memory
  clearing 273
modes
  configuration 270
    Boundary Scan 270
    Master SelectMAP 270
    Master Serial 270
    Slave SelectMAP 270
    Slave Serial 270
  NO_CHANGE 114
  READ_FIRST 114
  WRITE_FIRST 113
MSK files 467
MTBF
  defined 492
MultiLINX cable 492
multiplexers 150
  clocks 60
  large 150
  wide-input 154
multipliers
  embedded 167

N

National Institute of Standards and Technology (NIST) 250
netlist
  defined 492
NO_CHANGE mode 114
Non-Registered Multiplier Power section 454
non-solder-mask defined (NSMD) 430

O

OBUF 179
OBUFT 181
operating frequency ranges 53
optimization 492
output buffer (OBUF) 179
output clock precision
  timing parameters 55
output DDR 229
output DDR with 3-state control 231
output drive strength 185
output files
  BitGen 466
  name, PROMGen 476
  overwriting 473
  PROMGen 474
output power/ground pairs 187
overview of user guide 15

P

package specifications 409
packages
  flip-chip 420
  thermal considerations 421
pads 429
  defined 492
parallel termination 426
partitioning 493
PC20-84 specification 480
PCB layout considerations 423
PCF files
  BitGen 466
PCI
  defined 493
PCI33_3 206
PCI66_3 206
PCIX 206
PCMCIA
  defined 493
persist option 269
phase shifting 80, 93
pin-locking 493
pinout diagrams 369
pinout information 360
pins 269
  chip enable 489
  CS 489
  dedicated 269, 361
  DIN 489
  DONE 489
  DOUT/BUSY 489
  dual-function 269
  INIT 491
  power 273
  PROGRAM 494
  types 360
  WRITE 496
pin-to-pin timing model 48
PIP
  defined 493
placement 493
placement constraints 145
port addressing scheme 119
port signals 85
power estimator 450
  results 456
power pins 273
primitives
  LVDS 241, 247
PRM files 474
PROGRAM pin 494
programmable interconnect point (PIP) 493
programmable logic device (PLD) 493
PROMGen
  -b option 475
  -c option 475
  -d option 475
  description 473
  examples 477
  flow diagram 473
  -help option 475
  input files 474
  -l option 475
  -n option 475
  -o option 476
  options 475
  output file name 476
  output files 474
  -p option 476
  -r option 476
  -s option 476
  supported families 473
  -u option 476
  -x option 477
PROMs
  bit swapping 474

files, description 474
formats 476
loading files 476
multiple files 477
package specifications 479
sizes 476

R

rawbits file 467
RBT files 466, 467
read operations
dynamic 141
static 141
READ_FIRST mode 114
readback
defined 494
readback data 494
Registered Multiplier Power section 455
resources available 15
routability guidelines 428
routing
amount 452
challenges 428
defined 494
examples 431
strategy 429
RST 86

S

schematic
defined 494
SelectI/O-Ultra
single-ended resources 174
SelectMAP
interface defined 494
SelectRAM
defined 494
distributed 112
series termination 426
shift registers
cascadable 140
fully synchronous 146
operation 139
static length 147
signals
bidirectional 182
simulation
defined 494
simultaneous switching output (SSO) 187
single-ended I/O standards 174
single-ended SelectI/O-Ultra resources 174
Slave SelectMAP mode 270
Slave Serial mode 270
slew rate 185
slices 160
defined 494
SO20 specification 481
solder balls 428
solder-mask defined (SMD) 430
specifications
PC20-84 480
PROM packages 479
SO20 481
VQ44 482
SRAM
defined 495
SRL16 139
SRLC16 140
SSTL2_I 204
SSTL2_II 205
SSTL3_I 202
SSTL3_II 203
STARTUP_WAIT attribute 86
static length shift registers 147
static read operations 141
static timing
defined 495
submicron
defined 495
Sum of Products (SOP) 160
sync word
defined 495
synchronous
defined 495
synchronous DRAM 235
synthesis
defined 495

T

TBUF
defined 495
termination techniques 186
terminations
parallel 426
series 426
thermal considerations 421
thermal management 422
timing 495
Timing Analyzer 21
timing driven
defined 495
timing models 21
block SelectRAM 30
CLB / slice 22
DCM 53
embedded multiplier 34
IOB 36
pin-to-pin 48
timing parameters
block SelectRAM 31
DCM 53
embedded multiplier 34
general slice 23
global clock input to output 49
global clock setup and hold 51
input clock tolerances 54
IOB 3-state 44
IOB input 37
miscellaneous DCM 56
output clock precision 55
slice distributed RAM 26
slice SRL 29
TMULT 34
transmission line effects 186, 425
TRCE 21
Triple Data Encryption Algorithm (TDEA) 250
Triple DES 250

U

UART
defined 495
USB
defined 495

V

VBATT 253, 361, 483
VCC decoupling 423
VCCAUX 361
VCCINT 361
VCCO 186, 361
verification
using ChipScope ILA 357
VHDL and Verilog templates 75, 105, 122, 132, 136, 148, 156, 162, 172, 235
Virtex-II
DCI 210
DES 250
LUTs 139
multiplexers 150
package specifications 409
pinout diagrams 369
pinouts 360
power estimator 450
slices 160
VME
defined 495
VQ44 specification 482
VREF 185

W

wide-input multiplexers 154
WRITE pin 496
WRITE_FIRST mode 113