

Spartan-3 Generation Configuration User Guide

*Extended Spartan-3A,
Spartan-3E, and Spartan-3
FPGA Families*

UG332 (v1.5) March 16, 2009





Xilinx is disclosing this user guide, manual, release note, and/or specification (the "Documentation") to you solely for use in the development of designs to operate with Xilinx hardware devices. You may not reproduce, distribute, republish, download, display, post, or transmit the Documentation in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Xilinx expressly disclaims any liability arising out of your use of the Documentation. Xilinx reserves the right, at its sole discretion, to change the Documentation without notice at any time. Xilinx assumes no obligation to correct any errors contained in the Documentation, or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information.

THE DOCUMENTATION IS DISCLOSED TO YOU "AS-IS" WITH NO WARRANTY OF ANY KIND. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DOCUMENTATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS. IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOSS OF DATA OR LOST PROFITS, ARISING FROM YOUR USE OF THE DOCUMENTATION.

© 2006–2009, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/05/06	1.0	Initial release.
02/26/07	1.1	Added configuration information for the Spartan-3AN FPGA family. Added Chapter 10, "Internal Master SPI Mode" describing how a Spartan®-3AN FPGA configures from its internal In-System Flash memory. Increased ConfigRate settings for Spartan-3A/3AN FPGAs based on improved data setup time (Table 4-11 and Table 5-8). Added links to new reference designs using the Spartan-3E and Spartan-3A Starter Kit boards.
05/23/07	1.2	Added Spartan-3A DSP family configuration information. Added " Bitstream Format ," page 39 . Added " Indirect SPI Programming using iMPACT ," page 134 . Updated " Limitations when Reprogramming via JTAG if FPGA Set for BPI Configuration ," page 171 . Updated JTAG ID values in Table 12-4 , page 246 . Added more information to " Configuration Watchdog Timer (CWDT) and Fallback ," page 285 .
11/21/07	1.3	Noted in " Non-Continuous SelectMAP Data Loading " that " Deasserting CSL_B " is not supported in the Spartan-3A, Spartan-3AN, and Spartan-3A DSP FPGA families. Added " Indirect Parallel Flash Programming Using iMPACT ". Updated Figure 5-14 to show D[7:0] inputs clocked on rising CCLK edge. Added " Byte Swapping " description. Added JTAG TAP Controller State descriptions in Table 9-2 . Updated Spartan-3AN FPGA Variant Select options in Table 10-2 . Updated Figure 14-20 and description to note that MultiBoot Variant Select is based on Read Command from GENERAL2 register when NEWMODE=1, not BOOTVSEL bits in MODE_REG. Updated software version references throughout. Updated documentation links throughout.
07/01/08	1.4	Added " Schedule of Figures " and " Schedule of Tables ". Updated STMicroelectronics and Intel Flash memory references to Numonyx. Noted HSWAP_EN is a dedicated pin in the Spartan-3 family. Noted that Extended Spartan-3A family FPGA DOUT also functions as BUSY as in Spartan-3/3E FPGAs. Noted that Extended Spartan-3A family FPGAs do not support 1.8V configuration due to V _{CCO2T} requirement of 2.0V. Updated references to iMPACT indirect Flash programming support using Spartan-3 generation devices.
03/16/09	1.5	Updated nomenclature for Extended Spartan-3A family. Added fifth paragraph below " Overview and Design Considerations ," page 27 . Added " Additional Resources ," page 47 , and " V_{CCAUX} Level ," page 77 . Revised Table 5-6 , page 152 , Figure 5-6 , page 159 , and Figure 5-14 , page 169 . Revised Table 14-8 , page 278 . Added Chapter 17, "Configuration Details" and Chapter 18, "Readback" .

Table of Contents

Revision History	3
Schedule of Figures	15
Schedule of Tables	21
Chapter 1: Overview and Design Considerations	
Design Considerations	27
Will the FPGA load configuration data itself from external or internal memory or will an external processor/microcontroller download configuration data?	28
Does the application use a single FPGA or multiple FPGAs?	32
Is the “easiest possible” configuration solution the more important consideration? ...	33
Will the application require a nonvolatile FPGA?	34
Is the “lowest cost” solution the more important consideration?	34
Is “fastest possible configuration time” the more important consideration?	34
Will the FPGA or FPGAs be loaded with a single configuration image or loaded with multiple images?	35
What I/O voltages are required in the end application?	35
Will the FPGA application need to store nonvolatile data?	35
Should the FPGA I/O pins be pulled High via resistors during configuration?	35
Does the application target a specific FPGA density or should it support migrating to other FPGA densities in the same package footprint?	36
What is the anticipated production lifetime for the end product?	37
Do you want to protect your FPGA bitstream against unauthorized duplication?	38
Do you want to load multiple FPGAs with the same configuration bitstream?	38
Will the FPGA be used in a PCI™ application?	38
Where to go for debugging support	38
FPGA Configuration Bitstream Sizes	38
Uncompressed Bitstream Image Size	38
Bitstream Format	39
Synchronization Word	39
Array ID	40
Data Frames	40
CRC	41
Bitstream Compression	41
Packet Format	42
Setting Bitstream Options, Generating an FPGA Bitstream	42
ISE Software Project Navigator	42
BitGen Command Line Utility	46
Additional Resources	47
Data Sheets	47
Application Notes	47
Chapter 2: Configuration Pins and Behavior during Configuration	
General Configuration Control Pins	49

Choose a Configuration Mode: M[2:0]	50
M[2:0] Functional Differences between Spartan-3 Generation Families	50
Extended Spartan-3A Family and Spartan-3E FPGA Families	51
Spartan-3 FPGA Family	51
Defining M[2:0] after Configuration for Minimum Power Consumption.	51
DONE Pin	52
Associated Bitstream Generator (BitGen) Options	52
DONE Synchronizes Multiple FPGAs in a Daisy Chain or Broadside (Ganged) Configuration	54
Program or Reset FPGA: PROG_B	56
Configuration Clock: CCLK	56
CCLK Differences between Spartan-3 Generation FPGA Families	56
CCLK Design Considerations	58
ConfigRate: Bitstream Option for CCLK.	60
Persist: Reserve CCLK As Part of SelectMAP Interface	60
Extended Spartan-3A Family and Spartan-3E FPGA Families	61
Spartan-3 FPGA Family	61
Initializing Configuration Memory, Configuration Error: INIT_B	61
After Configuration	61
Extended Spartan-3A Family FPGA Post-Configuration CRC	62
Extended Spartan-3A Family and Spartan-3E FPGA Families	62
Spartan-3 FPGA Family	62
Pull-Up Resistors During Configuration	62
Pins with Dedicated Pull-Up Resistors during Configuration	62
Pins with Optional Pull-Up Resistors during Configuration	64
FPGA Pull-Up Resistor Values	64
Pin Descriptions	65
Pin Behavior During Configuration	69
Extended Spartan-3A Family FPGA	70
Spartan-3E FPGAs	71
Spartan-3 FPGAs	72
Default I/O Standard During Configuration	72
Lowering VCCO_2 After Configuration in Extended Spartan-3A Family	73
Design Considerations for the HSWAP, M[2:0], and VS[2:0] Pins	75
Dedicating the HSWAP, PUDC_B, M[2:0], and VS[2:0] Pins	75
Reusing HSWAP, PUDC_B, M[2:0], and VS[2:0] After Configuration	75
Spartan-3E HSWAP Considerations	75
Dual-Purpose Pins Become User I/O	76
V_{CCAUX} Level	77

Chapter 3: Master Serial Mode

Master Serial Mode Connections	83
Voltage Compatibility.	84
Platform Flash PROM	84
FPGA	84
Spartan-3E and Spartan-3A/3A DSP FPGAs with V _{CCAUX} at 2.5V	84
Spartan-3 FPGAs.	84
JTAG Interface	84
Supported Platform Flash PROMs	85
CCLK Frequency	86
Daisy-Chain Configuration	86

Ganged or Broadside Configuration	86
JTAG Interface	88
Storing Additional User Data in Platform Flash	88
Generating the Bitstream for a Master Serial Configuration	90
ConfigRate: CCLK Frequency	90
StartupClk: CCLK	90
DriveDone: Actively Drive DONE Pin	90
GTS_cycle: Global Three-State Release Timing for Daisy Chains	90
Preparing a Platform Flash PROM File	90
iMPACT	90
Platform Flash In-System Programming via JTAG using iMPACT	95
Prepare Board for Programming	95
Programming via iMPACT	95
Production Programmers	99
Additional Information	99

Chapter 4: Master SPI Mode

Master SPI Mode Differences between Spartan-3 Generation FPGA Families	104
Choosing a Compatible SPI Serial Flash	104
SPI Flash PROM Density Requirements	107
FPGA Connections to the SPI PROM	108
Voltage Compatibility	112
Power-On Precautions if System 3.3V Supply is Last in Sequence	112
Extended Spartan-3A Family and Configuration Watchdog Timer	114
CCLK Frequency	114
SPI Flash Interface after Configuration	116
If Not Using SPI Flash after Configuration	116
If Using SPI Flash Interface after Configuration	117
SPI Master Interface using FPGA Logic	117
Accessing SPI Flash PROM	118
Accessing other SPI-compatible Peripherals	118
Daisy-Chain Configuration	119
Ganged or Broadside Configuration	120
Programming Support	120
Third-Party Programmer (Off-board Programming)	121
Direct, SPI In-System Programming	122
Requirements for iMPACT Direct Programming Support	122
Programmable Cable Connections	122
Forcing FPGA SPI Bus Pins to High-impedance During Programming	123
Direct, In-system SPI Programming Using FPGA as Intermediary	124
Indirect, In-System SPI Programming Using FPGA JTAG Chain	124
Generating the Bitstream for a Master SPI Configuration	125
ConfigRate: CCLK Frequency	125
StartupClk: CCLK	125
DriveDone: Actively Drive DONE Pin	125
DONE_cycle: Daisy Chains with Spartan-3E Master	126
GTS_cycle: Global Three-State Release Timing for Daisy Chains	126

Preparing an SPI PROM File	126
iMPACT	126
PROMGen	130
Direct SPI Programming using iMPACT	131
Prepare Board for Programming	131
Programming via iMPACT	131
Indirect SPI Programming using iMPACT	134
Programming Setup	134
Using iMPACT	134
Serial Peripheral Interface (SPI) Configuration Timing	138
Multi-Package Layout	141
Saving Power	142
Deassert CSO_B to Enter Standby Mode	142

Chapter 5: Master BPI Mode

Overview	143
Master BPI Mode Differences between Spartan-3 Generation FPGA Families	145
PROM Address Generation	146
Voltage Compatibility	150
Compatible Parallel NOR Flash Families	150
Required Parallel Flash PROM Densities	151
CCLK Frequency	152
Using the BPI Interface after Configuration	153
Precautions Using x8/x16 Flash PROMs	154
Daisy Chaining	155
Parallel Daisy Chaining	156
Serial Daisy Chaining (Extended Spartan-3A Family FPGAs Only)	157
Using Xilinx Platform Flash PROMs with Master BPI Mode	158
ConfigRate Settings Using Platform Flash	160
Generating the Bitstream for a Master BPI Configuration	160
ConfigRate: CCLK Frequency	160
StartupClk: CCLK	160
DriveDone: Actively Drive DONE Pin	160
GTS_cycle: Global Three-State Release Timing for Daisy Chains	161
Preparing a Parallel NOR Flash PROM File	161
iMPACT	161
Indirect Parallel Flash Programming Using iMPACT	166
In-System Programming Support	166
Power-On Precautions if 3.3V Supply is Last in Sequence	167
Extended Spartan-3A Family and Configuration Watchdog Timer	168
Byte Peripheral Interface (BPI) Timing	169
Limitations when Reprogramming via JTAG if FPGA Set for BPI Configuration	171
Spartan-3E BPI Mode Interaction with Right and Bottom Edge Global Clock Inputs	171
BPI Data Ordering	172

Chapter 6: Master Parallel Mode

Chapter 7: Slave Parallel (SelectMAP) Mode

Voltage Compatibility	180
Daisy Chaining	181
Spartan-3E/Extended Spartan-3A Family Slave Parallel Daisy Chains	181
Slave Parallel Daisy Chains Using Any Modern Xilinx FPGA Family	182
SelectMAP Data Loading	182
CSI_B	182
RDWR_B	183
CCLK	183
BUSY	183
Continuous SelectMAP Data Loading	183
Non-Continuous SelectMAP Data Loading	185
Deasserting CSI_B	185
Pausing CCLK	186
SelectMAP ABORT	186
Configuration Abort Sequence Description	187
Readback Abort Sequence Description	187
ABORT Status Word	188
Resuming Configuration or Readback After an Abort	189
Persist	190
SelectMAP Reconfiguration	190
SelectMAP Data Ordering	191
Byte Swapping	191

Chapter 8: Slave Serial Mode

Voltage Compatibility	195
Daisy Chaining	195

Chapter 9: JTAG Configuration Mode and Boundary-Scan

JTAG Cable Voltage Compatibility	198
JTAG Device ID	199
JTAG User ID	199
Using JTAG Interface to Communicate to a Configured FPGA Design	199
Boundary-Scan for Spartan-3 Generation FPGAs	
Using IEEE Standard 1149.1	199
Test Access Port (TAP)	200
TAP Controller	202
Boundary-Scan Architecture	203
Boundary-Scan Register	203
Bit Sequence Boundary-Scan Register	204
Instruction Register	205
BYPASS Register	206
Identification (IDCODE) Register	206
JTAG Configuration Register (Boundary-Scan)	206
USERCODE Register	207

USER1 and USER2 Registers	207
Using Boundary-Scan in Spartan-3 Generation FPGAs	207
Programming Cables and Headers	207
Programming an FPGA Using JTAG	208
Mode Pin Considerations when Programming a Spartan-3AN FPGA via JTAG using iMPACT	213
Configuration via JTAG using an Embedded Controller	213

Chapter 10: Internal Master SPI Mode

Internal Flash Memory	216
Mode Select Pins, M[2:0]	216
Variant Select Pins, VS[2:0]	216
Supply Voltage Requirements	217
VCCAUX	217
VCCO_2	217
Sequencing	217
Accessing the Internal SPI Flash PROM After Configuration	217
No Configuration Daisy Chains in Internal Master SPI Mode	218
Generating the Bitstream for a Master SPI Configuration	218
ConfigRate: CCLK Frequency	218
StartupClk: CCLK	218
DriveDone: Actively Drive DONE Pin	218
Programming a Spartan-3AN FPGA Using JTAG	219
Preparing an In-System Flash Programming File	220
iMPACT	220
PROMGen	226
Programming Spartan-3AN FPGAs Using iMPACT	228
Third-Party Programmer Support	228
BPM Microsystems	228
Production Hardware Programming Solutions	228
Programming Socket Modules and Software	229

Chapter 11: Configuration Bitstream Generator (BitGen) Settings

Chapter 12: Sequence of Events

Overview	239
Setup for Configuration (Steps 1-3)	239
Wake from Reset	239
Power-On Reset (POR)	240
PROG_B Pin	241
Power-Up Timing	241
Clear Configuration Memory (Initialization)	243
Sample Control Pins	243
Delaying Configuration	243
Bitstream Loading (Steps 4-7)	244
Synchronization	244
Check Array IDCODE	245
Load Configuration Data Frames	247

Cyclic Redundancy Check	247
Startup	248
Startup Clock Source	250
Waiting for DCMs to Lock, DCI to Match	250

Chapter 13: Configuration-Related Design Primitives

Boundary-Scan (BSCAN)	253
Usage	254
Port Descriptions	254
Start-Up (STARTUP)	255
Usage	256
Port Descriptions	256
Readback Capture (CAPTURE)	256
Usage	257
Port Description	257
Attributes	257
Internal Configuration Access Port (ICAP)	258
Usage	258
Port Description	258
Device DNA Access Port (DNA_PORT)	259
Usage	259
Port Descriptions	260
Attributes	260

Chapter 14: Reconfiguration and MultiBoot

Overview	261
MultiBoot Options Compared between Spartan-3 Generation FPGA Families	261
Spartan-3E MultiBoot	263
Generating a Spartan-3E MultiBoot PROM Image using iMPACT	264
PROMGen Report File	269
Spartan-3E MultiBoot using Xilinx Platform Flash PROMs	270
Extended Spartan-3A Family MultiBoot	271
Specifying the Next MultiBoot Configuration Address	271
Required Data Spacing between MultiBoot Images	272
Flash Sector, Block, or Page Boundaries	272
Additional Memory Space Required for DCM_WAIT	272
MultiBoot Command Sequence (ICAP Example)	273
Design Specification	273
FPGA Application Run Time	274
MultiBoot from an Address Preloaded during Configuration	274
MultiBoot to a Address Specified by the FPGA Application	274
MultiBoot using SelectMAP	275
MultiBoot using Slave Serial	275
MultiBoot using JTAG	276
MultiBoot Registers	276
Next MultiBoot Start Address (GENERAL1 , GENERAL2)	276
Command Register (CMD)	277
Configuration Mode Register (MODE_REG)	278

Generating an Extended Spartan-3A Family MultiBoot PROM Image using iMPACT	278
Configuration Fallback	284
Configuration Watchdog Timer (CWDT) and Fallback	285
CRC Error and Fallback	285
Fallback Limited to 3 Additional Tries	285
Advanced Capabilities	286
Switching between MultiBoot Configuration Memory Types	286
MultiBoot Design Examples	287

Chapter 15: Protecting FPGA Designs

Basic FPGA Hardware-Level Security Options	289
Spartan-3 and Spartan-3E Security Levels	290
Extended Spartan-3A Family Security Levels	290
Setting the Security Level in the Bitstream	290
ISE Software Project Navigator	290
BitGen Command-Line Utility	292
Approaches to Design Security	292
Security Bits	293
Encryption	293
Authentication	293
Extended Spartan-3A Family Unique Device Identifier (Device DNA)	294
Identifier Value	294
Operation	294
Interface Timing	295
Identifier Memory Specifications	296
Extending Identifier Length	296
JTAG Access to Device Identifier	297
iMPACT Access to Device Identifier	297
Authentication Design Examples	297
Extended Spartan-3A Family FPGA: Imprinting or Watermarking the Configuration PROM with Device DNA	298
Spartan-3E FPGA: Leveraging Security Features in Select Commodity Flash PROMs	299
Extended Spartan-3A Family FPGA: Authenticating a Downloaded Design	302
Authenticating any FPGA Design Using External Secure PROM	303
Handling Failed Authentications	304
No Functionality	304
Limited Functionality	304
Full Functionality with Time Out	304
Active Defense	305
Authentication Algorithm	305
Manufacturing Logistics	305
Additional Uses of Authentication and Device ID	306
Protecting Intellectual Property (IP)	306
Code and Data Security	306
U.S. Legal Protection of FPGA Configuration Bitstream Programs	306
Additional Information	308

Chapter 16: Configuration CRC

CRC Checking during Configuration	309
Spartan-3 and Spartan-3E Configuration CRC Errors	309
Configuration CRC Enabled by Default	309
Possible CRC Escapes	309
Extended Spartan-3A Family Configuration CRC Errors and Configuration Watchdog Timer	310
Robust CMOS Configuration Latches (CCLs)	310
Post-Configuration CRC (Extended Spartan-3A Family Only)	310
Overview	310
Continuous CRC Checking Until Configuration, JTAG or Suspend Event	312
Clock Source	312
CRC Checking Time	312
Behavior when CRC Error Occurs	312
Verifying CRC Error Behavior	313
Preparing an Application to Use the Post-Configuration CRC Feature	313
Example User Constraints File (UCF)	313
CONFIG Constraints	314
Bitstream Generator Options	314
Design Considerations	315
Techniques to Check Distributed and Block RAM Contents	315

Chapter 17: Configuration Details

Configuration Memory Frames	317
Configuration Control Logic	318
Packet Types	318
Type 1 Packet	318
Type 2 Packet	319
Configuration Registers	320
Command Register (CMD)	321
Control Register (CTL)	322
Frame Address Registers (FAR_MAJ and FAR_MIN)	323
Status Register (STAT)	323
Configuration Options Registers (COR1 and COR2)	324
Suspend Options Register (PWRDN_REG)	326
Housecleaning Options Register (HC_OPT_REG)	327
Multiple Frame Write Register (MFWR)	327
Bitstream Composition	327
Default Initial Configuration Process	328

Chapter 18: Readback

Preparing a Design for Readback	331
Readback Command Sequences	332
Accessing Configuration Registers through the SelectMAP Interface	332
Configuration Register Read Procedure (SelectMAP)	333
Configuration Memory Read Procedure (SelectMAP)	334
Accessing Configuration Registers through the JTAG Interface	336
Configuration Register Read Procedure - JTAG	337
Configuration Memory Read Procedure (1149.1 JTAG)	338
Configuration Memory Read Procedure (1532 JTAG)	341

Verifying Readback Data	342
Readback Capture	346

Schedule of Figures

Chapter 1: Overview and Design Considerations

<i>Figure 1-1: Spartan-3 Generation Self-Loading (Master) Configuration Modes</i>	29
<i>Figure 1-2: Spartan-3 Generation Downloaded (Slave) Configuration Modes</i>	30
<i>Figure 1-3: Spartan-3 Generation Configuration Daisy-Chain Options</i>	33
<i>Figure 1-4: Extended Spartan-3A Family Type 1 Packet Header</i>	42
<i>Figure 1-5: Setting Bitstream Generator Options from ISE Project Navigator</i>	42
<i>Figure 1-6: Bitstream Generator General Options</i>	43
<i>Figure 1-7: Bitstream Generator Configuration Options</i>	44
<i>Figure 1-8: Bitstream Generator Startup Options</i>	45
<i>Figure 1-9: Bitstream Generator Readback Options</i>	46

Chapter 2: Configuration Pins and Behavior during Configuration

<i>Figure 2-1: DONE and INIT_B Synchronize Daisy-Chain or Broadside Configurations</i>	55
<i>Figure 2-2: Differences between Spartan-3/3E and Extended Spartan-3A Family FPGAs for Master Configuration Modes</i>	58
<i>Figure 2-3: Point-to-Point: Master CCLK Output Drives Single Clock Load</i>	59
<i>Figure 2-4: Multi-Drop: Master CCLK Output Drives Two Clock Inputs</i>	59
<i>Figure 2-5: Star Topology Is Not Recommended</i>	60
<i>Figure 2-6: Using Resistor Divider Network to Meet VCCO_2 POR Threshold</i>	74
<i>Figure 2-7: Stylized Configuration Waveforms Showing When Dual-Purpose Pins Become Active</i>	77

Chapter 3: Master Serial Mode

<i>Figure 3-1: Master Serial Mode Using Platform Flash PROM (Spartan-3E or Spartan-3A/3A DSP FPGA, V_{CCAUX} = 2.5V)</i>	80
<i>Figure 3-2: Master Serial Mode Using Platform Flash PROM (Extended Spartan-3A Family FPGA, V_{CCAUX} = 3.3V)</i>	81
<i>Figure 3-3: Master Serial Mode Using Platform Flash PROM (Spartan-3 FPGA)</i>	82
<i>Figure 3-4: Multi-FPGA Daisy-Chain Configuration Using Xilinx Platform Flash PROM</i>	87
<i>Figure 3-5: Multiple, Identical FPGAs Programmed with the Same Bitstream</i>	87
<i>Figure 3-6: Various Methods to Use Platform Flash PROM after Configuration</i>	89
<i>Figure 3-7: Double-click Generate PROM, ACE or JTAG File</i>	91
<i>Figure 3-8: Prepare a PROM File</i>	91
<i>Figure 3-9: Set Options for Xilinx Platform Flash PROM</i>	92
<i>Figure 3-10: Select Platform Flash PROM</i>	93
<i>Figure 3-11: Review PROM Formatting Settings</i>	94
<i>Figure 3-12: Add FPGA Configuration Bitstream File(s)</i>	94

Figure 3-13: Generate PROM File	95
Figure 3-14: Program Platform Flash PROM using JTAG	96
Figure 3-15: iMPACT Automatically Detects JTAG Chain	96
Figure 3-16: Bypass Programming the FPGA	97
Figure 3-17: Select the Platform Flash Programming File	97
Figure 3-18: Program the Platform Flash PROM	98
Figure 3-19: PROM Programming Options	98

Chapter 4: Master SPI Mode

Figure 4-1: SPI Flash Configuration Interface for M25P-compatible Devices	102
Figure 4-2: SPI Flash Configuration Interface for Atmel DataFlash Devices	103
Figure 4-3: SPI Flash PROM/FPGA Power-On Timing if 3.3V Supply is Last in Power-On Sequence	113
Figure 4-4: If Not Using SPI after Configuration, Drive CSO_B Pin High	116
Figure 4-5: Using the SPI Flash Interface After Configuration	117
Figure 4-6: Daisy Chaining from SPI Flash Mode	119
Figure 4-7: Multiple, Identical FPGAs Programmed with the Same Bitstream	120
Figure 4-8: Using FPGA's JTAG Test Chain to Program Attached SPI Flash	125
Figure 4-9: Double-click Generate PROM, ACE or JTAG File	126
Figure 4-10: Prepare a PROM File	127
Figure 4-11: Set Options for a 3rd-Party SPI PROM	128
Figure 4-12: Select SPI PROM Density	128
Figure 4-13: Review PROM Formatting Settings	129
Figure 4-14: Add FPGA Configuration Bitstream File(s)	129
Figure 4-15: Generate PROM File	130
Figure 4-16: iMPACT Supports Direct Programming for SPI Serial Flash Memories	131
Figure 4-17: Select a Previously-formatted PROM File	132
Figure 4-18: Select a Supported SPI Flash Memory Device	132
Figure 4-19: Directly Program Supported SPI Flash PROM	133
Figure 4-20: SPI PROM Programming Options	133
Figure 4-21: Indirect Programming Method Uses JTAG	135
Figure 4-22: Select the FPGA Bitstream File and Enable SPI Programming	135
Figure 4-23: iMPACT Uses the JTAG Clock Input TCK for Startup Clock when Programming via JTAG	136
Figure 4-24: Select the SPI PROM Programming File	136
Figure 4-25: Select SPI Flash PROM Type	136
Figure 4-26: Bypass the Platform Flash PROM	137
Figure 4-27: iMPACT Presents JTAG Chain, Shows Attached Flash PROM	137
Figure 4-28: SPI PROM Programming Options	138
Figure 4-29: Waveforms for Serial Peripheral Interface (SPI) Configuration	139
Figure 4-30: Multi-Package Layout for the M25Pxx Family on Spartan-3E Starter Kit	142

Chapter 5: Master BPI Mode

<i>Figure 5-1: Spartan-3E FPGA Configured from Parallel NOR Flash</i>	144
<i>Figure 5-2: Extended Spartan-3A Family FPGA Configured from Parallel NOR Flash</i>	145
<i>Figure 5-3: FPGA Supports x8 Interface before Configuration and Optional x16 Interface after Configuration</i>	154
<i>Figure 5-4: Parallel Daisy Chain using BPI Mode</i>	156
<i>Figure 5-5: Serial Daisy Chains are Only Available for Extended Spartan-3A Family BPI Mode</i>	158
<i>Figure 5-6: Master BPI Mode Using Xilinx Parallel Platform Flash PROMs (XCFxxP)</i>	159
<i>Figure 5-7: Double-click Generate PROM, ACE or JTAG File</i>	161
<i>Figure 5-8: Prepare a PROM File</i>	162
<i>Figure 5-9: Set Options for a Generic Parallel PROM</i>	163
<i>Figure 5-10: Select Parallel PROM Size and Configuration Style</i>	164
<i>Figure 5-11: Select FPGA Bitstream Files</i>	165
<i>Figure 5-12: Generate Parallel PROM File</i>	166
<i>Figure 5-13: Parallel NOR Flash PROM/FPGA Power-On Timing if 3.3V Supply is Last in Power-On Sequence</i>	168
<i>Figure 5-14: BPI Configuration Timing Waveform (Spartan-3E BPI Down mode shown)</i>	169

Chapter 6: Master Parallel Mode

Chapter 7: Slave Parallel (SelectMAP) Mode

<i>Figure 7-1: Slave Parallel Mode (Spartan-3E and Extended Spartan-3A Family FPGAs)</i>	176
<i>Figure 7-2: Slave Parallel Mode (Spartan-3 FPGAs)</i>	177
<i>Figure 7-3: Slave Parallel Daisy Chain for Spartan-3E/Extended Spartan-3A Family FPGAs</i>	181
<i>Figure 7-4: Slave Parallel Daisy Chain Using Any Modern Xilinx FPGA</i>	182
<i>Figure 7-5: SelectMAP Continuous Data Loading</i>	184
<i>Figure 7-6: SelectMAP Non-Continuous Data Loading with Controlled CSI_B</i>	185
<i>Figure 7-7: Non-Continuous SelectMAP Data Loading with Controlled CCLK</i>	186
<i>Figure 7-8: Configuration Abort Sequence</i>	187
<i>Figure 7-9: Readback Abort Sequence</i>	188
<i>Figure 7-10: Byte Swapping Example</i>	192

Chapter 8: Slave Serial Mode

<i>Figure 8-1: Slave Serial Configuration</i>	194
---	-----

Chapter 9: JTAG Configuration Mode and Boundary-Scan

<i>Figure 9-1: JTAG Configuration Interface</i>	198
<i>Figure 9-2: Typical JTAG (IEEE 1149.1) Architecture</i>	200

<i>Figure 9-3: Test Access Port (TAP) State Machine</i>	201
<i>Figure 9-4: Boundary-Scan Logic per I/O Pin</i>	204
<i>Figure 9-5: Spartan-3 Generation Boundary-Scan Timing Waveforms</i>	207
<i>Figure 9-6: Target Interface Connector Dimensions and Pin Assignments</i>	208
<i>Figure 9-7: Double-click Configure Device (iMPACT)</i>	209
<i>Figure 9-8: Configure Devices Using JTAG</i>	209
<i>Figure 9-9: iMPACT Automatically Detects Devices on the JTAG Chain</i>	210
<i>Figure 9-10: iMPACT Prompts for FPGA Bitstream</i>	210
<i>Figure 9-11: iMPACT Automatically Adjusts FPGA Startup Clock for JTAG Configuration</i>	211
<i>Figure 9-12: Click Bypass to Skip Platform Flash Programming</i>	211
<i>Figure 9-13: Click Bypass to Skip CPLD Programming</i>	212
<i>Figure 9-14: Double-Click Program to Configure FPGA via JTAG</i>	212
<i>Figure 9-15: FPGA Programming Options</i>	213

Chapter 10: Internal Master SPI Mode

<i>Figure 10-1: Spartan-3AN FPGA using Internal Master SPI Flash Mode</i>	215
<i>Figure 10-2: Spartan-3AN SPI_ACCESS Design Primitive</i>	218
<i>Figure 10-3: Double-click Generate PROM, ACE or JTAG File</i>	220
<i>Figure 10-4: Prepare a PROM File</i>	221
<i>Figure 10-5: Set Options for Spartan-3AN In-System Flash PROM</i>	222
<i>Figure 10-6: Select a Spartan-3AN FPGA</i>	223
<i>Figure 10-7: Specify the FPGA Configuration Bitstream(s)</i>	224
<i>Figure 10-8: Review Spartan-3AN In-System Memory Formatting Settings</i>	225
<i>Figure 10-9: Add FPGA Configuration Bitstream File(s)</i>	225
<i>Figure 10-10: iMPACT View of the Spartan-3AN In-System Flash Memory</i>	226
<i>Figure 10-11: Generate the Spartan-3AN In-System Flash File</i>	226

Chapter 11: Configuration Bitstream Generator (BitGen) Settings

Chapter 12: Sequence of Events

<i>Figure 12-1: Spartan-3 Generation FPGA Configuration Process</i>	239
<i>Figure 12-2: FPGA Wake from Reset</i>	239
<i>Figure 12-3: Extended Spartan-3A Family and Spartan-3E Reset Circuitry (Spartan-3 is similar)</i>	240
<i>Figure 12-4: FPGA Power-Up Timing Waveforms (Master Modes)</i>	242
<i>Figure 12-5: Clear Configuration Memory (Initialization)</i>	243
<i>Figure 12-6: Sample Control Pins (Mode Select, Variant Select)</i>	243
<i>Figure 12-7: Synchronization</i>	244
<i>Figure 12-8: Check Array ID</i>	245
<i>Figure 12-9: Load Configuration Data Frames</i>	247
<i>Figure 12-10: Cyclic Redundancy Check</i>	247
<i>Figure 12-11: Startup Sequence</i>	248

<i>Figure 12-12: Default Start-Up Sequence</i>	249
<i>Figure 12-13: Extended Spartan-3A Family, Spartan-3E FPGA Configuration Logic Conceptual Block Diagram</i>	252

Chapter 13: Configuration-Related Design Primitives

<i>Figure 13-1: BSCAN Primitive for Extended Spartan-3A Family FPGAs</i>	253
<i>Figure 13-2: STARTUP Primitive for Extended Spartan-3A Family and Spartan-3E FPGAs</i>	255
<i>Figure 13-3: CAPTURE Primitive for Extended Spartan-3A Family FPGAs (other families are similar)</i>	256
<i>Figure 13-4: ICAP Primitive (only available on Extended Spartan-3A Family FPGAs)</i>	258
<i>Figure 13-5: DNA_PORT Primitive (only available on Extended Spartan-3A Family FPGAs)</i>	259

Chapter 14: Reconfiguration and MultiBoot

<i>Figure 14-1: Example Spartan-3E MultiBoot Application using 1Mbyte Parallel Flash PROM</i>	263
<i>Figure 14-2: Prepare a MultiBoot PROM Image</i>	264
<i>Figure 14-3: Select a PROM Supporting MultiBoot for Spartan-3E FPGAs</i>	265
<i>Figure 14-4: Select the Configuration Direction of the First MultiBoot Image</i>	266
<i>Figure 14-5: Select a PROM Size and Add It to the Design</i>	266
<i>Figure 14-6: Confirm the PROM Settings</i>	267
<i>Figure 14-7: Select the First MultiBoot Configuration Image</i>	267
<i>Figure 14-8: Select the Second MultiBoot Configuration Image</i>	268
<i>Figure 14-9: Generate the PROM File Using the Specified Parameters</i>	269
<i>Figure 14-10: PROMGen Report File (*.prm)</i>	270
<i>Figure 14-11: Spartan-3A MultiBoot Example using XC3S700A and SPI Flash</i>	279
<i>Figure 14-12: Prepare a MultiBoot PROM Image</i>	279
<i>Figure 14-13: Select a PROM Supporting MultiBoot for Extended Spartan-3A Family FPGAs</i>	280
<i>Figure 14-14: SPI or Parallel Flash PROMs are Supported</i>	281
<i>Figure 14-15: Enter a PROM Density and Specify MultiBoot Image Start Locations</i>	281
<i>Figure 14-16: Confirm PROM Settings</i>	282
<i>Figure 14-17: Select the First (Default) Configuration Image</i>	283
<i>Figure 14-18: Select the Third MultiBoot Configuration Image</i>	283
<i>Figure 14-19: Generate the PROM File Using the Specified Parameters</i>	284
<i>Figure 14-20: Extended Spartan-3A Family MultiBoot Configuration Mode Control</i>	286

Chapter 15: Protecting FPGA Designs

<i>Figure 15-1: Setting Bitstream Generator Options from ISE Project Navigator</i>	291
<i>Figure 15-2: Bitstream Generator Security Options</i>	291
<i>Figure 15-3: Extended Spartan-3A Family DNA_PORT Design Primitive</i>	294
<i>Figure 15-4: DNA_PORT Operation</i>	295

<i>Figure 15-5: Possible Options for DIN Input</i>	297
<i>Figure 15-6: Extended Spartan-3A Family FPGA Configures Normally</i>	298
<i>Figure 15-7: Extended Spartan-3A Family FPGA Authenticates the PROM Image Against Device DNA</i>	298
<i>Figure 15-8: Authentication Fails Using an Unauthorized Copy</i>	299
<i>Figure 15-9: Spartan-3E FPGA Authentication Example using Commodity Flash PROM with Identifier</i>	300
<i>Figure 15-10: Spartan-3E FPGA Authenticates the PROM Image Against the PROM's Device ID</i>	301
<i>Figure 15-11: Intelligent Host Downloads a Spartan-3A Bitstream</i>	302
<i>Figure 15-12: Host Reads Device DNA, Generates Authentication Value</i>	302
<i>Figure 15-13: Host Writes Authentication Value to Enable FPGA Application</i>	302
<i>Figure 15-14: FPGA Authentication Using SHA-1 Secure EEPROM</i>	303
<i>Figure 15-15: Extended Spartan-3A Family Device DNA Used as a Key to Protect Embedded Processing Applications</i>	306

Chapter 16: Configuration CRC

<i>Figure 16-1: Conceptual Overview of Post-Configuration CRC Calculator</i>	311
<i>Figure 16-2: UCF Constraints for Post-Configuration CRC</i>	313
<i>Figure 16-3: Checking Block RAM Contents Using Simple Parity</i>	315

Chapter 17: Configuration Details

Chapter 18: Readback

<i>Figure 18-1: Changing the SelectMAP Port from Write to Read Control</i>	332
<i>Figure 18-2: SelectMAP Status Register Read</i>	334
<i>Figure 18-3: IEEE 1532 JTAG Readback Flow</i>	342
<i>Figure 18-4: Readback Data Stream</i>	343
<i>Figure 18-5: Comparing Readback Data Using the .msd and .rbd Files</i>	344
<i>Figure 18-6: Comparing Readback Data Using the .msk and .bit Files</i>	345
<i>Figure 18-7: Extended Spartan-3A Library Primitive</i>	346
<i>Figure 18-8: Logic Allocation File Format</i>	347

Schedule of Tables

Chapter 1: Overview and Design Considerations

<i>Table 1-1: Spartan-3 Generation Self-Loading Configuration Modes and Memory Sources</i>	29
<i>Table 1-2: Spartan-3 Generation Configuration Options</i>	31
<i>Table 1-3: PROM Families and Footprint Compatible Package Migration</i>	37
<i>Table 1-4: Number of Bits in an Uncompressed FPGA Bitstream Image</i>	39
<i>Table 1-5: Configuration Data Frames</i>	40
<i>Table 1-6: Maximum CCLK Frequency When Using Compressed Bitstream</i>	41
<i>Table 1-7: Command Line to Review Bitstream Generator Options per Family</i>	46

Chapter 2: Configuration Pins and Behavior during Configuration

<i>Table 2-1: Mode Pin Settings and Associated FPGA Configuration Mode by Family</i> . .	50
<i>Table 2-2: M[2:0] Mode Pin Differences between Spartan-3 Generation FPGAs</i>	50
<i>Table 2-3: Default Post-Configuration Behavior of M[2:0] Pin</i>	52
<i>Table 2-4: DriveDone Bitstream Generator Option</i>	53
<i>Table 2-5: DonePin Bitstream Generator Option</i>	53
<i>Table 2-6: Interaction between DriveDone and DonePin Bitstream Generator Options for DONE Pin</i>	54
<i>Table 2-7: PROG_B Operation</i>	56
<i>Table 2-8: CCLK Differences between Spartan-3 Generation FPGA Families</i>	57
<i>Table 2-9: Pins with Dedicated Pull-Up Resistors during Configuration (All Spartan-3 Generation FPGAs)</i>	63
<i>Table 2-10: Pins with Dedicated Pull-Up Resistors during Configuration (Extended Spartan-3A Family FPGAs Only)</i>	63
<i>Table 2-11: Pins with Dedicated Pull-Up Resistors during Configuration (Spartan-3 FPGA Family Only)</i>	64
<i>Table 2-12: Pull-Up Resistor during Configuration Control Input</i>	64
<i>Table 2-13: Pull-Up Resistor Ranges by Spartan-3 Generation Family</i>	65
<i>Table 2-14: Recommended External Pull-Up or Pull-down Resistor Values to Define Input Values during Configuration</i>	65
<i>Table 2-15: Spartan-3 Generation Configuration Pins, Associated Modes, and Function</i>	66
<i>Table 2-16: Extended Spartan-3A Family FPGAs: Pin Behavior during Configuration</i> .	70
<i>Table 2-17: Spartan-3E FPGAs: Pin Behavior during Configuration</i>	71
<i>Table 2-18: Pin Behavior during Configuration for Spartan-3 FPGA Family</i>	72
<i>Table 2-19: Default I/O Standard Setting During Configuration</i>	72
<i>Table 2-20: Supported Configuration Interface Voltages</i>	73
<i>Table 2-21: Pull-up or Pull-down Values for HSWAP, M[2:0], and VS[2:0]</i>	76

Chapter 3: Master Serial Mode

<i>Table 3-1: Spartan-3E/Spartan-3A/3A DSP FPGA Connections</i>	80
<i>Table 3-2: Spartan-3 FPGA Connections to Platform Flash PROM</i>	82
<i>Table 3-3: Master Serial Configuration Mode Connections</i>	83
<i>Table 3-4: Number of Bits to Program a Spartan-3 Generation FPGA and Smallest Platform Flash PROM</i>	85
<i>Table 3-5: Maximum ConfigRate Settings Using Platform Flash (Serial Mode, Commercial Range)</i>	86
<i>Table 3-6: Xilinx Platform Flash Production Programmers</i>	99

Chapter 4: Master SPI Mode

<i>Table 4-1: Master SPI Mode Support within Spartan-3 Generation FPGAs</i>	104
<i>Table 4-2: SPI Flash Memory Devices Officially Supported by Xilinx and Programmed Using iMPACT</i>	105
<i>Table 4-3: SPI Serial Flash PROMs Supported by iMPACT</i>	105
<i>Table 4-4: SPI Read Commands Supported by Spartan-3 Generation FPGAs</i>	106
<i>Table 4-5: Other SPI Flash Memory Devices With Data Sheet Compatibility (Unverified by Xilinx, Unsupported in iMPACT)</i>	107
<i>Table 4-6: Number of Bits to Program an Extended Spartan-3A family or Spartan-3E FPGA and Smallest SPI Flash PROM</i>	108
<i>Table 4-7: Example SPI Flash PROM Connections and Pin Naming</i>	108
<i>Table 4-8: Serial Peripheral Interface (SPI) Connections</i>	110
<i>Table 4-9: Example Minimum Power-On to Select Times for Various SPI Flash PROMs</i>	112
<i>Table 4-10: Spartan-3E and Extended Spartan-3A Family DSP Power-On Reset Timing and Thresholds</i>	114
<i>Table 4-11: FPGA ConfigRate Setting and Corresponding SPI Flash PROM Clock-to-Output Requirements (TV)</i>	115
<i>Table 4-12: Summary of SPI Flash PROM Programming Options</i>	121
<i>Table 4-13: Xilinx Download Header Signal Description for In-System SPI Flash PROM Programming</i>	123
<i>Table 4-14: PROM Generator Command Options</i>	130
<i>Table 4-15: FPGA Timing Symbols for Serial Peripheral Interface (SPI) Configuration Mode</i>	140
<i>Table 4-16: Configuration Timing Requirements for Attached SPI Serial Flash</i>	141

Chapter 5: Master BPI Mode

<i>Table 5-1: BPI Configuration Mode Differences between Spartan-3 Generation FPGA Families</i>	146
<i>Table 5-2: BPI Addressing Control</i>	147
<i>Table 5-3: Byte-Wide Peripheral Interface (BPI) Connections</i>	148
<i>Table 5-4: Example Compatible Parallel NOR Flash Families</i>	151
<i>Table 5-5: Number of Bits to Program an Extended Spartan-3A Family or Spartan-3E FPGA and Smallest Usable Parallel PROM</i>	151
<i>Table 5-6: Maximum ConfigRate Settings for Parallel Flash PROMs</i>	

(Commercial Temperature Range)	152
Table 5-7: FPGA Connections to Flash PROM with IO15/A-1 Pin	155
Table 5-8: Maximum ConfigRate Settings Using Parallel Platform Flash	160
Table 5-9: Example Minimum Power-On to Setup Times for Various Parallel NOR Flash PROMs	167
Table 5-10: Configuration Timing Requirements for Attached Parallel NOR Flash ..	170
Table 5-11: Spartan-3E: Shared BPI Configuration Pins and Global Buffer Input Pins	172

Chapter 6: Master Parallel Mode

Chapter 7: Slave Parallel (SelectMAP) Mode

Table 7-1: Slave Parallel (SelectMAP) Function Overview	178
Table 7-2: Slave Parallel Mode Connections	179
Table 7-3: ABORT Status Word	189
Table 7-4: Example ABORT Sequence	189
Table 7-5: Pins Affected by Persist	190
Table 7-6: Bit Ordering for SelectMAP 8-Bit Mode	191

Chapter 8: Slave Serial Mode

Table 8-1: Slave Serial Mode Connections	196
--	-----

Chapter 9: JTAG Configuration Mode and Boundary-Scan

Table 9-1: JTAG Cable Interface and Current-Limiting Resistor Requirements	199
Table 9-2: TAP Controller States	201
Table 9-3: Spartan-3 Generation TAP Controller Pins	202
Table 9-4: Spartan-3 Generation JTAG Registers	203
Table 9-5: Spartan-3 Generation Boundary-Scan Instructions	205
Table 9-6: Instruction Capture Values	206
Table 9-7: Mating Connectors for 2 mm pitch, 14 Conductor Ribbon Cable	208

Chapter 10: Internal Master SPI Mode

Table 10-1: Number of Bits to Program a Spartan-3AN FPGA and Internal SPI Flash Memory	216
Table 10-2: Spartan-3AN FPGA Supported Variant Select (VS[2:0]) Options	217
Table 10-3: Locations of Default Bitstream and Second MultiBoot Bitstream	224
Table 10-4: PROM Generator Command Options	227
Table 10-5: Spartan-3AN PROMGen Size Settings	227
Table 10-6: BPM Microsystems Programmers Supporting Spartan-3AN FPGAs	228
Table 10-7: BPM Microsystems Socket Modules and Software for Spartan-3AN FPGAs	229

Chapter 11: Configuration Bitstream Generator (BitGen) Settings

<i>Table 11-1: Command Line to Review Bitstream Generator Options per Family</i>	231
<i>Table 11-2: Spartan-3 Generation Bitstream Generator (BitGen) Options</i>	232

Chapter 12: Sequence of Events

<i>Table 12-1: Power-On Reset Threshold Voltages</i>	241
<i>Table 12-2: FPGA Power-Up Timing Specifications</i>	242
<i>Table 12-3: Spartan-3 Generation FPGA Synchronization Word</i>	244
<i>Table 12-4: Spartan-3 Generation FPGA Array ID Codes</i>	246
<i>Table 12-5: STAT Register</i>	247
<i>Table 12-6: User-Selectable Cycle of Startup Events</i>	248
<i>Table 12-7: Default BitGen Sequence of Startup Events</i>	248
<i>Table 12-8: Signals Relating to Startup Sequencer</i>	251

Chapter 13: Configuration-Related Design Primitives

<i>Table 13-1: BSCAN Primitives by FPGA Family</i>	254
<i>Table 13-2: BSCAN Primitive Connections</i>	254
<i>Table 13-3: STARTUP Primitives by FPGA Family</i>	255
<i>Table 13-4: STARTUP Primitive Connections</i>	256
<i>Table 13-5: CAPTURE Primitive by FPGA Family</i>	257
<i>Table 13-6: CAPTURE Primitive Connections</i>	257
<i>Table 13-7: CAPTURE Attributes</i>	257
<i>Table 13-8: ICAP_SPARTAN3A Primitive Connections</i>	258
<i>Table 13-9: DNA_PORT Primitive Connections</i>	260
<i>Table 13-10: DNA_PORT Attributes</i>	260

Chapter 14: Reconfiguration and MultiBoot

<i>Table 14-1: MultiBoot Options on Spartan-3 Generation FPGA Families</i>	262
<i>Table 14-2: Command Sequence to Initiate MultiBoot from a Preloaded Address</i>	274
<i>Table 14-3: Command Sequence to Initiate MultiBoot to a Specified Address</i>	275
<i>Table 14-4: GENERAL1 Register Definition</i>	276
<i>Table 14-5: GENERAL2 Register Definition for BPI Mode Options</i>	277
<i>Table 14-6: GENERAL2 Register Definition for SPI Mode Options</i>	277
<i>Table 14-7: CMD Register Definition</i>	277
<i>Table 14-8: MODE_REG Bit Options</i>	278

Chapter 15: Protecting FPGA Designs

<i>Table 15-1: Spartan-3 and Spartan-3E Security Levels</i>	290
<i>Table 15-2: Extended Spartan-3A Family BitGen Security Levels</i>	290
<i>Table 15-3: Relation between ISE Project Navigator and BitGen Options</i>	292
<i>Table 15-4: Programmable Logic Security Options Compared</i>	292
<i>Table 15-5: DNA_PORT Operations</i>	295

<i>Table 15-6: DNA_PORT Interface Timing</i>	295
<i>Table 15-7: Identifier Memory Characteristics</i>	296
<i>Table 15-8: Example Flash PROMs with Embedded Unique Identifiers</i>	299

Chapter 16: Configuration CRC

<i>Table 16-1: Post-Configuration CRC CONFIG Constraints</i>	314
<i>Table 16-2: Post-Configuration CRC Bitstream Generator Options</i>	314

Chapter 17: Configuration Details

<i>Table 17-1: Extended Spartan-3A Frame Count and Length</i>	318
<i>Table 17-2: Type 1 Packet Header Format</i>	319
<i>Table 17-3: Type 1 Packet Data Format</i>	319
<i>Table 17-4: Opcode Format</i>	319
<i>Table 17-5: Type 2 Packet Header</i>	319
<i>Table 17-6: Type 2 Packet Word Count 1</i>	319
<i>Table 17-7: Type 2 Packet Word Count 2</i>	320
<i>Table 17-8: Type 2 Packet Data</i>	320
<i>Table 17-9: Configuration Registers</i>	320
<i>Table 17-10: Command Register Codes</i>	321
<i>Table 17-11: Control Register Description</i>	322
<i>Table 17-12: Frame Address Register Description</i>	323
<i>Table 17-13: Status Register Description</i>	324
<i>Table 17-14: Configuration Options Register Description</i>	324
<i>Table 17-15: Suspend Options Register Description</i>	326
<i>Table 17-16: Housecleaning Options Register Description</i>	327
<i>Table 17-17: Configuration Sequence</i>	328

Chapter 18: Readback

<i>Table 18-1: Status Register Readback Command Sequence (SelectMAP)</i>	333
<i>Table 18-2: Shutdown Readback Command Sequence (SelectMAP)</i>	335
<i>Table 18-3: Shifting in the JTAG CFG_IN and CFG_OUT Instructions</i>	336
<i>Table 18-4: Status Register Readback Command Sequence (JTAG)</i>	337
<i>Table 18-5: Shutdown Readback Command Sequence (JTAG)</i>	339
<i>Table 18-6: Readback Files</i>	342
<i>Table 18-7: Capture Signals</i>	346

Overview and Design Considerations

Xilinx® Field Programmable Gate Arrays (FPGAs) are highly flexible, reprogrammable logic devices that leverage advanced CMOS manufacturing technologies, similar to other industry-leading processors and processor peripherals. Like processors and peripherals, Xilinx FPGAs are fully user programmable. For FPGAs, the program is called a *configuration bitstream*, which defines the FPGA's functionality. The bitstream loads into the FPGA at system power-up or upon demand by the system.

The process whereby the defining data is loaded or programmed into the FPGA is called *configuration*. Configuration is designed to be flexible to accommodate different application needs and, wherever possible, to leverage existing system resources to minimize system costs.

Similar to microprocessors, Xilinx FPGAs optionally load or boot themselves automatically from an external nonvolatile memory device. Alternatively, similar to microprocessor peripherals, Spartan-3 generation FPGAs can be downloaded or programmed by an external “smart agent”, such as a microprocessor, DSP processor, microcontroller, PC, or board tester. In either case, the configuration data path is either serial to minimize pin requirements or byte-wide for maximum performance or for easier interfaces to processors or to byte-wide Flash memory.

Similar to both processors and processor peripherals, Xilinx FPGAs can be reprogrammed, in system, on demand, an unlimited number of times. After configuration, the FPGA configuration bitstream is stored in highly robust CMOS configuration latches (CCLs). Although CCLs are reprogrammable like SRAM memory, CCLs are designed primarily for data integrity, not for performance. The data stored in CCLs is written only during configuration and remains static unless changed by another configuration event.

This user guide provides both an introduction to the configuration options available to the user, and a detailed description of the configuration logic. This user guide includes the Extended Spartan-3A family, which includes the Spartan-3A, Spartan-3AN, and Spartan-3A DSP platforms. The user guide also includes the earlier Spartan-3 and Spartan-3E families. Together, these families are sometimes referred to as the Spartan-3 generation. Most basic configuration features are similar between the families, and differences are noted where necessary.

Design Considerations

Before starting a new FPGA design, spend a few minutes to consider which FPGA configuration mode best matches your system requirements. Each configuration mode dedicates certain FPGA pins and may borrow others. Similarly, the configuration mode may place voltage restrictions on some FPGA I/O banks.

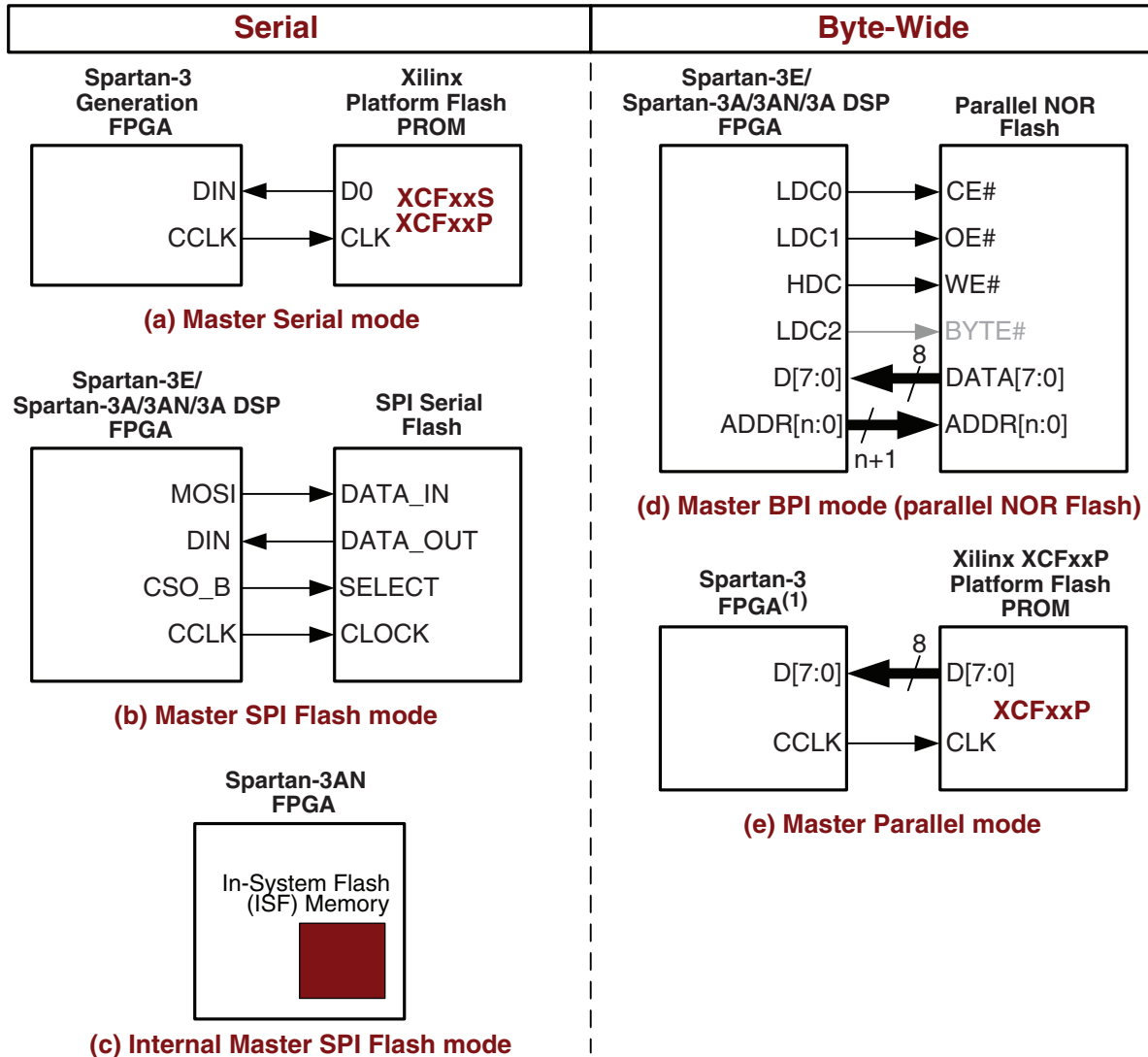
If you have already selected an FPGA configuration mode, feel free to jump to the relevant section in the user guide. Otherwise, please evaluate the following design considerations to understand the options available.

Will the FPGA load configuration data itself from external or internal memory or will an external processor/microcontroller download configuration data?

Spartan-3 generation FPGAs are designed for maximum flexibility. The FPGA either automatically loads itself with configuration data, like a processor, or alternatively, another external intelligent device like a processor or microcontroller can download the configuration data. It is your choice and [Table 1-2](#) summarizes the available options.

The self-loading FPGA configuration modes, generically called *Master* modes, are available with either a serial or byte-wide data path as shown in [Figure 1-1](#). The Master modes leverage various types of nonvolatile memories to store the FPGA's configuration information, as shown in [Table 1-1](#). In Master mode, the FPGA's configuration bitstream typically resides in nonvolatile memory on the same board, generally external to the FPGA. The FPGA internally generates a configuration clock signal called CCLK and the FPGA controls the configuration process.

Spartan-3AN FPGAs optionally configure from internal In-System Flash (ISF) memory, as shown in [Figure 1-1c](#). In this mode, the configuration memory and the control and data signals are inside the package. Spartan-3AN FPGAs also optionally support all the other Spartan-3A FPGA configuration modes, as well.



Notes:

- 1. Remaining Spartan-3 Generation FPGAs support XCFxxP Platform Flash PROMs via Master BPI mode.

UG332_c1_01_052207

Figure 1-1: Spartan-3 Generation Self-Loading (Master) Configuration Modes

Table 1-1: Spartan-3 Generation Self-Loading Configuration Modes and Memory Sources

External Memory	Information on FPGA Configuration Mode	Supported Spartan-3 Generation Families
Xilinx Platform Flash PROM (either XCFxxS or XCFxxP PROMs)	Chapter 3, "Master Serial Mode"	All
Xilinx Platform Flash PROM (XCFxxP PROMs only)	Chapter 6, "Master Parallel Mode"	Primarily Spartan-3 FPGAs, but possible in Spartan-3E/3A/3AN/3A DSP FPGAs using BPI mode or Slave Parallel mode

Table 1-1: Spartan-3 Generation Self-Loading Configuration Modes and Memory Sources (Cont'd)

External Memory	Information on FPGA Configuration Mode	Supported Spartan-3 Generation Families
Commodity Parallel NOR Flash PROM	Chapter 5, "Master BPI Mode"	Spartan-3E, Extended Spartan-3A family FPGAs
Commodity SPI Serial Flash PROM	Chapter 4, "Master SPI Mode"	Spartan-3E, Extended Spartan-3A family FPGAs

The downloaded FPGA configuration modes, generically called Slave modes, are also available with either a serial or byte-wide data path. In Slave mode, an external "intelligent agent" such as a processor, microcontroller, DSP processor, or tester downloads the configuration image into the FPGA, as shown in Figure 1-2. The advantage of the Slave configuration modes is that the FPGA bitstream can reside just about anywhere in the overall system. The bitstream could reside in Flash, on board, along with the host processor's code. It could reside on a hard disk. It could originate somewhere over a network connection. The possibilities are nearly endless.

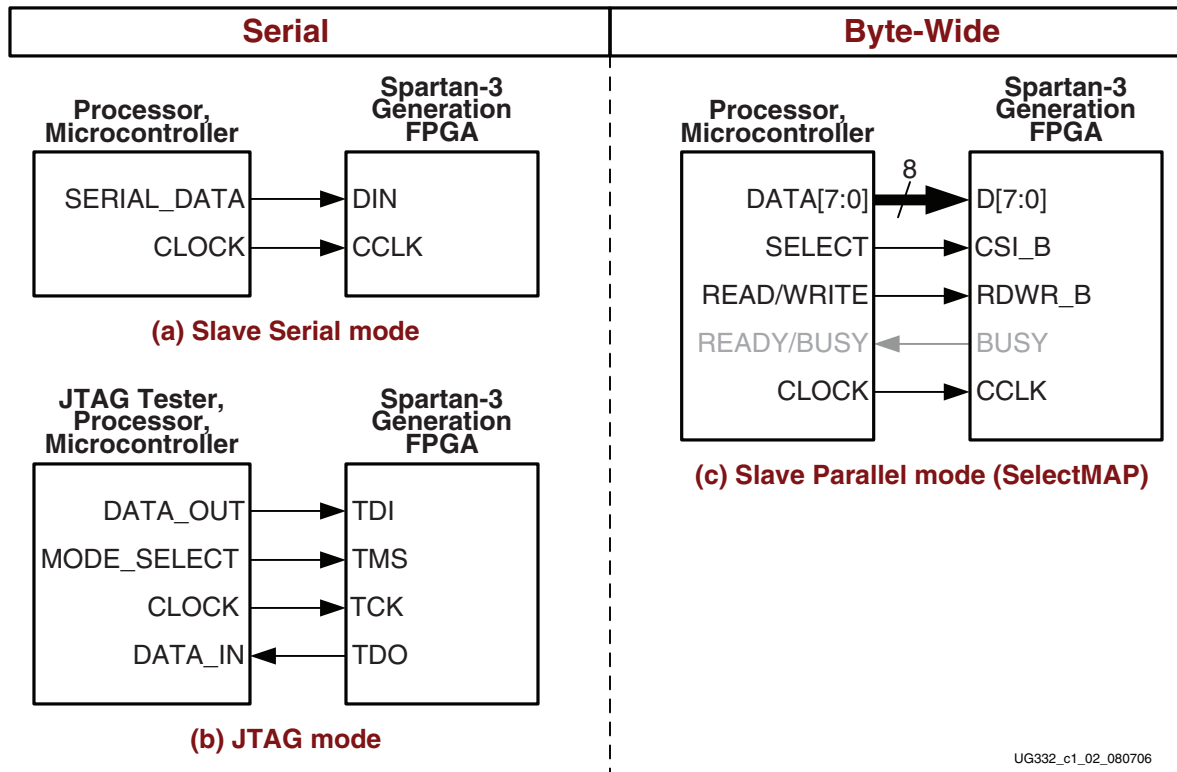


Figure 1-2: Spartan-3 Generation Downloaded (Slave) Configuration Modes

The Slave Parallel mode, also called SelectMAP mode in other FPGA architectures, is essentially a simple byte-wide processor peripheral interface, including a chip-select input and a read/write control input. The Slave Serial mode is extremely simple, consisting only of a clock and serial data input.

The four-wire JTAG interface is common on many board testers and debugging hardware. In fact, the Xilinx programming cables for Spartan-3 generation FPGAs, listed below, use the JTAG interface for prototype download and debugging. Regardless of which configuration mode is ultimately used in the application, it is best to also include a JTAG

configuration path for easy design development. Also see “Programming Cables and Headers,” page 207.

- **Platform Cable USB II**
<http://www.xilinx.com/products/devkits/HW-USB-II-G.htm>
- **Parallel Cable IV**
<http://www.xilinx.com/products/devkits/HW-PC4.htm>

Table 1-2: Spartan-3 Generation Configuration Options

	Master Serial	SPI	BPI	Master Parallel	Internal Master SPI	Slave Parallel	Slave Serial	JTAG
Spartan-3 Generation Families	All	Spartan-3A Spartan-3AN Spartan-3A DSP Spartan-3E	Spartan-3A Spartan-3AN Spartan-3A DSP Spartan-3E	Spartan-3 only	Spartan-3AN only	All	All	All
M[2:0] mode pin settings	<0:0:0>	<0:0:1>	<0:1:0>=Up Spartan-3E only: <0:1:1>=Down	<0:1:1:	<0:1:1>	<1:1:0>	<1:1:1>	<1:0:1>
Data width	Serial	Serial	Byte-wide	Byte-wide	Serial	Byte-wide	Serial	Serial
Configuration memory source	Xilinx Platform Flash	Commodity SPI serial Flash	Commodity parallel NOR Flash or Xilinx parallel Platform Flash	Xilinx parallel Platform Flash , etc.	Internal In-System Flash (ISF) memory	Any source via micro-controller, CPU, Xilinx parallel Platform Flash , etc.	Any source via micro-controller, CPU, Xilinx Platform Flash , etc.	Any source via micro-controller, CPU, System ACE™ CF , etc.
Clock source	Internal oscillator					External clock signal applied on CCLK pin		External clock on TCK pin
Total I/O pins borrowed during configuration	8	13	46	12	7	21	8	0
Configuration mode for downstream daisy-chained FPGAs	Slave Serial	Slave Serial	Slave Parallel Extended Spartan-3A family only: Slave Serial	Slave Serial	Not Supported	Slave Parallel or Memory Mapped	Slave Serial	JTAG
Stand-alone FPGA applications (no external download host)	✓	✓	✓	✓	✓	Possible using XCFxxP Platform Flash, which optionally generates CCLK	Possible using XCFxxP Platform Flash, which optionally generates CCLK	

Does the application use a single FPGA or multiple FPGAs?

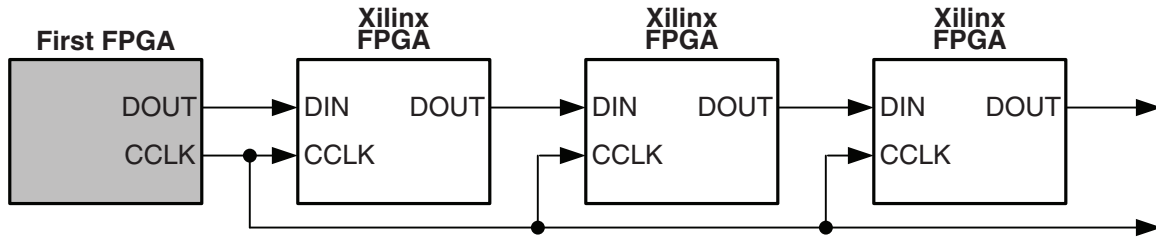
Most Spartan-3 generation FPGA applications use a single FPGA. However, some applications require multiple FPGAs for increased logic density or I/O. Obviously, each FPGA in a multi-FPGA design could have its own separate configuration source. However, using a configuration daisy-chain, multiple FPGAs share a single configuration source. Daisy-chaining reduces system costs and simplifies programming and logistics.

The most common style is a serial daisy chain, illustrated in [Figure 1-3a](#). Generally, the first device in the chain may use any one of the configuration modes, except JTAG mode. When the first device finishes loading its configuration bitstream, it passes data to the downstream FPGAs via its DOUT serial data output pin.

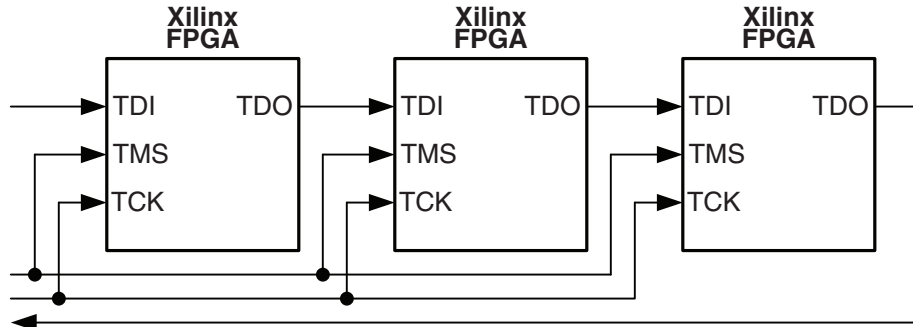
The JTAG interface also supports multi-FPGA configuration as shown in [Figure 1-3b](#). The TDO serial data output is connected to the TDI serial data input of the next device in the chain. The mode select input, TMS, and the clock input, TCK, are common to all devices in the JTAG chain. The TDO serial data output of the last device in the chain feeds back to the JTAG connector.

Lastly, [Figure 1-3c](#) shows a parallel daisy chain. All of the FPGA connections are common, except for the chip select inputs, which are unique per FPGA.

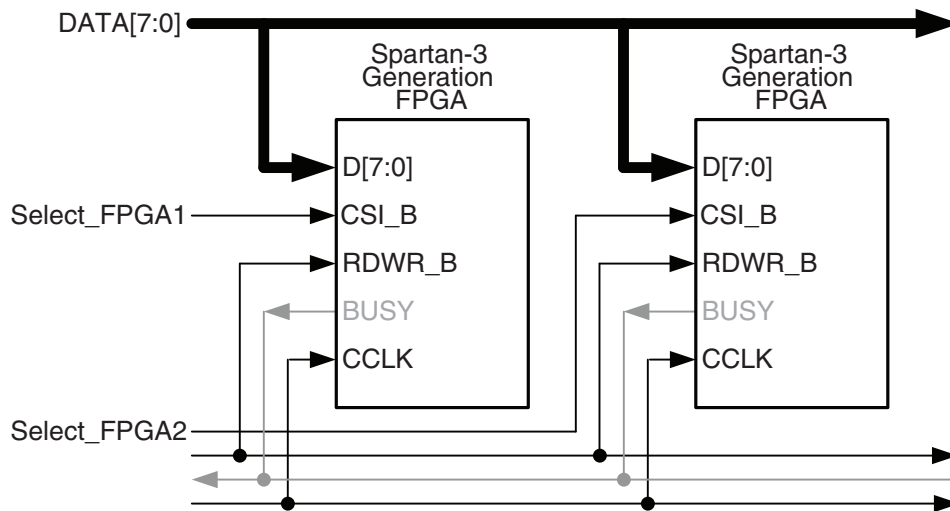
Caution! The Spartan-3AN FPGA family does not support configuration daisy-chains when configured using the Internal Master SPI mode.



(a) Serial Daisy Chain (using Slave Serial mode)



(b) Multi-FPGA configuration JTAG mode



(c) Parallel Daisy Chain (using Slave Parallel mode)

UG332_c1_03_080706

Figure 1-3: Spartan-3 Generation Configuration Daisy-Chain Options

Is the “easiest possible” configuration solution the more important consideration?

Let's face it, in some applications, the easiest solution is the best solution. The best solution for these applications is either Internal Master SPI mode supported only by Spartan-3AN FPGAs or Master Serial mode using a Xilinx Platform Flash PROM, which is available for any Spartan-3 generation FPGA. These solutions use the fewest FPGA pins, have flexible I/O voltage support, and is fully supported by iMPACT, the Xilinx JTAG-based programming software.

Will the application require a nonvolatile FPGA?

A nonvolatile FPGA provides convenience, security, and a cost-effective single-chip solution. Multiple FPGA bitstream options can be stored in a single device, plus any nonvolatile data needed for the system. Up to 11 Mb of user data is available in the Spartan-3AN family. For more information on using this memory space, see [UG333, Spartan-3AN FPGA In-System Flash user guide](#).

Is the “lowest cost” solution the more important consideration?

For cost-sensitive applications, obviously the lowest-cost configuration solution is best. However, which option is lowest cost? The answer depends on your specific application.

- Is there spare external nonvolatile memory already available in the system in which to store the FPGA configuration bitstream(s)? The bitstream image can be stored in system memory, stored on a hard drive, or even downloaded remotely over a network connection. If so, consider one of the downloaded modes, [Master Parallel Mode](#), [Slave Serial Mode](#), or [JTAG Configuration Mode and Boundary-Scan](#).
- Is there a way to consolidate the nonvolatile memory required in the application? For example, can the FPGA configuration bitstream(s) be stored with any processor code for the board? If the processor is a [MicroBlaze™](#) soft processor core embedded in the FPGA, the FPGA configuration data and the MicroBlaze code can easily share the same nonvolatile memory device.
- Spartan-3A and Spartan-3E FPGAs optionally configure from commodity SPI serial Flash and parallel NOR Flash memories. Because these memories have common footprints and multiple suppliers, they may have lower pricing due to the highly-competitive marketplace.

Is “fastest possible configuration time” the more important consideration?

Some applications require that the logic be operational within a short time. Certain FPGA configuration modes and methods are faster than others. The configuration time includes the initialization time plus the configuration time. Configuration time depends on the size of the device and speed of the configuration logic. For example, an XC3S1400A programming at 10 MHz will require 4755296 bits / 10 MHz or approximately 500 ms.

- At the same clock frequency, parallel configuration modes are inherently faster than the serial modes, since they program 8 bits at a time.
- Configuring a single FPGA is inherently faster than configuring multiple FPGAs in a daisy-chain. In a multi-FPGA design where configuration speed is a concern, configure each FPGA separately and in parallel.
- In Master modes, the FPGA internally generates the CCLK configuration clock signal. By default, the CCLK frequency starts out low but can be increased using the ConfigRate bitstream option. The maximum supported CCLK frequency setting depends on the read specifications for the attached nonvolatile memory. A faster memory may allow for faster configuration.
- Furthermore, in Master modes, the FPGA's CCLK output frequency varies with process, voltage, and temperature. The fastest guaranteed configuration rate depends on the slowest guaranteed CCLK frequency as shown in the respective data sheet. If an external clock is available on the board, it is also possible to configure the FPGA in a Slave mode while still using an attached nonvolatile memory.

Will the FPGA or FPGAs be loaded with a single configuration image or loaded with multiple images?

In most FPGA applications, the FPGA is loaded only when the system is powered on.

However, some applications reload the FPGA multiple times while the system is operating, with different FPGA bitstreams for different functions. For example, the FPGA may be loaded with one bitstream to implement a power-on self-test, followed by a second bitstream with the final application. In many test equipment applications, the FPGA is loaded with different bitstreams to execute hardware-assisted tests. In this way, one smaller FPGA can implement the equivalent functionality of a larger ASIC or gate array device.

The downloaded or Slave configuration modes easily support reloading the FPGA with multiple images. However, this is also possible on Spartan-3E and Extended Spartan-3A family FPGAs using the MultiBoot feature.

See [Chapter 14, “Reconfiguration and MultiBoot”](#) for more information.

What I/O voltages are required in the end application?

The chosen FPGA configuration mode places some constraints on the FPGA application, specifically the I/O voltage allowed on the FPGA's configuration banks.

For example, the SPI or BPI modes leverage third-party Flash memory components that are usually 3.3V-only devices. This then requires that the I/O voltage on the bank or banks attached to the memory also be 3.3V. In most applications, this is not an issue.

However, if a voltage other than 3.3V is required, specifically 2.5V, consider using a Xilinx Platform Flash PROM, which supports a range of output voltages via a separate supply on the Platform Flash PROM.

Will the FPGA application need to store nonvolatile data?

Some FPGA applications store data in external nonvolatile memory. Spartan-3E or Spartan-3A/3A DSP FPGAs provide some useful enhancements for these applications.

- Spartan-3E and Spartan-3A/3A DSP FPGAs can configure directly from external commodity serial or parallel Flash PROMs.
- The Flash PROM address, data, and control pins are only borrowed by the FPGA during configuration. After configuration, the FPGA has full read/write control over these pins.
- The FPGA configuration bitstreams and the application's nonvolatile data can share the same PROM, reducing overall system cost.

See [Chapter 4, “Master SPI Mode”](#) or [Chapter 5, “Master BPI Mode”](#) for additional information.

Should the FPGA I/O pins be pulled High via resistors during configuration?

Some of the FPGA pins used during configuration have dedicated pull-up resistors during configuration. However, the majority of user-I/O pins have optional pull-up resistors that can be enabled during the configuration process. During configuration, a single control line determines whether the pull-up resistors are enabled or disabled. The name of the control pin varies by Spartan-3 generation family. On Extended Spartan-3A family FPGAs, this pin is called [PUDC_B](#) (pull-up during configuration, active Low) and on Spartan-3E FPGAs, this same pin is called [HSWAP](#), short for hot-swap. On Spartan-3 FPGAs, the same pin is called [HSWAP_EN](#).

Why enable the pull-up resistors during configuration? Floating signal levels are problematic in CMOS logic systems. Other logic components in the system may require a valid input level from the FPGA. The internal pull-up resistors generate a logic High level on each pin. Generally, a device driving signals into the FPGA can overcome the pull-up resistor. Similarly, an individual pin can be pulled down using an appropriately-sized external pull-down resistor.

Why disable pull-up resistors during configuration? In hot-swap or hot-insertion applications, the pull-up resistors provide a potential current path to the I/O power rail. Turning off the pull-up resistors disables this potential path. However, then external pull-up or pull-down resistors may be required on each individual I/O pin.

See “Pull-Up Resistors During Configuration,” page 62 for additional information.

Does the application target a specific FPGA density or should it support migrating to other FPGA densities in the same package footprint?

The package footprint and pinouts for Xilinx Spartan-3 generation FPGAs are designed to allow migration between different densities within a specific family. For example, three different Spartan-3E FPGAs support the identical package footprint when using the 320-ball fine-pitch ball grid array package (FG320). As shown in [Table 1-4](#), the smallest of devices, the XC3S500E, requires approximately 2.2 Mbits for configuration. The largest of these devices, the XC3S1600E, requires 5.7 Mbits for configuration.

Likewise, an FPGA application may store other nonvolatile data in the Flash memory, requiring a larger storage device.

To support design migration between device densities, allow sufficient configuration memory to cover the largest device in the targeted package. In the example provided above, allow up to 5.7 Mbits for configuration. This allows the application to use any Spartan-3E FPGA available in the FG320 package.

In downloaded applications, simply reserve enough space in memory for the largest anticipated, uncompressed FPGA bitstream.

In self-loaded applications, use a PROM footprint and the associated FPGA configuration mode to facilitate easy migration. [Table 1-3](#) provides example migration options using different FPGA configuration modes, different PROM families, and different package options. For example, Xilinx Platform Flash provides excellent migration between 1 to 4 Mbits using the XCFxxS serial family and between 8 to 32 Mbits using the XCFxxP parallel family. If an application spans between the two, use two separate footprints, one for each Platform Flash sub family. Be aware that the XCFxxP Flash family requires a 1.8V core supply voltage input while the XCFxxS requires 3.3V. Both families provide 3.3V I/O.

Table 1-3: PROM Families and Footprint Compatible Package Migration

Config. Mode	PROM Family	Package Option	PROM Density in Bits/Associated Part Numbers						
			1M	2M	4M	8M	16M	32M	64M
Master Serial Mode	XCFxxS serial Platform Flash	VO20	XCF01S	XCF02S	XCF04S	—	—	—	—
	XCFxxP parallel Platform Flash	VO48	—	—	—	XCF08P	XCF16P	XCF32P	—
		FS48	—	—	—	XCF08P	XCF16P	XCF32P	—
Master SPI Mode	ST-compatible SPI Flash (Multi-Package Footprint)	8SOIC JEDEC 8SOIC EIAJ 16SOIC 8MLP	Part number varies by vendor.						
	Atmel AT45DBxxxD SPI Flash (Multi-Package Footprint)	8SOIC JEDEC 8SOIC EIAJ 8CASON	'011D	'021D	'041D	'081D	'161D	'321D	'642D
Master BPI Mode	x8 Parallel NOR Flash	40-pin TSOP	Part number varies by vendor.					—	—
	x8/x16 Parallel NOR Flash	48-pin TSOP	—	Part number varies by vendor.					
	x8 or x8/x16 Parallel NOR Flash	48-ball FBGA	—	Part number varies by vendor.					

Notes:

- Platform Flash PROMs also work in Master BPI mode, as described in [“Using Xilinx Platform Flash PROMs with Master BPI Mode,”](#) page 158.

The SPI serial Flash vendors offer a wider migration range but do require a multi-package footprint. For example, the Atmel DataFlash SPI serial Flash family spans the range of 1 Mbit to 64 Mbit using a single footprint that accommodates the JEDEC and EIAJ versions of the 8-pin SOIC package along with the 8-connector CASON package. The STMicro (Numonyx) SPI serial Flash has uses a different footprint that uses a combined 8-pin and 16-pin SOIC footprint and is also compatible with devices from multiple SPI Flash vendors. See [“Multi-Package Layout,”](#) page 141.

Similarly, parallel Flash supports a wide density range in a common, multi-vendor package footprint.

What is the anticipated production lifetime for the end product?

Consider whether your application has a relatively short or a relatively long production lifetime. Commodity memories generally have a shorter production lifetime than the proprietary Xilinx Platform Flash PROMs. For example, if building an industrial application that will be manufactured for five years or more, then Xilinx Platform Flash PROMs may provide better long-term availability. Similarly, the In-System Flash (ISF) memory on Spartan-3AN comes integrated with the FPGA.

Products with shorter production lifetimes may benefit from the multi-vendor pricing and multi-sourcing of commodity memories.

Do you want to protect your FPGA bitstream against unauthorized duplication?

Like processor code, the bitstream that defines the FPGA's functionality loads into the FPGA during power-on. Consequently, this means that an unscrupulous company can capture the bitstream and create an unauthorized copy of the design.

Like processors, there are multiple techniques to protect the FPGA bitstream and any intellectual property (IP) cores embedded in the FPGA. The most powerful of these is called *authentication* and is more fully described in [Chapter 15, "Protecting FPGA Designs."](#)

Do you want to load multiple FPGAs with the same configuration bitstream?

Generally, there is one configuration bitstream image per FPGA in a system. As shown in [Figure 1-3](#), multiple, different FPGA bitstream images can share a single configuration PROM by leveraging a configuration daisy-chain. However, what if all the FPGAs in the application have the same part number and use the same bitstream? Fortunately, in this case, only a single bitstream image is required. An alternative solution, called a ganged or broad-side configuration, loads multiple, similar FPGAs with the same bitstream. See [Figure 3-5, page 87](#) or [Figure 4-7, page 120](#) for an example.

Caution! The Spartan-3AN FPGA family does not support configuration daisy-chains when configured using the Internal Master SPI mode.

Will the FPGA be used in a PCI™ application?

The PCI™ Local Bus Specification, Revision 3.0 ("the PCI specification") defines a number of power and reset requirements. These requirements, when considered in an FPGA implementation, create several challenges that must be addressed for long term reliability and broad interoperability. See [XAPP457](#), "Powering and Configuring Spartan-3 generation FPGAs in Compliant PCI Applications", for more details.

Where to go for debugging support

This user guide attempts to make FPGA configuration easy and straight forward. Should problems occur, please visit the interactive Configuration Debug Guide to you through the configuration debugging process.

- **Configuration Debug Guide**
http://survey.xilinx.com/ss/wsb.dll/Xilinx/Configuration_Debug_Guide.htm

FPGA Configuration Bitstream Sizes

By default, FPGA configuration images are uncompressed. In an uncompressed FPGA bitstream, the size of the image is constant regardless of the complexity of the underlying FPGA application. Put another way, a single inverter requires the same bitstream size as a complex MPEG4 encoder implemented in the same FPGA array.

Uncompressed Bitstream Image Size

[Table 1-4](#) provides the number of bits in an uncompressed FPGA bitstream for each specific part number of the Spartan-3 generation.

Table 1-4: Number of Bits in an Uncompressed FPGA Bitstream Image

Spartan-3 Generation FPGA Family	FPGA Part Number	Number of Configuration Bits
Spartan-3A/3AN FPGA	XC3S50A/AN	437,312
	XC3S200A/AN	1,196,128
	XC3S400A/AN	1,886,560
	XC3S700A/AN	2,732,640
	XC3S1400A/AN	4,755,296
Spartan-3A DSP	XC3SD1800A	8,197,280
	XC3SD3400A	11,718,304
Spartan-3E FPGA	XC3S100E	581,344
	XC3S250E	1,353,728
	XC3S500E	2,270,208
	XC3S1200E	3,841,184
	XC3S1600E	5,969,696
Spartan-3 FPGA	XC3S50	439,264
	XC3S200	1,047,616
	XC3S400	1,699,136
	XC3S1000	3,223,488
	XC3S1500	5,214,784
	XC3S2000	7,673,024
	XC3S4000	11,316,864
	XC3S5000	13,271,936

Bitstream Format

The typical FPGA user does not need a bit-level understanding of the configuration stream. However, for the purpose of understanding configuration options and for debugging, an overview of the bitstream format is helpful. For more details, see the chapter [Configuration Details](#) and [XAPP452: Spartan-3 Advanced Configuration Architecture](#).

Synchronization Word

Embedded at the beginning of an FPGA configuration bitstream is a special synchronization word. The synchronization word alerts the FPGA to upcoming configuration data and aligns the configuration data with the internal configuration logic. Any data on the configuration input pins prior to synchronization is ignored. Because the synchronization word is automatically added by the Xilinx bitstream generation software, this step is transparent in most applications. The length and contents of the synchronization word differ between the Extended Spartan-3A family FPGA families and the Spartan-3 and Spartan-3E FPGA families, as outlined in [Table 12-3](#).

Array ID

Next the array IDCODE is embedded in the bitstream so that the FPGA can check that it matches its internal array ID. This prevents the FPGA from mistakenly attempting to load configuration data intended for a different FPGA array. For example, the array ID check prevents an XC3S1000 from being configured with an XC3S200 bitstream. [Table 12-4](#) shows the array ID codes.

Data Frames

Next is the internal configuration memory, partitioned into segments called "data frames." The configuration memory can be visualized as a rectangular array of bits. The bits are grouped into vertical frames that are one-bit wide and extend from the top of the array to the bottom. A frame is the atomic unit of configuration. It is the smallest portion of the configuration memory that can be written to or read from. The number and size of frames varies with device size (see [Table 1-5](#)). The total number of configuration bits for a particular device is calculated by multiplying the number of frames by the number of bits per frame, and then adding the total number of bits needed to perform the configuration register writes.

Table 1-5: Configuration Data Frames

FPGA Family	FPGA Part Number	Number of Frames	Frame Length in Bits
Spartan-3A/3AN FPGA	XC3S50A/AN	367	1,184
	XC3S200A/AN	540	2,208
	XC3S400A/AN	692	2,720
	XC3S700A/AN	844	3,232
	XC3S1400A/AN	996	4,768
Spartan-3A DSP	XC3SD1800A	1,414	5,792
	XC3SD3400A	1,718	6,816
Spartan-3E FPGA	XC3S100E	368	1,568
	XC3S250E	577	2,336
	XC3S500E	729	3,104
	XC3S1200E	958	4,000
	XC3S1600E	1,186	5,024
Spartan-3 FPGA	XC3S50	368	1,184
	XC3S200	615	1,696
	XC3S400	767	2,208
	XC3S1000	995	3,232
	XC3S1500	1,223	4,384
	XC3S2000	1,451	5,280
	XC3S4000	1,793	6,304
	XC3S5000	1,945	6,816

CRC

Next is the Cyclic Redundancy Check (CRC) value. As the configuration data frames are loaded, the FPGA calculates a CRC value. After the configuration data frames are loaded, the configuration bitstream issues a Check CRC instruction to the FPGA. If the CRC value calculated by the FPGA does not match the expected CRC value in the bitstream, then the FPGA pulls INIT_B Low and aborts configuration. Refer to “CRC Checking during Configuration,” page 309 for additional information.

Bitstream Compression

By default, FPGA bitstreams are uncompressed. However, Spartan-3 generation FPGAs support basic bitstream compression. The compression is fairly simple, yet effective for some applications. The ISE® bitstream generator software examines the FPGA bitstream for any duplicate configuration data frames. These duplicates occur often in the following situations.

- FPGA designs with unused block RAM or hardware multipliers.
- FPGA design with low logic utilization, *i.e.*, most of the FPGA array is empty.

The ISE software can then generate a compressed FPGA bitstream. When the FPGA configures, the internal configuration controller copies the redundant data frame to multiple locations. Because of the extra processing required by the FPGA configuration controller, the maximum configuration clock frequency is reduced to 20 MHz on Spartan-3 and Spartan-3E FPGAs, as shown in Table 1-6. Extended Spartan-3A family FPGAs support the full CCLK frequency range, even with compressed bitstreams.

Table 1-6: Maximum CCLK Frequency When Using Compressed Bitstream

	Spartan-3 FPGA	Spartan-3E FPGA	Spartan-3A Spartan-3AN Spartan-3A DSP FPGA
Maximum CCLK Frequency When Using Compressed Bitstream	20 MHz	20 MHz	80 MHz

The amount of compression is non-deterministic. Changes to the source FPGA design may cause the size of the compressed bitstream to grow. Sparse, mostly-empty FPGA designs have the greatest overall compression factor. Similarly, FPGA designs with an empty column of block RAM have a high compression factor.

The overall benefits of a compressed bitstream are as follows.

- Smaller memory footprint.
- Faster programming time for nonvolatile memory.

There are two methods to generate a compressed bitstream, from within the ISE Project Navigator or from the command line.

From Project Navigator, check the **Enable BitStream Compression** option, shown as Step 4 in Figure 1-6.

From the command line, add the **-g Compress** option to the BitGen command line.

```
bitgen -g Compress <other options>...
```

Furthermore, the parallel Platform Flash PROMs offer their own compression mechanisms.

Packet Format

A Spartan-3 generation bitstream consists of a specific sequence of writes to the configuration registers. After synchronization, all data, register writes, and frame data are encapsulated in packets. There are two kinds of packets: Type 1 and Type 2. A Type 1 packet consists of two parts: a header and the data. The header (see [Figure 1-4](#)) describes which register is being accessed, whether it is a read or write operation, and the size of the data to follow. The data portion, always immediately following the header, is the number of 32-bit words specified in the header.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Type			Op		Register Address						Word Count				
0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x

1. Type = "001" for Type 1 and "010" for Type 2
2. Op = "10" for Write and "01" for Read

Figure 1-4: Extended Spartan-3A Family Type 1 Packet Header

For information on Spartan-3 FPGA packet formats, see [XAPP452 Spartan-3 Advanced Configuration Architecture](#).

Setting Bitstream Options, Generating an FPGA Bitstream

After specifying and compiling an FPGA design, generate an FPGA bitstream using either the ISE Project Navigator or the bitstream generator command-line utility, BitGen. The specific details of the bitstream options are described throughout this user guide.

ISE Software Project Navigator

[Figure 1-5](#) shows how to set options for the Bitstream Generator from within the ISE Project Navigator window.

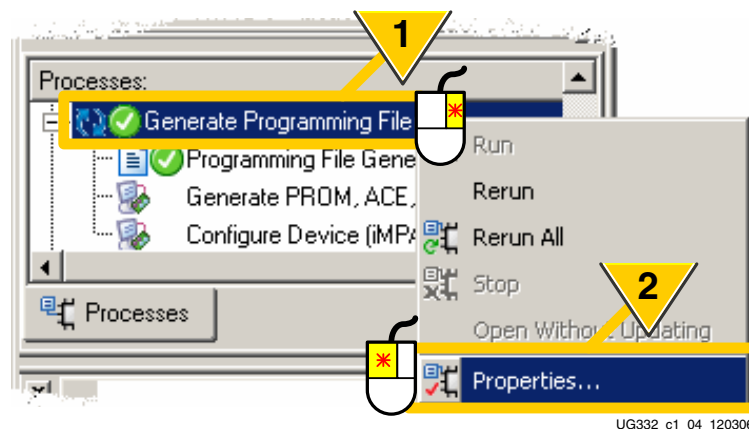
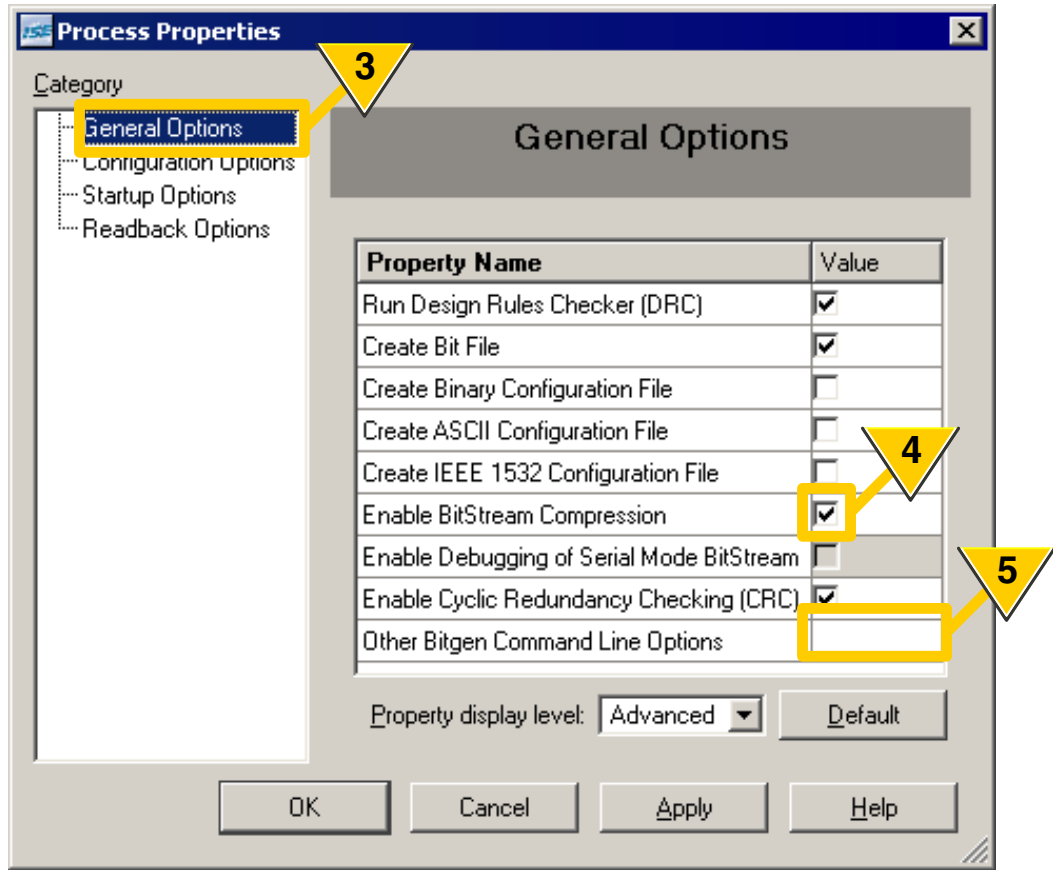


Figure 1-5: Setting Bitstream Generator Options from ISE Project Navigator

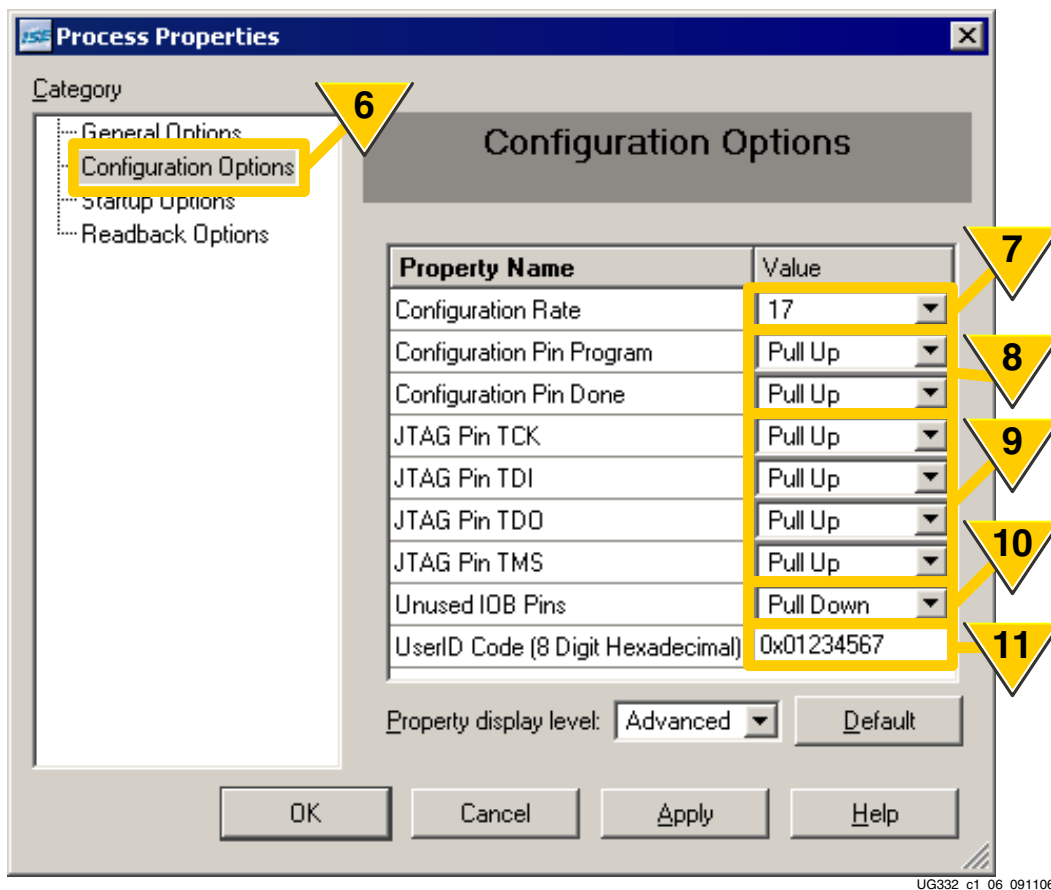
1. Right-click **Generate Programming File**.
2. Click **Properties**.



ug332_C1_05_091106

Figure 1-6: Bitstream Generator General Options

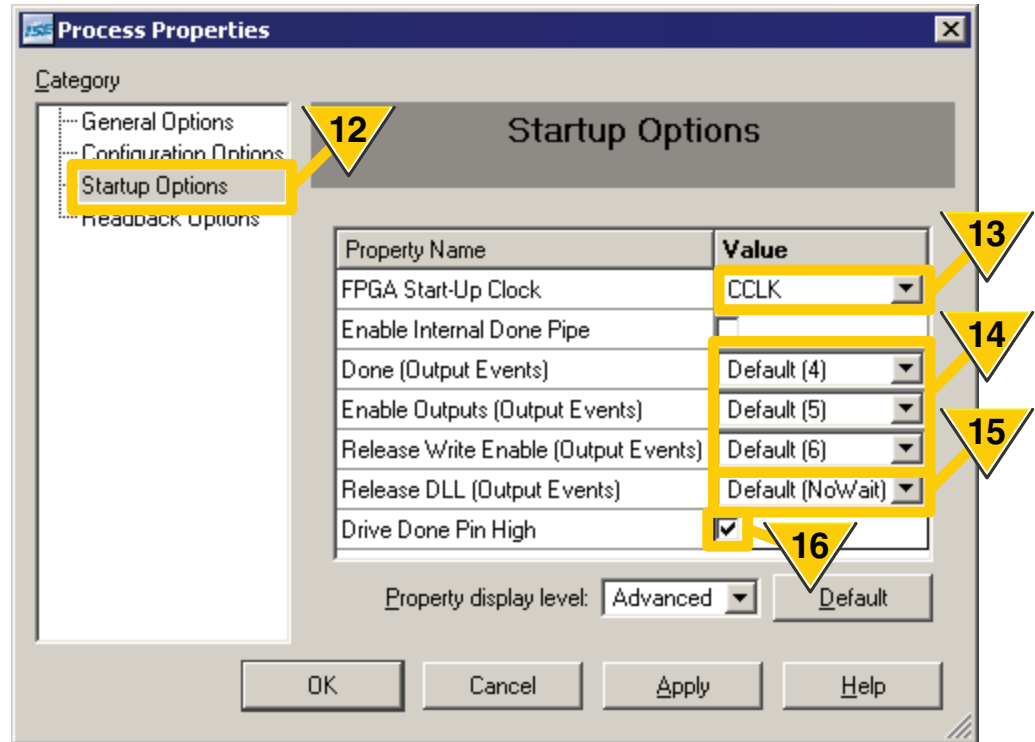
3. Click **General Options**, as shown in [Figure 1-6](#).
4. To compress the FPGA bitstream, check **Enable BitStream Compression**.
5. To enter specific bitstream generator command-line options that are not already supported by the graphical interface, enter the option strings in the space provided.



UG332_c1_06_091106

Figure 1-7: Bitstream Generator Configuration Options

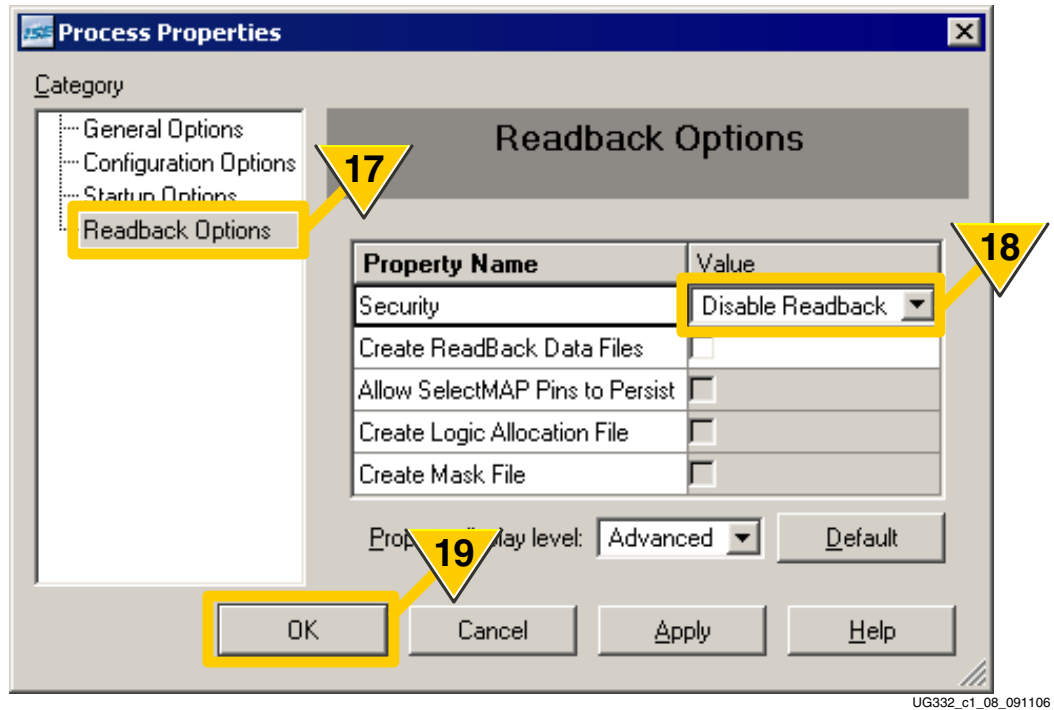
6. Click **Configuration Options**, as shown in [Figure 1-7](#).
7. If using one of the Master configuration modes, set the CCLK **Configuration Rate** frequency. This setting is not used for Slave mode configuration. The specific setting depends on the specific FPGA family, the attached configuration memory, and the configuration mode. Specific values are recommended in later chapters, depending on the speed of the attached memory.
8. The FPGA's DONE and PROG_B (Program) pins each have a dedicated pull-up resistor during configuration. These resistors become optional after configuration. The specific example is from a Spartan®-3E FPGA application. Spartan-3 and Spartan-3A FPGAs have additional options.
9. The FPGA's JTAG pins each have a dedicated pull-up resistor during configuration. These resistors become optional after configuration.
10. By default, unused I/O blocks are configured as inputs with a pull-down resistor. Other options are available. See [UnusedPin](#) bitstream option.
11. Each FPGA bitstream can include an 8-digit hexadecimal (32-bit) identifier that can be read via the FPGA's JTAG port.



UG332_c1_07_120106

Figure 1-8: Bitstream Generator Startup Options

12. Click **Startup Options**, as shown in [Figure 1-8](#).
13. After the FPGA configuration bitstream is loaded into the FPGA, the FPGA enters its Startup phase. The timing of each Startup cycle is controlled by a selectable clock source. See [“Startup Clock Source,”](#) page 250.
14. The Startup phase of FPGA configuration provides six different cycles to synchronize the following startup events. The event can be assigned to a specific cycle or be synchronized to the DONE signal. See [“Startup,”](#) page 248.
 - ◆ The timing of when output drivers are enabled
 - ◆ The timing of when the write-protect lock is removed from writable clocked elements
 - ◆ The timing of when the DONE pin goes active.
15. If the DCM_WAIT=TRUE attribute is set on a Digital Clock Manager (DCM) within the FPGA, the FPGA optionally waits for the Delay-Locked Loop (DLL) within the DCM to lock to the incoming clock signal before finishing configuration. See [“Waiting for DCMs to Lock, DCI to Match,”](#) page 250.
16. The FPGA’s DONE pin can actively drive High after configuration. This option should only be set for single-FPGA applications or for the last FPGA in a multi-FPGA configuration daisy chain. See [“DONE Pin,”](#) page 52.



UG332_c1_08_091106

Figure 1-9: Bitstream Generator Readback Options

17. Click **Readback Options**, as shown in Figure 1-9.
18. By default, FPGA bitstreams can be read back via JTAG. Other options exist to disable FPGA readback. See “Basic FPGA Hardware-Level Security Options,” page 289.
19. Click **OK** when finished.

BitGen Command Line Utility

For designers that prefer command-line processing and to support scripting, the ISE software also provides a command-line bitstream generator utility called BitGen.

For a quick summary of available options for particular FPGA family, type the command shown in Table 1-7 in a DOS box or command window.

Table 1-7: Command Line to Review Bitstream Generator Options per Family

FPGA Family	Command Line
Spartan-3 FPGAs	<code>bitgen -help spartan3</code>
Spartan-3E FPGAs	<code>bitgen -help spartan3e</code>
Spartan-3A FPGAs	<code>bitgen -help spartan3a</code>
Spartan-3AN FPGAs	<code>bitgen -help spartan3an</code>
Spartan-3A DSP FPGAs	<code>bitgen -help spartan3adsp</code>

For complete documentation on the bitstream generator software, please refer to the BitGen chapter of the following software manual.

- **ISE 10.1 Development System Reference Guide**
<http://toolbox.xilinx.com/docsan/xilinx10/books/docs/dev/dev.pdf>

Additional Resources

The following also provide information related to FPGA configuration.

Data Sheets

[DS529](#), Spartan-3A FPGA Family: Data Sheet

[DS557](#), Spartan-3AN FPGA Family Data Sheet

[DS610](#), Spartan-3A DSP FPGA Family: Data Sheet

[DS312](#), Spartan-3E FPGA Family: Complete Data Sheet

[DS099](#), Spartan-3 FPGA Family Data Sheet

Application Notes

Configuration

www.xilinx.com/support/documentation/topicfpgafeaturedesign_configur.htm

Reference Designs for Boards

Spartan-3A/3AN: www.xilinx.com/products/boards/s3astarter/reference_designs.htm

Spartan-3E: www.xilinx.com/products/boards/s3estarter/reference_designs.htm

Spartan-3: www.xilinx.com/products/boards/DO-SPAR3-DK/reference_designs.htm

Configuration Pins and Behavior during Configuration

The FPGA's configuration flexibility means that many pins serve multiple purposes. Some pins are merely borrowed during configuration, only to be released back to the FPGA application as user-defined I/O pins. Other pins are dedicated to configuration. This chapter describes how these various pins behave during the configuration process. During configuration includes the time when the FPGA first powers up, after PROG_B is pulsed Low, or during configuration or re-configuration including MultiBoot.

General Configuration Control Pins

A few pins control the overall FPGA configuration process. These include the following and are similar on all Spartan[®]-3 generation FPGAs. The four-wire JTAG interface is a separate and independent configuration interface discussed primarily in [Chapter 9, "JTAG Configuration Mode and Boundary-Scan"](#).

- The mode select pins, **M[2:0]**, define the configuration mode that the FPGA uses to load its configuration data.
- The **DONE** pin, when High, indicates when the FPGA successfully completed loading its configuration data.
- The program pin, **PROG_B**, initiates the configuration process. The FPGA also automatically initiates configuration on power-up. The JTAG interface has a separate JTAG command to initiate configuration. The PROG_B pin also forces a master reset on the FPGA.
- The configuration clock pin, **CCLK**, defines the timing for the FPGA's configuration process. If the **M[2:0]** mode select pins define a Master mode, then the FPGA internally generates CCLK. If the **M[2:0]** mode select pins define a Slave mode, then CCLK is an input to the FPGA from an external timing reference.
- The **INIT_B** pins performs multiple functions. At the start of configuration, INIT_B goes Low indicating that the FPGA is clearing its internal configuration memory--a process called *housecleaning*. Later, when the FPGA is actively loading its configuration bitstream, INIT_B goes Low if the bitstream fails its CRC check. On Extended Spartan-3A family FPGAs, if so enabled in the FPGA application, the INIT_B pin also potentially signals a post-configuration CRC error.
- During configuration, some pins have built-in pull-up resistors. The remaining pins each have an optional pull-up resistor controlled by a single control input pin. This pin has different names on different architectures as shown in [Table 2-12](#).

Choose a Configuration Mode: M[2:0]

The mode select pins, M[2:0], define the configuration mode that the FPGA uses to load its bitstream, as shown in Table 2-1. The logic levels applied to the mode pins is sampled on the rising edge of INIT_B, immediately after the FPGA completes initializing its internal configuration memory.

Table 2-1: Mode Pin Settings and Associated FPGA Configuration Mode by Family

M[2:0]	FPGA Family			
	Spartan-3	Spartan-3E	Spartan-3A Spartan-3A DSP	Spartan-3AN
<0:0:0>	Master Serial (Platform Flash) Mode			
<0:0:1>	Reserved	Master SPI Mode		
<0:1:0>	Reserved	BPI Up		
<0:1:1>	Master Parallel	BPI Down	Reserved	Internal Master SPI
<1:0:0>	Reserved			
<1:0:1>	JTAG Mode			
<1:1:0>	Slave Parallel Mode			
<1:1:1>	Slave Serial Mode			

M[2:0] Functional Differences between Spartan-3 Generation Families

Table 2-2 summarizes the slight differences in functionality between the Spartan-3 generation families.

Table 2-2: M[2:0] Mode Pin Differences between Spartan-3 Generation FPGAs

	Spartan-3 FPGA	Spartan-3E FPGA	Extended Spartan-3A Family FPGAs
Available as possible user I/O pin after configuration?	No	Yes	Yes
Dedicated internal pull-up resistor during configuration?	Yes	No	Yes
Mechanism to define post-configuration behavior	<i>M2Pin</i> , <i>M1Pin</i> , <i>M0Pin</i> bitstream options	User I/O	User I/O
Input supply voltage	V _{CCAUX}	V _{CCO_2}	V _{CCO_2}
Output supply voltage	N/A	V _{CCO_2}	V _{CCO_2}
Same voltage as other pins in the configuration interface?	Only when interface is at 2.5V	Yes	Yes

Extended Spartan-3A Family and Spartan-3E FPGA Families

On the Extended Spartan-3A and Spartan-3E FPGA families, the [M\[2:0\]](#) mode select pins are borrowed during configuration and become full user I/O after configuration successfully completes. The [M\[2:0\]](#) pins are powered by the V_{CCO_2} supply.

Spartan-3E FPGAs

The Spartan-3E FPGA mode pins do not have dedicated pull-up resistors during configuration. However, these pins have optional pull-up resistors during configuration, controlled by the Spartan-3E [HSWAP](#) pin. If the mode pins are unconnected and if the [HSWAP](#) is Low, then the Spartan-3E FPGA defaults to the Slave Serial configuration mode ([M\[2:0\]](#) = <1:1:1>).

Extended Spartan-3A Family FPGAs

The Extended Spartan-3A family FPGA mode pins have dedicated internal pull-up resistors during configuration, regardless of the [PUDC_B](#) pin. If the mode pins are unconnected, then the Extended Spartan-3A family FPGA defaults to the Slave Serial configuration mode ([M\[2:0\]](#) = <1:1:1>).

Spartan-3 FPGA Family

On the Spartan-3 FPGA family, the [M\[2:0\]](#) mode select pins are dedicated inputs, powered by the V_{CCAUX} supply.

Before and during configuration, the mode pins have a relatively strong internal pull-up resistor to the V_{CCAUX} supply, regardless of the [HSWAP_EN](#) pin.

If the mode pins are unconnected, then the FPGA defaults to the Slave Serial configuration mode ([M\[2:0\]](#) = <1:1:1>). These resistors can be controlled *after* the Spartan-3 FPGA successfully configures using the bitstream generator options [M2Pin](#), [M1Pin](#), and [M0Pin](#). These options define whether a pull-up resistor, pull-down resistor, or no resistor is present on its respective mode pin, M0, M1, or M2. By default, all three pins will have an internal pull-up resistor to V_{CCAUX} .

Defining M[2:0] after Configuration for Minimum Power Consumption

During configuration, the [M\[2:0\]](#) pin may be tied directly to power or ground, tied High or Low using external resistors, or actively driven by an external component. To further minimize power consumption, adjust the post-configuration behavior of the [M\[2:0\]](#) pins so that they match the required configuration setting shown in [Table 2-1, page 50](#), either by defining their value in the FPGA application or by adjusting the associated bitstream options. Essentially, avoid any unnecessary current paths through pull-up or pull-down resistors.

[Table 2-3](#) summarizes the default post-configuration behavior on both Spartan-3 and Spartan-3E/3A/-3AN FPGA families, which have slightly different functionality.

Table 2-3: Default Post-Configuration Behavior of M[2:0] Pin

Spartan-3 FPGAs	Spartan-3E, Extended Spartan-3A Family FPGAs
After configuration, the M[2:0] pins have optional pull-up and pull-down resistors controlled by the M2Pin , M1Pin , and M0Pin bitstream options. Unless changed in the bitstream, all three M[2:0] have pull-up resistors.	After configuration, the M[2:0] pin are available as user-I/O pins. If these pins are not defined in the FPGA application, then these pins are treated as unused I/O pins. The behavior of unused I/O pins is defined by the UnusedPin bitstream option. Unless defined in the FPGA application or changed via the UnusedPin option, all three M[2:0] have internal pull-down resistors.

DONE Pin

The FPGA actively drives the **DONE** pin Low during configuration. When the configuration process successfully completes, the FPGA either actively drives the **DONE** pin High (“**DriveDone**”) or allows the **DONE** pin to float High using either an internal or external pull-up resistor, controlled by the **DonePin** bitstream generator option.

In a multi-FPGA daisy-chain or broadside configuration, the open-drain option permits the **DONE** lines of multiple FPGAs to be tied together, so that the common node transitions High only after all of the FPGAs have successfully completed configuration. Externally holding the open-drain **DONE** pin Low stalls the “**Startup**” sequence.

The **DONE** pin is powered by the V_{CCAUX} supply. The **DONE** pin functionality is common to all Spartan-3 generation FPGAs.

Associated Bitstream Generator (BitGen) Options

The **DONE** pin has various option bits that controls this pin’s behavior during and after configuration. These options are summarized immediately below and described in detail on the next few pages.

- **DriveDone** defines whether the **DONE** pin is an active driver or an open-drain output.
- **DonePin** defines whether or not the **DONE** pin has an internal pull-up resistor.
- **DONE_cycle** defines the **Startup** state where is **DONE** driven High or released to float High.
- **DonePipe** adds an extra pipelining stage before the FPGA actually completes configuration.

DriveDone

The *DriveDone* bitstream generator option, shown in [Table 2-4](#), defines whether the DONE pin has a totem-pole output that actively drives High or acts an open-drain output. If configured as an open-drain output—which is the default behavior—then a pull-up resistor is required to produce a High logic level. The *DonePin* bitstream option controls the pull-up resistor.

Table 2-4: DriveDone Bitstream Generator Option

Setting	Description
No	Default. The DONE pin is an open-drain output. A pull-up resistor to V_{CCAUX} is required. An internal pull-up resistor is available using the <i>DonePin:Pullup</i> bitstream generator option.
Yes	The DONE pin actively drives High when the FPGA completes the configuration process.

Set *DriveDone:Yes* in single-FPGA applications or for the first design in a multi-FPGA design.

This option is set graphically in the “ISE Software Project Navigator,” [page 42](#) by checking **Drive Done Pin High** during Step 16 in [Figure 1-8](#), [page 45](#).

See [Table 2-6](#) for the interaction between *DriveDone* and *DonePin*.

DonePin

The *DonePin* bitstream generator option, shown in [Table 2-5](#), defines whether or not an internal pull-up resistor is present on the DONE pin to pull the pin to V_{CCAUX} . If the pull-up resistor is eliminated, then the DONE pin must be pulled High using an external 300 Ω to 3.3k Ω pull-up resistor.

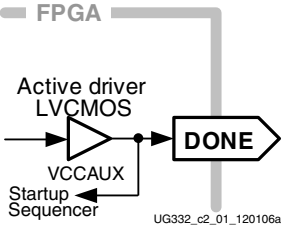
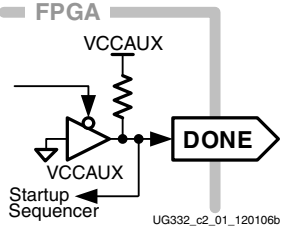
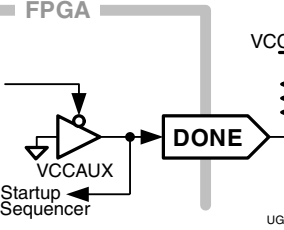
Table 2-5: DonePin Bitstream Generator Option

Setting	Description
Pullup	Default. After configuration, the DONE pin has an internal pull-up resistor to V_{CCAUX} .
Pullnone	There is no internal pull-up resistor on DONE. An external 300 Ω to 3.3k Ω pull-up resistor to V_{CCAUX} is required. The pull-up resistor must be strong enough to pull the DONE pin to a valid High within less than one CCLK cycle.

This option is set graphically in the “ISE Software Project Navigator,” [page 42](#) by selecting the **Configuration Pin Done** setting during Step 8 in [Figure 1-7](#), [page 44](#).

See [Table 2-6](#) for the interaction between *DriveDone* and *DonePin*.

Table 2-6: Interaction between DriveDone and DonePin Bitstream Generator Options for DONE Pin

	DONE Actively Drives	Open-Drain with Internal Pull-up (Default)	Open-Drain with External Pull-Up
Diagram	 <p>UG332_c2_01_120106a</p>	 <p>UG332_c2_01_120106b</p>	 <p>UG332_c2_01_120106c</p>
DriveDone:	Yes	No	No
DonePin:	Pullnone	Pullup	Pullnone
Recommended Usage for Various Configuration Topographies			
Single FPGA	Best	OK	OK but requires external pull-up
Daisy-Chain	Only on first FPGA in the chain	For all down-stream FPGAs in the chain. Also allowed on the first FPGA in the chain.	OK but requires external pull-up
Broadside	Do Not Use!	All FPGAs in a broadside configuration	OK but requires external pull-up

DONE_cycle

The *DONE_cycle* option controls during which cycle the DONE pin is asserted during the Startup sequence, just prior to the completion of a successful configuration. See “Startup,” page 248.

This option is set graphically in the “ISE Software Project Navigator,” page 42 by adjusting the **Done (Output Events)** setting during Step 14 in Figure 1-8, page 45.

DonePipe

The *DonePipe* option is used in a some multi-FPGA applications.

After all DONE pins are released in a multi-FPGA configuration, the DONE pin must transition from Low to High in a single Startup clock cycle (*StartupClk*). If additional time is required for the DONE signal to rise within a single Startup cycle, set the *DonePipe:Yes* bitstream generator option for all devices in the daisy chain or broadside configuration.

Set this option graphically in “ISE Software Project Navigator,” page 42 by checking the **Enable Internal Done Pipe** option box shown in Figure 1-8, page 45.

DONE Synchronizes Multiple FPGAs in a Daisy Chain or Broadside (Ganged) Configuration

In a single-FPGA application, the DONE pin merely indicates when the FPGA successfully configures.

In a multi-FPGA daisy-chain or broadside application, however, the DONE pin also synchronizes the “Startup” sequence of all the FPGAs, ensuring that the FPGAs transition smoothly from the configuration process to the active FPGA application. Figure 2-1 provides a three-FPGA example. In a daisy-chain application, FPGAs of different densities

and architectures are configured in series with different bitstreams. In a broad-side example, multiple, identical FPGAs are simultaneously loaded with the same bitstream.

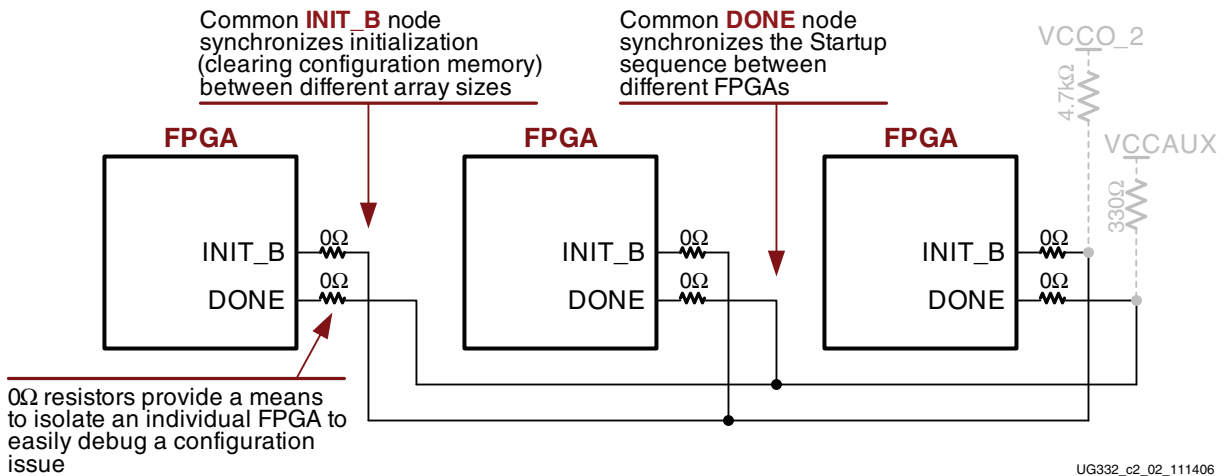


Figure 2-1: **DONE and INIT_B Synchronize Daisy-Chain or Broadside Configurations**

Connect All DONE Pins

Connect the DONE pins for all devices in a multi-FPGA daisy chain or broadside configuration. For debugging purposes, it is often helpful to have a way of disconnecting individual DONE pins from the common DONE signal, so that devices can be individually configured through the serial or JTAG interface. In Figure 2-1, the FPGAs can be disconnected by temporarily remove the 0-ohm resistors on the board. Stake-pin or wire jumpers also work.

DONE Pin Bitstream Generator Options

When generating the bitstream files for each of the FPGAs in the daisy-chain or broadside configuration, set the DONE pin options as indicated in Table 2-6, page 54.

Also, to successfully configure a daisy-chain, the *GTS_cycle* bitstream option must be set to a Startup phase after the *DONE_cycle* setting for all FPGAs in the chain. This is the software default setting. Optionally, set *GTS_cycle:Done*.

Cautions When Mixing Spartan-3A FPGAs with V_{CCAUX} = 3.3V and Other Spartan-3 Generation FPGAs in a Daisy-Chain Configuration

The DONE pin is powered by the FPGA's V_{CCAUX} supply. The V_{CCAUX} voltage on Spartan-3 and Spartan-3E FPGAs is solely 2.5V. For Spartan-3A FPGAs, however, the V_{CCAUX} voltage can be either 2.5V or 3.3V. Spartan-3AN FPGAs require V_{CCAUX} at 3.3V. See "V_{CCAUX} Level".



Caution! In a multi-FPGA configuration that mixes Extended Spartan-3A family and other Xilinx® FPGAs where the Extended Spartan-3A family V_{CCAUX} = 3.3V, check for voltage compatibility on the common DONE node.

Program or Reset FPGA: PROG_B

The PROG_B pin is an asynchronous control input to the FPGA. When Low, the PROG_B pin resets the FPGA, initializing the configuration memory. When released, the PROG_B begins the configuration processes. The initialization process does not start until PROG_B returns High. Asserting PROG_B Low for an extended period delays the configuration process. The various PROG_B functions are outlined in [Table 2-7](#).

At power-up or after a master reset, PROG_B always has a pull-up resistor to V_{CCAUX} , regardless of the “Pull-Up Resistors During Configuration” control input. After configuration, the bitstream generator option *ProgPin* defines whether or not the pull-up resistor remains active. By default, the *ProgPin* option retains the pull-up resistor.

Table 2-7: PROG_B Operation

PROG_B Input	Response
Power-up	Internal “Power-On Reset (POR)” circuit automatically initiates FPGA configuration process.
Low-going pulse 	Initiate (re)configuration process and continue to completion.
Extended Low 	Initiate (re)configuration process and stall process in the “Clear Configuration Memory (Initialization)” step. Configuration is stalled until PROG_B returns High.
1	If the configuration process is started, continue to completion. If configuration process is complete, the FPGA remains configured.

After configuration, hold the PROG_B input High. Any Low-going pulse on PROG_B, lasting 500 ns or longer (300 ns in the Spartan-3 FPGAs), restarts the configuration process.

The PROG_B pin functionality is identical among all Spartan-3 generation FPGAs.

Configuration Clock: CCLK

The configuration clock signal, CCLK, synchronizes the reading or writing of configuration data. In Master modes, CCLK is generated from an internal oscillator within the FPGA. In Slave modes, CCLK is an input, driven by the external device providing the configuration data.

CCLK Differences between Spartan-3 Generation FPGA Families

[Table 2-8](#) summarizes the primary differences between the various Spartan-3 generation FPGA families. On Spartan-3 FPGAs, the CCLK pin is a dedicated function while on the other families, CCLK becomes available as a user-programmable I/O pin after configuration successfully completes.

The CCLK pin is an input-only pin for the Slave Serial and Slave Parallel configuration modes.

Table 2-8: CCLK Differences between Spartan-3 Generation FPGA Families

	Spartan-3	Spartan-3E	Spartan-3A/3AN Spartan-3A DSP
CCLK pin becomes full user-I/O after configuration	No; dedicated pin	Yes	Yes
CCLK pin supply voltage	V _{CCAUX}	VCCO_2	VCCO_2
CCLK pin behavior after configuration	Pull-up or pull-down resistor controlled by <i>CclkPin</i> bitstream option	User I/O	User I/O
CCLK pin directionality during Master mode configuration	I/O	I/O	Output only for improved signal integrity
CCLK frequency options during Master mode configuration (<i>ConfigRate</i>)	3, 6, 12, 25, 50	1, 3, 6, 12, 25, 50	1, 3, 6, 7, 8, 10, 12, 13, 17, 22, 25, 27, 33, 44, 50, 100
CCLK frequency variation	±50% of <i>ConfigRate</i> frequency	Fully characterized. Specified in data sheet.	Fully characterized. Specified in data sheet.

In the Master configuration modes, the FPGA internally generates the CCLK clock source. As shown in [Figure 2-2](#), there are slight differences in the CCLK circuitry between the Spartan-3 / Spartan-3E FPGA families and the Extended Spartan-3A family.

As shown in [Figure 2-2a](#), Spartan-3/3E FPGAs drive the internally-generated CCLK signal to an output. Like the configuration PROM connected to the FPGA, the FPGA's internal configuration logic is clocked by the CCLK signal at the FPGA pin, which simplifies the interface timing. However, any switching noise on the CCLK pin potentially also affects the FPGA. Therefore, treat CCLK as a full bidirectional I/O pin for signal integrity analysis; the FPGA uses the value at the pin to clock internal logic. See "[CCLK Design Considerations](#)," page 58.

As shown in [Figure 2-2b](#), CCLK is strictly an output on Extended Spartan-3A family FPGAs in the Master configuration modes. The FPGA's internal configuration logic is clocked by the internally-generated CCLK signal and is not susceptible to external switching noise. That said, good signal integrity on the CCLK board trace is a good design practice.

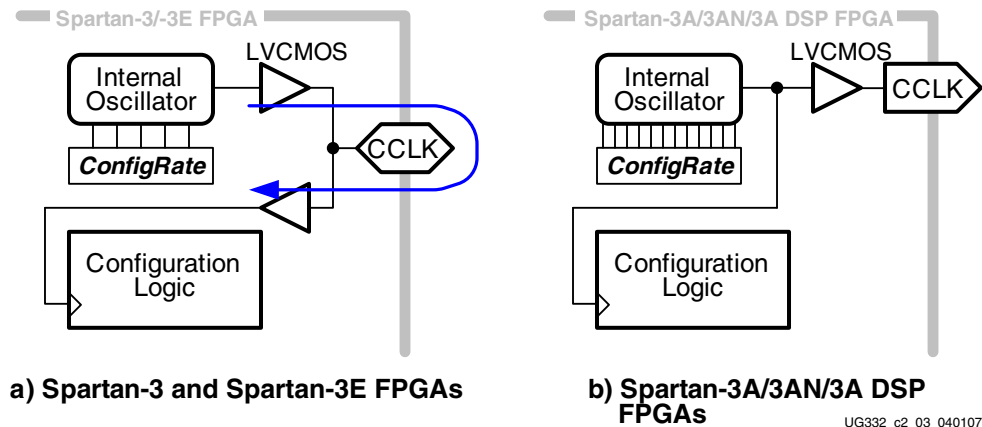


Figure 2-2: Differences between Spartan-3/3E and Extended Spartan-3A Family FPGAs for Master Configuration Modes

CCLK Design Considerations

The FPGA's configuration process is controlled by the CCLK configuration clock. Consequently, signal integrity of CCLK is important to guarantee successful configuration. Poor CCLK signal integrity caused by ringing or reflections potentially causes double-clocking, which might result in failed configuration.

Although the CCLK frequency is relatively low, the FPGA's output edge rates are fast. Therefore, pay careful attention to the CCLK signal integrity on the printed circuit board. Signal integrity simulation with IBIS is recommended. For all configuration modes except JTAG, the signal integrity must be considered at every CCLK trace destination, including the FPGA's CCLK pin.

This analysis is especially important for Spartan-3E FPGAs where the FPGA re-uses the CCLK pin as a user-I/O after configuration. In these cases, there might be unrelated devices attached to CCLK, which add additional trace length and signal destinations.

In the Master Serial, SPI, and BPI configuration modes, the FPGA drives the CCLK pin and CCLK should be treated as a full bidirectional I/O pin for signal integrity analysis. In BPI mode, CCLK is only connected to other devices in multi-FPGA daisy-chains, but switching noise at the FPGA pin could potentially cause false clocking.

The best signal integrity is ensured by following these basic PCB guidelines:

- Route the CCLK signal as a 50 Ω controlled-impedance transmission line.
- Route the CCLK signal without any branching. Do not use a "star" topology.
- Keep stubs, if required, shorter than 12.5 mm (0.5 inches).
- Terminate the end of the CCLK transmission line.

The clock termination examples shown below use parallel termination (Thevenin), but other approaches are acceptable. In parallel termination, the resistor values are twice the characteristic impedance of the board trace. The examples shown assume 50 Ω trace impedance. The disadvantage of parallel termination is that there is always a current path. Using series termination at the source and the end minimizes power, but use IBIS simulation to optimize resistor values for the specific application.

Figure 2-3 shows the basic point-to-point topology where the CCLK output from the Master FPGA drives one clock input receiver, either on the configuration PROM or on a slave FPGA.

Caution! On Spartan-3E and Extended Spartan-3A family FPGAs, be sure to define a valid logic level on CCLK. Otherwise, the clock trace might float and cause spurious clocking to other devices in the system.

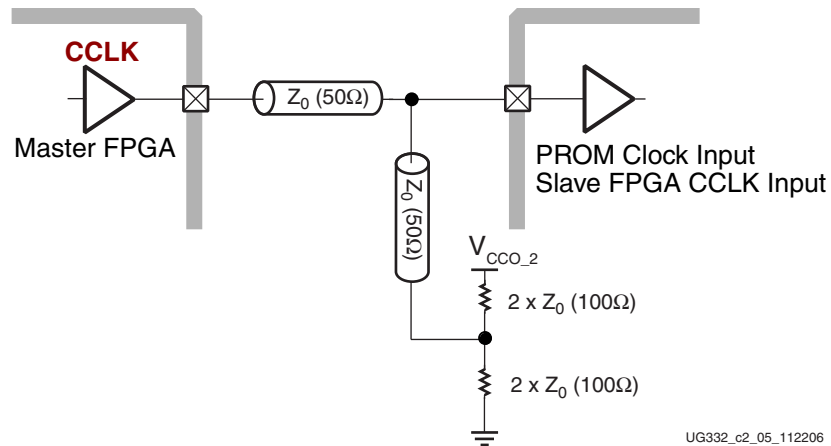


Figure 2-3: Point-to-Point: Master CCLK Output Drives Single Clock Load

Figure 2-4 shows the basic multi-drop flyby topology where the CCLK output from the Master FPGA drives two or more clock input receivers. Constrain the trace length on any clock stubs.

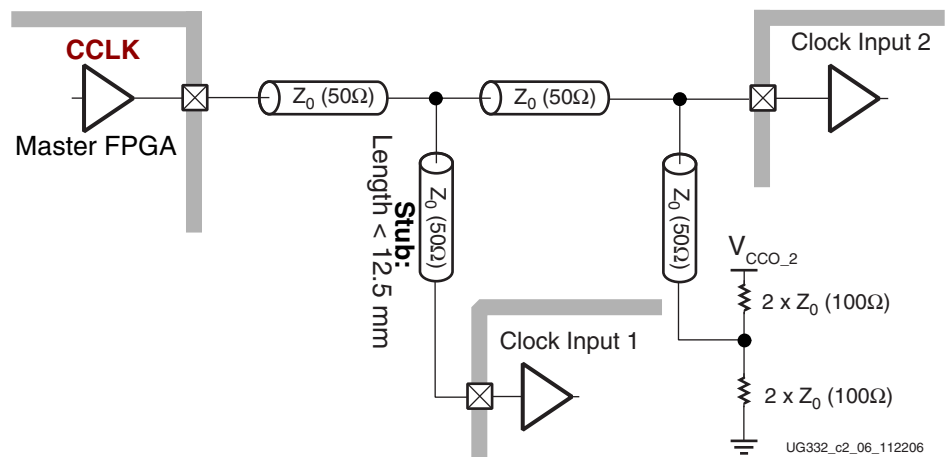


Figure 2-4: Multi-Drop: Master CCLK Output Drives Two Clock Inputs

Figure 2-5 shows a star topology where the Master FPGA CCLK transmission line branches to the multiple clock receiver inputs. The branch point creates a significant impedance discontinuity. Do not use this topology.

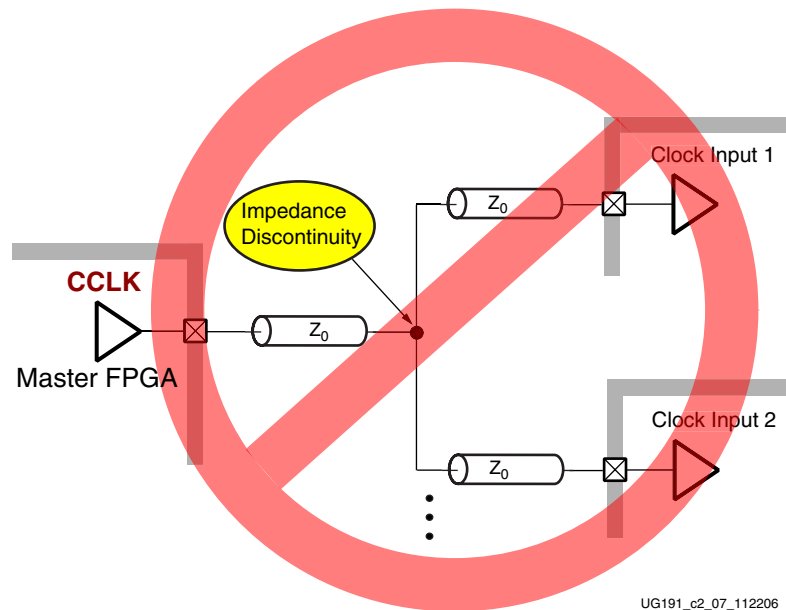


Figure 2-5: Star Topology Is Not Recommended

ConfigRate: Bitstream Option for CCLK

For Master configuration mode, the *ConfigRate* bitstream generator option defines the frequency of the internally-generated CCLK oscillator. The actual frequency is approximate due to the characteristics of the silicon oscillator and varies by up to 50% over the temperature and voltage range. On Spartan-3E and Extended Spartan-3A family FPGAs, the resulting frequency for every *ConfigRate* setting is fully characterized and specified in the associated FPGA family data sheet. At power-on, CCLK always starts operation at its lowest frequency. Use the *ConfigRate* option to set the oscillator frequency to one of the other values shown in Table 2-8.

Set this option graphically in “ISE Software Project Navigator,” page 42, as shown in Step 7 in Figure 1-7, page 44.

The FPGA does not start operating at the higher CCLK frequency until the *ConfigRate* control bits are loaded during the configuration process.

Persist: Reserve CCLK As Part of SelectMAP Interface

By default, any clocks applied to CCLK after configuration are ignored unless the bitstream option *Persist:Yes* is set, which retains the configuration interface. If *Persist:Yes*, then all clock edges are potentially active events, depending on the other configuration control signals. On Spartan-3E and Extended Spartan-3A family FPGAs, CCLK becomes a full-featured user-I/O pin after configuration.

Extended Spartan-3A Family and Spartan-3E FPGA Families

On the Extended Spartan-3A and Spartan-3E FPGA families, the CCLK pin is borrowed during configuration and becomes a full user I/O after configuration successfully completes.

The CCLK pin does not have a dedicated pull-up resistor during configuration. However, CCLK has an optional pull-up resistor to VCCO_2 during configuration, controlled by the Spartan-3E [HSWAP](#) pin or the Extended Spartan-3A family [PUDC_B](#) pin.

If the CCLK pin is not otherwise used by the FPGA application, then drive the pin High or Low.

Spartan-3 FPGA Family

During configuration, the CCLK pin has a dedicated internal pull-up resistor to V_{CCAUX}, regardless of the [HSWAP_EN](#) pin. After configuration, the CCLK pin is pulled High to V_{CCAUX} by default as defined by the [CclkPin](#) bitstream selection, although this behavior is programmable.

Any clocks applied to CCLK after configuration are ignored unless the bitstream option [Persist:Yes](#) is set, which retains the configuration interface. The [Persist:No](#) by default. However, if [Persist:Yes](#), then all clock edges are potentially active events, depending on the other configuration control signals.

Initializing Configuration Memory, Configuration Error: INIT_B

The INIT_B pin serves multiple purposes during configuration. Shortly after power is applied, the FPGA drives the INIT_B pin Low, indicating that initialization (*i.e.*, housecleaning) of the configuration memory has in progress. When INIT_B returns High, the FPGA samples the [M\[2:0\]](#) mode select pins and begins the configuration process.

During configuration, the INIT_B pin is an open-drain, bidirectional I/O pin with a dedicated, internal pull-up resistor, required to produce a High logic level. On Extended Spartan-3A family and Spartan-3E FPGAs, the INIT_B pull-up resistor connects to VCCO_2; on Spartan-3 FPGAs, the pull-up resistor connects to VCCO_4 or VCCO_BOTTOM (the connected VCCO_4 and VCCO_5), depending on the package style.

In a multi-FPGA daisy-chain or broadside configuration, connect (wire-AND) the INIT_B pins from all FPGAs together, as shown in [Figure 2-1, page 55](#). The common node ensures that all FPGAs in the design complete their respective housecleaning before any of the FPGAs is allowed to start configuring. The common node transitions High only after all of the FPGAs have been successfully initialized.

Externally holding this pin Low beyond the initialization phase delays the start of configuration. This action stalls the FPGA at the configuration step just before the [M\[2:0\]](#) mode select pins are sampled. See [“Delaying Configuration,” page 243](#).

During configuration, the FPGA indicates the occurrence of a configuration data error (*i.e.*, CRC error) by asserting INIT_B Low. See [“CRC Checking during Configuration,” page 309](#).

After Configuration

After configuration successfully completes, *i.e.*, when the [DONE](#) pin goes High, the INIT_B pin is available as a full user-I/O pin. The only exception is if the [“Extended Spartan-3A Family FPGA Post-Configuration CRC”](#) feature is enabled in the application, in which case the INIT_B is dedicated after configuration as well.

If the INIT_B pin is not used by the FPGA application after configuration, actively drive it High or Low. If left undefined, INIT_B, like all other unused pins, is defined by default as an input with an internal pull-down resistor. If the FPGA board uses an external pull-up resistor on INIT_B, then the unused pin will float at an intermediate value due to the presence of both a pull-up and pull-down resistor. To change the default configuration for unused pins, change the *UnusedPin* bitstream generator option setting.

If the bitstream generator option *Persist:Yes* is set, then INIT_B is reserved after configuration completes.

Extended Spartan-3A Family FPGA Post-Configuration CRC

If using a Spartan-3A FPGA, and if using the post-configuration CRC feature, then the INIT_B pin becomes a dedicated pin and flags any difference in the CRC signature during normal FPGA operation. See “[Post-Configuration CRC \(Extended Spartan-3A Family Only\)](#),” page 310 for more information.

Extended Spartan-3A Family and Spartan-3E FPGA Families

[INIT_B](#) is located in I/O Bank 2 and its output voltage determined by VCCO_2.

Spartan-3 FPGA Family

[INIT_B](#) is located in I/O Bank 4 and its output voltage determined by VCCO_4 or VCCO_BOTTOM, depending on package style.

Pull-Up Resistors During Configuration

The FPGA's configuration control pins have a dedicated, internal pull-up resistor that is active during the configuration process. All other I/O or Input-only pins have an optional pull-up resistor during configuration, controlled by a separate control input. The name of the control input varies by Spartan-3 generation family, as shown in [Table 2-12](#).

Pins with Dedicated Pull-Up Resistors during Configuration

[Table 2-9](#) shows the configuration control pins on all Spartan-3 generation FPGAs that have a built-in, dedicated, pull-up resistor during configuration. The table also indicates the supply rail to which the resistor is connected. The dedicated configuration pins also have a separate bitstream generator (BitGen) option setting that controls the pin's behavior after configuration.

Table 2-9: Pins with Dedicated Pull-Up Resistors during Configuration (All Spartan-3 Generation FPGAs)

Pin Name	Pull-Up Resistor Supply Rail	Post Configuration Control
PROG_B	V _{CCAUX}	<i>ProgPin</i> BitGen setting
DONE	V _{CCAUX}	<i>DonePin</i> and <i>DriveDone</i> BitGen settings
Pull-up during Configuration control input, HSWAP, PU _{DC} _B, or HSWAP_EN (see Table 2-12)	V _{CCO_0}	<i>Spartan-3E and Extended Spartan-3A family FPGAs</i> : User I/O after configuration. Controlled by the FPGA application <i>Spartan-3 FPGA</i> : Controlled by <i>HswapenPin</i> BitGen setting
INIT_B	<i>Spartan-3E/3A/3AN/ Spartan-3A DSP FPGAs</i> : V _{CCO_2} <i>Spartan-3 FPGA</i> : V _{CCO_4} or V _{CCO_BOTTOM}	User I/O after configuration. Controlled by the FPGA application
TDI	V _{CCAUX}	<i>TdiPin</i> BitGen setting
TMS	V _{CCAUX}	<i>TmsPin</i> BitGen setting
TCK	V _{CCAUX}	<i>TckPin</i> BitGen setting
TDO	V _{CCAUX}	<i>TdoPin</i> BitGen setting

As highlighted in Table 2-2, page 50, the Extended Spartan-3A family FPGAs add a few more dedicated internal pull-up resistors, as shown in Table 2-10. On Spartan-3E FPGAs, these pins do not have a dedicated internal pull-up resistor, but do have an optional pull-up resistor controlled when HSWAP = 0.

Table 2-10: Pins with Dedicated Pull-Up Resistors during Configuration (Extended Spartan-3A Family FPGAs Only)

Pin Name	Pull-Up Resistor Supply Rail	Post Configuration Control
M[2:0]	V _{CCO_2}	User I/O after configuration. Controlled by the FPGA application
VS[2:0]	V _{CCO_2}	Pull-up resistors only active when M[2:0]=<0:0:1>, Master SPI mode, or in Spartan-3AN FPGAs when M[2:0]=<0:1:1>, Internal Master SPI mode. User I/O after configuration. Controlled by the FPGA application

The Spartan-3 FPGA family uses dedicated configuration pins, as shown in Table 2-11. The post-configuration behavior is controlled by bitstream settings.

Table 2-11: Pins with Dedicated Pull-Up Resistors during Configuration (Spartan-3 FPGA Family Only)

Pin Name	Pull-Up Resistor Supply Rail	Post Configuration Control
M2	V _{CCAUX}	<i>M2Pin</i> BitGen setting
M1	V _{CCAUX}	<i>M1Pin</i> BitGen setting
M0	V _{CCAUX}	<i>M0Pin</i> BitGen setting
CCLK	V _{CCAUX}	<i>CclkPin</i> BitGen setting

Pins with Optional Pull-Up Resistors during Configuration

All user-I/O pins, input-only pins, and dual-purpose pins that are not actively involved in the currently-selected configuration mode are high impedance (floating, three-stated, Hi-Z) during the configuration process. These pins are indicated in [Table 2-17](#) as gray shaded table entries or cells.

A control input determines whether all user-I/O pins, input-only pins, and dual-purpose pins have a pull-up resistor to the supply rail or not. The control input has different names on different FPGA families as shown in [Table 2-12](#), but all function similarly. When the control is Low, each pin has an internal pull-up resistor that is active throughout configuration, starting immediately on power-up. Once the Mode pins are read, some of the dual-purpose pins will take on their configuration function for the remainder of the configuration process. After configuration, pull-up and pull-down resistors are available in the FPGA application by instantiating PULLUP or PULLDOWN primitive or by applying similarly-named constraints to a specific pin.

Table 2-12: Pull-Up Resistor during Configuration Control Input

FPGA Family	Pin Name	Function
Spartan-3A/3AN/ 3A DSP FPGA	PUDC_B	0: Pull-up resistors enabled during configuration 1: No pull-up resistors during configuration. Pins that are not active during the configuration process float Hi-Z.
Spartan-3E FPGA	HSWAP	
Spartan-3 FPGA	HSWAP_EN	

The control pin itself has a pull-up resistor enabled during configuration. However, the VCCO_0 supply voltage must be applied before the pull-up resistor becomes active. If the VCCO_0 supply ramps after the VCCO_2 power supply, do not let the control input pin float; tie the pin to the desired logic level externally. Note that Spartan-3E step 0 silicon requires that VCCINT be applied before VCCAUX when using a pull-up on HSWAP.

FPGA Pull-Up Resistor Values

The value of the dedicated and optional pull-up resistors is specified as a current, symbol I_{PJ} in the respective Spartan-3 generation data sheet. The equivalent resistor values provided in [Table 2-13](#) are for reference. The pull-up resistors on the Spartan-3 FPGA family are stronger than the other families.

Caution! The pull-up resistors in Spartan-3 FPGAs are strong, especially at higher V_{CCO} voltages.

Table 2-13: Pull-Up Resistor Ranges by Spartan-3 Generation Family

Voltage Range	Spartan-3 FPGA	Spartan-3E FPGA	Spartan-3A/3AN Spartan-3A DSP FPGA	Units
V_{CCAUX} or $V_{CCO} = 3.0$ to $3.6V$			5.1 to 23.9	k Ω
$V_{CCO} = 3.0$ to $3.45V$	1.27 to 4.11	2.4 to 10.8		
V_{CCAUX} or $V_{CCO} = 2.3$ to $2.7V$	1.15 to 3.25	2.7 to 11.8	6.2 to 33.1	
$V_{CCO} = 1.7$ to $1.9V$	2.45 to 9.10	4.3 to 20.2	8.4 to 52.6	

Table 2-14: Recommended External Pull-Up or Pull-down Resistor Values to Define Input Values during Configuration

PUDC_B, HSWAP, or HSWAP_EN	Desired Pull Direction	I/O Standard	Spartan-3 FPGA	Spartan-3E FPGA	Spartan-3A/3AN Spartan-3A DSP FPGA
= 0 (also applies to all pins that have a dedicated pull-up resistor during configuration, see "Pins with Dedicated Pull-Up Resistors during Configuration," page 62)	Pull-Up	All	No pull-up required. Internal pull-up resistors are enabled. See Table 2-13 for resistor range.		
	Pull-Down (required to overcome maximum I_{RPU} current and guarantee V_{IL})	LVC MOS33 LV TTL	$\leq 330 \Omega$	$\leq 620 \Omega$	$\leq 1.1 k\Omega$
		LVC MOS25	$\leq 470 \Omega$	$\leq 820 \Omega$	$\leq 1.8 k\Omega$
		LVC MOS18	$\leq 510 \Omega$	$\leq 820 \Omega$	$\leq 3.3 k\Omega$
		LVC MOS15	$\leq 820 \Omega$	$\leq 1.2 k\Omega$	$\leq 5.4 k\Omega$
LVC MOS12	$\leq 1.5 k\Omega$	$\leq 1.5 k\Omega$	$\leq 9.6 k\Omega$		
= 1 (optional pull-up resistors are disabled during configuration. Does not apply to pins with dedicated pull-up resistors during configuration)	Pull-Up (required to overcome single-load, maximum I_L leakage current and guarantee V_{IH})	LVC MOS33 LV TTL	$\leq 40 k\Omega$	$\leq 100 k\Omega$	
		LVC MOS25	$\leq 60 k\Omega$		
		LVC MOS18	$\leq 37 k\Omega$		
		LVC MOS15	$\leq 28 k\Omega$		
		LVC MOS12	$\leq 38 k\Omega$		
	Pull-Down (required to overcome single-load, maximum I_L leakage current and guarantee V_{IL})	LVC MOS33 LV TTL	$\leq 32 k\Omega$	$\leq 80 k\Omega$	
		LVC MOS25	$\leq 70 k\Omega$		
		LVC MOS18	$\leq 38 k\Omega$		
		LVC MOS15	$\leq 38 k\Omega$		
		LVC MOS12	$\leq 59 k\Omega$		

Pin Descriptions

Table 2-15 lists the various pins involved in the configuration process, including which configuration mode, the pin's direction, and a summary description. The table also describes how to use the pin during and after configuration.

Table 2-15: Spartan-3 Generation Configuration Pins, Associated Modes, and Function

Pin Name	Config. Mode(s)	FPGA Direction	Description	During Configuration	After Configuration
HSWAP or PUDC_B or HSWAP_EN (depends on FPGA family)	All	Input	User I/O Pull-Up Control. When Low during configuration, enables pull-up resistors in all I/O pins to respective I/O bank V _{CCO} input. 0: Pull-ups during configuration 1: No pull-ups	Drive at valid logic level throughout configuration.	Spartan-3: Dedicated pin (don't care after configuration) Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP: User I/O
M[2:0]	All	Input	Mode Select. Selects the FPGA configuration mode as defined in Table 2-1 .	Must be at the logic levels shown in Table 2-1, page 50 . Sampled when INIT_B goes High.	User I/O (dedicated on Spartan-3 FPGAs)
DIN	Serial Modes, SPI	Input	Serial Data Input. for all serial configuration modes	Receives serial data from PROM serial data output.	User I/O
CCLK	Master Modes, SPI, BPI	Output (treat as I/O for signal integrity)	Configuration Clock. Generated by FPGA internal oscillator. Frequency controlled by ConfigRate bitstream generator option. See "Configuration Clock: CCLK," page 56 .	Drives PROM's clock input.	User I/O (dedicated on Spartan-3 FPGAs)
	Slave Modes	Input	Configuration clock input.	Input configuration clock source.	
DOUT		Output	Serial Data Output.	Not used in single-FPGA designs; DOUT is pulled up, not actively driving. In a serial daisy-chain configuration, this pin connects to DIN input of the next FPGA in the chain.	User I/O
INIT_B	All	Open-drain bidirectional I/O	Initialization Indicator. Active Low. See "Initializing Configuration Memory, Configuration Error: INIT_B," page 61 .	Drives Low after power-on reset (POR) or when PROG_B pulsed Low while the FPGA is clearing its configuration memory. If a CRC error detected during configuration, FPGA again drives INIT_B Low.	User I/O. If unused in the application, drive INIT_B High or Low to avoid a floating value. See INIT_B "After Configuration".

Table 2-15: Spartan-3 Generation Configuration Pins, Associated Modes, and Function (Cont'd)

Pin Name	Config. Mode(s)	FPGA Direction	Description	During Configuration	After Configuration
DONE	All	Open-drain bidirectional I/O	FPGA Configuration Done. Low during configuration. Goes High when FPGA successfully completes configuration. Powered by V_{CCAUX} supply. 0: FPGA not configured 1: FPGA configured See “DONE Pin,” page 52	Actively drives Low during configuration.	When High, indicates that the FPGA successfully configured.
PROG_B	All	Input	Program FPGA. Active Low. When asserted Low for 500 ns or longer, forces the FPGA to restart its configuration process by clearing configuration memory and resetting the DONE and INIT_B pins. If driving externally with a 3.3V output, use an open-drain or open-collector driver or use a current limiting series resistor. See “Program or Reset FPGA: PROG_B,” page 56.	Must be High during configuration to allow configuration to start.	Drive PROG_B Low and release to reprogram FPGA.
Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP FPGA: VS[2:0]	Master SPI	Input	Variant Select. Instructs the FPGA how to communicate with the attached SPI Flash PROM.	Must be at the logic levels shown in Table 4-2, page 105. Sampled when INIT_B goes High.	User I/O
Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP FPGA: MOSI	Master SPI	Output	Serial Data Output.	FPGA sends SPI Flash memory read commands and starting address to the PROM’s serial data input.	User I/O
Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP FPGA:CSO_B	Master SPI	Output	Chip Select Output. Active Low.	Connects to the SPI Flash PROM’s Slave Select input. If HSWAP/PUDC_B = 1, connect this signal to a 4.7 k Ω pull-up resistor to 3.3V.	Drive CSO_B High after configuration to disable the SPI Flash and reclaim the MOSI, DIN, and CCLK pins. Optionally, re-use this pin and MOSI, DIN, and CCLK to continue communicating with SPI Flash.

Table 2-15: Spartan-3 Generation Configuration Pins, Associated Modes, and Function (Cont'd)

Pin Name	Config. Mode(s)	FPGA Direction	Description	During Configuration	After Configuration
Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP FPGA: CSL_B Spartan-3 FPGA: CS_B	BPI, Slave Parallel	Input	Chip Select Input. Active Low.	Active-Low.	User I/O. If bitstream option <i>Persist:Yes</i> , becomes part of SelectMap parallel peripheral interface.
RDWR_B	BPI, Slave Parallel	Input	Read/Write Control. Active Low write enable. Read functionality typically only used after configuration, if bitstream option <i>Persist:Yes</i> .	Must be Low throughout configuration. Do not change logic level while <i>CSL_B</i> is Low	User I/O. If bitstream option <i>Persist:Yes</i> , becomes part of SelectMap parallel peripheral interface.
Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP FPGA: LDC0	BPI	Output	PROM Chip Enable	Connect to parallel PROM chip-select input (<i>CS#</i>). FPGA drives this signal Low throughout configuration.	User I/O. If the FPGA does not access the PROM after configuration, drive this pin High to deselect the PROM. A[23:0], D[7:0], LDC[2:1], and HDC then become available as user I/O.
Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP FPGA: LDC1	BPI	Output	PROM Output Enable	Connect to the parallel PROM output-enable input (<i>OE#</i>). The FPGA drives this signal Low throughout configuration.	User I/O
Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP FPGA: HDC	BPI	Output	PROM Write Enable	Connect to parallel PROM write-enable input (<i>WE#</i>). FPGA drives this signal High throughout configuration.	User I/O
Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP FPGA: LDC2	BPI	Output	PROM Byte Mode	This signal is not used for x8 PROMs. For PROMs with a x8/x16 data width control, connect to PROM byte-mode input (<i>BYTE#</i>).	User I/O. Drive this pin High after configuration to use a x8/x16 PROM in x16 mode.

Table 2-15: Spartan-3 Generation Configuration Pins, Associated Modes, and Function (Cont'd)

Pin Name	Config. Mode(s)	FPGA Direction	Description	During Configuration	After Configuration
Spartan-3E FPGA: A[23:0] Spartan-3A Spartan-3AN Spartan-3A DSP FPGA: A[25:0]	BPI	Output	Parallel PROM Address outputs	Connect to PROM address inputs.	User I/O.
D[7:0]	Master Parallel, BPI, Slave Parallel, SelectMAP	Input	Data Input	Data captured by FPGA	User I/O. If bitstream option <i>Persist:Yes</i> , becomes part of SelectMap parallel peripheral interface.
Spartan-3/ Spartan-3E FPGA: BUSY	BPI, Slave Parallel (SelectMAP)	Output	FPGA Busy Indicator. Used primarily in Slave Parallel interfaces that operate at 50 MHz and faster. Same function is on DOUT pin in the Extended Spartan-3A family.	Not used during BPI mode configuration but actively drives.	User I/O. If bitstream option <i>Persist:Yes</i> , becomes part of SelectMap parallel peripheral interface.

Pin Behavior During Configuration

Table 2-16, Table 2-17, and Table 2-18 show how various pins on Spartan-3 generation FPGAs behave during the configuration process. The actual behavior depends on the settings applied to the M2, M1, and M0 (M[2:0]) mode select pins and the pin that controls the optional pull-up resistors, called HSWAP, PUDC_B, or HSWAP_EN depending on the specific Spartan-3 generation FPGA family. The M[2:0] mode select pins determine which of the I/O pins are active and borrowed during configuration and how they function. In JTAG configuration mode, no user-I/O pins are borrowed for configuration.

The **Dedicated Pull-Up Resistor** column indicates pins that always have a pull-up resistor enabled during configuration, regardless of the PUDC_B, HSWAP, or HSWAP_EN input. After configuration, the behavior of these pins is either defined by specific bitstream generator options or by the FPGA application itself.

Table 2-16, Table 2-17, and Table 2-18 show the FPGA pins that are either borrowed or dedicated during configuration. The specific pins are listed by FPGA configuration mode along the top. For each pin, the table also indicates the power rail that supplies the pin during configuration. A numeric value such as "2", indicates that the associated pin is located in I/O Bank 2 and powered by the VCCO_2 supply inputs. Spartan-3E and Extended Spartan-3A family FPGAs have four I/O banks; the Spartan-3 FPGA family has eight I/O banks.

The pin names are color-coded using the same colors used in the package pinout tables and footprint diagrams found in the respective Spartan-3 generation data sheet. Black represents the dedicated JTAG pins; yellow represents the dedicated configuration pins; light blue represents the dual-purpose configuration pins that become user-I/O pins after configuration.

Extended Spartan-3A Family FPGA

Table 2-16 shows the various Extended Spartan-3A family FPGA pins that are either borrowed or dedicated during configuration.

Table 2-16: Extended Spartan-3A Family FPGAs: Pin Behavior during Configuration

Pin Name	Dedicated Pull-Up Resistor	Master Serial	SPI (Serial Flash)	Internal Master SPI	BPI (Parallel Flash)	JTAG	Slave Serial	Slave Parallel	Supply/I/O Bank
IO* (user-I/O) IP* (input-only)	–								See pinout table
TDI	Yes	TDI	TDI	TDI	TDI	TDI	TDI	TDI	V _{CCAUX}
TMS	Yes	TMS	TMS	TMS	TMS	TMS	TMS	TMS	V _{CCAUX}
TCK	Yes	TCK	TCK	TCK	TCK	TCK	TCK	TCK	V _{CCAUX}
TDO	Yes	TDO	TDO	TDO	TDO	TDO	TDO	TDO	V _{CCAUX}
PROG_B	Yes	PROG_B	PROG_B	PROG_B	PROG_B	PROG_B	PROG_B	PROG_B	V _{CCAUX}
DONE	Yes	DONE	DONE	DONE	DONE	DONE	DONE	DONE	V _{CCAUX}
PUDC_B	Yes	PUDC_B	PUDC_B	PUDC_B	PUDC_B	PUDC_B	PUDC_B	PUDC_B	0
M2	Yes	0	0	0	0	1	1	1	2
M1	Yes	0	0	1	1	0	1	1	2
M0	Yes	0	1	1	0	1	1	0	2
CCLK	–	CCLK (OUTPUT)	CCLK (OUTPUT)		CCLK (OUTPUT)		CCLK (INPUT)	CCLK (INPUT)	2
INIT_B	Yes	INIT_B	INIT_B	INIT_B	INIT_B		INIT_B	INIT_B	2
CSO_B	–		CSO_B		CSO_B			CSO_B	2
DOUT	–	DOUT	DOUT		DOUT ⁽⁵⁾		DOUT	DOUT ⁽⁵⁾	2
MOSI/CSI_B	–		MOSI		CSI_B			CSI_B	2
D[7:1]	–				D[7:1]			D[7:1]	2
D0/DIN	–	DIN	DIN		D0		DIN	D0	2
RDWR_B	–				RDWR_B			RDWR_B	2
VS[2:0]	Yes		VS[2:0]	VS[2:0]					2
A[25:0]	–				A[25:0]				1
LDC2	–				LDC2				1
LDC1	–				LDC1				1
LDC0	–				LDC0				1
HDC	–				HDC				1

Notes:

- Gray shaded cells represent pins that are in a high-impedance state (Hi-Z, floating) during configuration. These pins have an optional internal pull-up resistor to their respective V_{CCO} supply pin that is active during configuration if the PUDC_B input is Low. See “Pull-Up Resistors During Configuration,” page 62.
- The Spartan-3E HSWAP pin and the Extended Spartan-3A family PUDC_B pin have identical behavior, just different names. See “Pull-Up Resistors During Configuration,” page 62.
- The Internal Master SPI mode, M[2:0] = <0:1:1>, is only available on the Spartan-3AN FPGA family. V_{CCAUX} must be 3.3V when using this mode.
- CCLK is always an input pin in Slave configuration modes. For Master modes, CCLK must be treated as a bidirectional I/O pin for Spartan-3E FPGAs and is an output pin for Extended Spartan-3A family FPGAs.
- The DOUT output is not labeled as BUSY and the BUSY function is not required on Extended Spartan-3A family FPGAs. However, the pin can still toggle during Slave Parallel configuration and therefore should not be tied to user functions during configuration. Unlike Spartan-3E FPGAs, Extended Spartan-3A family FPGAs do use the DOUT pin in BPI serial daisy-chains, which are only supported on Extended Spartan-3A family FPGAs.

Spartan-3E FPGAs

Table 2-17 shows the various Spartan-3E FPGA pins that are either borrowed or dedicated during configuration.

Table 2-17: Spartan-3E FPGAs: Pin Behavior during Configuration

Pin Name	Dedicated Pull-Up Resistor	Master Serial	SPI (Serial Flash)	BPI (Parallel Flash)	JTAG	Slave Serial	Slave Parallel	Supply/ I/O Bank
IO* (user-I/O) IP* (input-only)	–							See pinout table
TDI	Yes	TDI	TDI	TDI	TDI	TDI	TDI	V _{CCAUX}
TMS	Yes	TMS	TMS	TMS	TMS	TMS	TMS	V _{CCAUX}
TCK	Yes	TCK	TCK	TCK	TCK	TCK	TCK	V _{CCAUX}
TDO	Yes	TDO	TDO	TDO	TDO	TDO	TDO	V _{CCAUX}
PROG_B	Yes	PROG_B	PROG_B	PROG_B	PROG_B	PROG_B	PROG_B	V _{CCAUX}
DONE	Yes	DONE	DONE	DONE	DONE	DONE	DONE	V _{CCAUX}
HSWAP	Yes	HSWAP	HSWAP	HSWAP	HSWAP	HSWAP	HSWAP	0
M2	–	0	0	0	1	1	1	2
M1	–	0	0	1	0	1	1	2
M0	–	0	1	0 = Up 1 = Down	1	1	0	2
CCLK	–	CCLK (I/O)	CCLK (I/O)	CCLK (I/O)		CCLK (INPUT)	CCLK (INPUT)	2
INIT_B	Yes	INIT_B	INIT_B	INIT_B		INIT_B	INIT_B	2
CSO_B	–		CSO_B	CSO_B			CSO_B	2
DOUT/BUSY	–	DOUT	DOUT	BUSY		DOUT	BUSY	2
MOSI/CSI_B	–		MOSI	CSI_B			CSI_B	2
D[7:1]	–			D[7:1]			D[7:1]	2
D0/DIN	–	DIN	DIN	D0		DIN	D0	2
RDWR_B	–			RDWR_B			RDWR_B	2
VS[2:0]	–		VS[2:0]	(Note 4)				2
A[23:17]	–		(Note 4)	A[23:17]				2
A[16:0]	–			A[16:0]				1
LDC2	–			LDC2				1
LDC1	–			LDC1				1
LDC0	–			LDC0				1
HDC	–			HDC				1

Notes:

- Gray shaded cells represent pins that are in a high-impedance state (Hi-Z, floating) during configuration. These pins have an optional internal pull-up resistor to their respective V_{CCO} supply pin that is active during configuration if the HSWAP input is Low. See “Pull-Up Resistors During Configuration,” page 62.
- The Spartan-3E HSWAP pin and the Extended Spartan-3A family PUDC_B pin have identical behavior, just different names. See “Pull-Up Resistors During Configuration,” page 62.
- CCLK is always an input pin in Slave configuration modes. For Master modes, CCLK must be treated as a bidirectional I/O pin for Spartan-3E FPGAs.
- On Spartan-3E FPGAs, the VS[2:0] pins used in Master SPI mode are shared with the A[19:17] address pins used in BPI mode.

Spartan-3 FPGAs

Table 2-18 shows the various Spartan-3 FPGA pins that are either borrowed or dedicated during configuration.

Table 2-18: Pin Behavior during Configuration for Spartan-3 FPGA Family

Pin Name	Dedicated Pull-Up Resistor	Master Serial	Master Parallel	JTAG	Slave Serial	Slave Parallel	Supply/ I/O Bank
IO* (user-I/O) IP* (input-only)	–						See pinout table
TDI	Yes	TDI	TDI	TDI	TDI	TDI	V _{CCAUX}
TMS	Yes	TMS	TMS	TMS	TMS	TMS	V _{CCAUX}
TCK	Yes	TCK	TCK	TCK	TCK	TCK	V _{CCAUX}
TDO	Yes	TDO	TDO	TDO	TDO	TDO	V _{CCAUX}
PROG_B	Yes	PROG_B	PROG_B	PROG_B	PROG_B	PROG_B	V _{CCAUX}
DONE	Yes	DONE	DONE	DONE	DONE	DONE	V _{CCAUX}
HSWAP_EN	Yes	HSWAP_EN	HSWAP_EN	HSWAP_EN	HSWAP_EN	HSWAP_EN	V _{CCAUX}
M2	Yes	0	0	1	1	1	V _{CCAUX}
M1	Yes	0	1	0	1	1	V _{CCAUX}
M0	Yes	0	1	1	1	0	V _{CCAUX}
CCLK	Yes	CCLK (I/O)	CCLK (I/O)		CCLK (INPUT)	CCLK (INPUT)	V _{CCAUX}
INIT_B	Yes	INIT_B	INIT_B		INIT_B	INIT_B	4
CS_B	–		CS_B			CS_B	5
DOUT/BUSY	–	DOUT	BUSY		DOUT	BUSY	4
D[7:4]	–		D[7:4]			D[7:4]	5
D[3:1]	–		D[3:1]			D[3:1]	4
D0/DIN	–	DIN	D0		DIN	D0	4
RDWR_B	–		RDWR_B			RDWR_B	5

Notes:

- Gray shaded cells represent pins that are in a high-impedance state (Hi-Z, floating) during configuration. These pins have an optional internal pull-up resistor to their respective V_{CCO} supply pin that is active during configuration if the HSWAP_EN input is Low.
- CCLK is always an input pin in Slave configuration modes. For Master modes, CCLK must be treated as a bidirectional I/O pin.

Default I/O Standard During Configuration

During configuration, which includes the time when the FPGA first powers up, after PROG_B is pulsed Low, or during configuration or re-configuration including MultiBoot, the FPGA's I/O pins are unconfigured. However, the FPGA pins involved in the configuration process are predefined to the settings shown in Table 2-19.

Table 2-19: Default I/O Standard Setting During Configuration

Pin(s)	I/O Standard	Output Drive	Slew Rate
All, including CCLK	LVC MOS25	8 mA	Slow

By default, the I/O pins are set for LVCMOS25 operation or 2.5V low-voltage CMOS. The setting is the same for both the Dedicated and Dual-Purpose pins. However, the Dual-Purpose pins can drive at different voltages, depending on the voltage applied to the relevant I/O bank.

The Dedicated configuration pins (see [Table 2-17, page 71](#) and [Table 2-18, page 72](#)) are always powered by V_{CCAUX} . On Spartan-3 and Spartan-3E FPGA families, V_{CCAUX} is always 2.5V, as shown in [Table 2-18](#). On Spartan-3A/3A DSP FPGAs, V_{CCAUX} can be either 2.5V or 3.3V. On Spartan-3AN FPGAs, V_{CCAUX} is always 3.3V. See “[V_{CCAUX} Level](#)”.

The Dual-Purpose configuration pin outputs operate at other voltages by appropriately setting the voltage on the associated V_{CCO} power rail. For Spartan-3A/3A DSP (and for Spartan-3AN FPGAs in modes other than Internal Master SPI) and Spartan-3E FPGAs, the Dual-Purpose configuration pins are supplied by the V_{CCO_2} rail, plus V_{CCO_1} in BPI mode. In Spartan-3 FPGAs, the Dual-Purpose configuration pins are supplied by V_{CCO_4} , plus V_{CCO_5} in any of the parallel configuration modes. In general, set the configuration voltage to either 2.5V or 3.3V. The change on the V_{CCO} supply also changes the I/O drive characteristics. For example, with $V_{CCO} = 3.3V$, the output current when driving High, I_{OH} , increases to approximately 12 to 16 mA, while the current when driving Low, I_{OL} , remains 8 mA.

If required, V_{CCO} may be set to 1.8V in the Spartan-3 and Spartan-3E families. See “[V_{CCAUX} Level](#)”. At $V_{CCO} = 1.8V$, the output current when driving High, I_{OH} , decreases slightly to approximately 6 to 8 mA. Again, the current when driving Low, I_{OL} , remains 8 mA. The output voltages will be determined by the V_{CCO} level, LVCMOS18 for 1.8V, LVCMOS25 for 2.5V, and LVCMOS33 for 3.3V.

Table 2-20: Supported Configuration Interface Voltages

FPGA Family	Dedicated Pins	Dual-Purpose Pins	
	Supported V_{CCAUX} Voltage Options	Dual-Purpose Configuration Pin Supply Rails	Supported Configuration Supply Voltage Options
Spartan-3A Spartan-3A DSP FPGAs	2.5V 3.3V	V_{CCO_2} (sometimes V_{CCO_1})	2.5V 3.3V
Spartan-3AN ⁽¹⁾	3.3V		
Spartan-3E FPGAs	2.5V	V_{CCO_2} (sometimes V_{CCO_1})	2.5V 3.3V
Spartan-3 FPGAs	2.5V	V_{CCO_4} (sometimes V_{CCO_5})	2.5V 3.3V

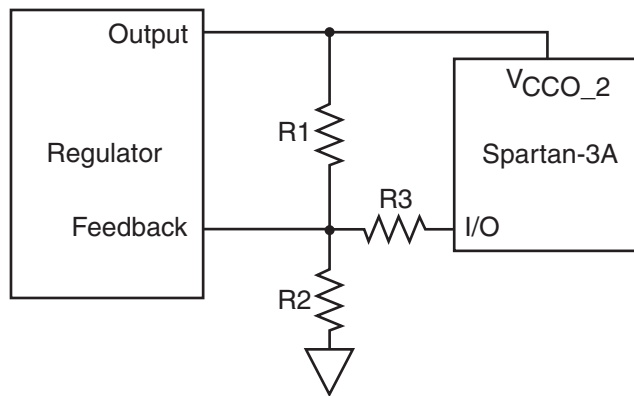
Notes:

1. Spartan-3AN FPGAs in Internal Master SPI mode only require the 3.3V V_{CCAUX} supply because there are no Dual-Purpose pins involved. In all other configuration modes, the Dual-Purpose pins are involved.

Lowering V_{CCO_2} After Configuration in Extended Spartan-3A Family

The Extended Spartan-3A family families have a V_{CCO2T} threshold requirement of 2.0V minimum and therefore cannot have V_{CCO_2} at 1.8V during configuration; however V_{CCO2T} does not apply after configuration and therefore V_{CCO_2} can be lowered to 1.8V or lower after DONE goes High.

This could be accomplished by using an adjustable regulator with a feedback loop to set the output voltage, using a resistor divider network to define the voltage values. One resistor would be connected to an I/O which is disabled before configuration (using PUDC_B High) and Low after configuration (driving it to GND in the design). In [Figure 2-6](#), R1 in series with R2 would set the regulator output voltage at 2.0V or higher for power-on and during configuration. Resistor R3 is enabled after configuration, and the parallel resistance of R2 and R3 would replace R2 to set the regulator voltage at 1.8V or lower after configuration.



UG332_c2_08_012709

Figure 2-6: Using Resistor Divider Network to Meet VCCO_2 POR Threshold

Design Considerations for the HSWAP, M[2:0], and VS[2:0] Pins

Spartan-3E and Extended Spartan-3A family FPGAs are unlike previous Spartan FPGA families. Nearly all of the Spartan-3E/3A/3AN/3A DSP dual-purpose configuration pins are available as full-featured user I/O pins after successful configuration.

The HSWAP or PUDC_B pin, the mode select pins (M[2:0]), and the variant-select pins (VS[2:0]) must have valid and stable logic values at the start of configuration. VS[2:0] are only used in the Master SPI configuration mode. The levels on the M[2:0] pins and VS[2:0] pins are sampled when the INIT_B pin returns High. See [Figure 2-7](#) for a timing example.

The HSWAP or PUDC_B pin defines whether FPGA user I/O pins have a pull-up resistor connected to their associated V_{CCO} supply pin during configuration or not, as shown [Table 2-20](#). HSWAP or PUDC_B must be valid at the start of configuration and remain constant throughout the configuration process.

The detailed schematics for each configuration mode indicate the required logic values for HSWAP or PUDC_B, M[2:0], and VS[2:0] but do not specify how the application provides the logic Low or High value. The HSWAP or PUDC_B, M[2:0], and VS[2:0] pins can be either dedicated or reused by the FPGA application.

Dedicating the HSWAP, PUDC_B, M[2:0], and VS[2:0] Pins

If the HSWAP or PUDC_B, M[2:0], and VS[2:0] pins are not required by the FPGA application after configuration, simply connect these pins directly to the V_{CCO} or GND supply rail shown in the appropriate configuration schematic.

Optionally, use external pull-up or pull-down resistors to define the appropriate logic level. The external resistors provide the ability to temporarily change the logic level for debugging purposes. Some of these pins have dedicated pull-up resistors during configuration. See [Table 2-14, page 65](#) for recommended resistor values.

Be sure to define the post-configuration behavior for these pins to avoid unnecessary current paths. For example, see “[Defining M\[2:0\] after Configuration for Minimum Power Consumption,](#)” [page 51](#).

Reusing HSWAP, PUDC_B, M[2:0], and VS[2:0] After Configuration

To reuse the HSWAP or PUDC_B, M[2:0], and VS[2:0] pins after configuration, use pull-up or pull-down resistors to define the logic values shown in the appropriate configuration schematic. Some of these pins have dedicated pull-up resistors during configuration. See [Table 2-14, page 65](#) for recommended resistor values.

Use the weakest external pull-up or pull-down resistor value acceptable in the application. The resistor must be strong enough to define a logic Low or High during configuration. However, when driving the HSWAP or PUDC_B, M[2:0], or VS[2:0] pins after configuration, an external output driver must be strong enough to overcome the pull-up or pull-down resistor value and generate the appropriate logic levels. For example, to overcome a 560 Ω pull-down resistor, a 3.3V FPGA I/O pin must use a 6 mA or stronger driver.

Spartan-3E HSWAP Considerations

For Spartan-3E FPGAs, the logic level on HSWAP dictates how to define the logic levels on M[2:0] and VS[2:0], as shown in [Table 2-21, page 76](#). If the application requires HSWAP to be High, a dedicated internal pull-up to V_{CCO_0} is available, although an external 3.3 to

4.7 k Ω resistor is recommended, especially if VCCO_2 (configuration) might be applied before VCCO_0 (HSWAP). Note that initial Spartan-3E silicon (step 0) required that V_{CCINT} be applied before V_{CCAUX} when using a pull-up on HSWAP.

If the application requires HSWAP to be Low during configuration, then HSWAP is either connected to GND or pulled Low using an appropriately sized external pull-down resistor to GND. The pin itself has an internal pull-up resistor to VCCO_0, so the external pull-down resistor must be strong enough to define a logic Low on HSWAP for the I/O standard used during configuration, as shown in [Table 2-14, page 65](#).

Once HSWAP is defined, use [Table 2-21](#) to define the logic values for M[2:0] and VS[2:0].

Table 2-21: Pull-up or Pull-down Values for HSWAP, M[2:0], and VS[2:0]

HSWAP Value	I/O Pull-up Resistors during Configuration	Required Resistor Value to Define Logic Level on HSWAP, M[2:0], or VS[2:0]	
		High	Low
0	Enabled	Pulled High via an internal pull-up resistor to the associated V _{CCO} supply. No external pull-up resistor is necessary.	Pulled Low using an appropriately sized pull-down resistor to GND, as shown in Table 2-14, page 65 .
1	Disabled	Pulled High using a 3.3 to 4.7 k Ω resistor to the associated V _{CCO} supply.	Pulled Low using a 3.3 to 4.7 k Ω resistor to GND.

Dual-Purpose Pins Become User I/O

All dual-purpose I/O pins that are borrowed during configuration become full-function I/O pins after configuration successfully completes. [Figure 2-7](#) shows stylized waveforms for some of the configuration control signals. On Spartan-3E and Extended Spartan-3A family FPGAs, the M[2:0] mode pins, the VS[2:0] pin in Master SPI mode, the CCLK pin, and the HSWAP or PUDC_B pin are borrowed during configuration. After configuration completes, the pins become available as user-I/O pins.

All dual-purpose I/O pins, except for CCLK, become available to the FPGA application immediately following the GTS cycle during the FPGA Startup sequence. The GTS cycle timing is controlled by the *GTS_cycle* bitstream option.

The CCLK configuration clock does not become a user-defined I/O until after the entire configuration sequence is complete.

See [Chapter 12, “Sequence of Events”](#) for more information.

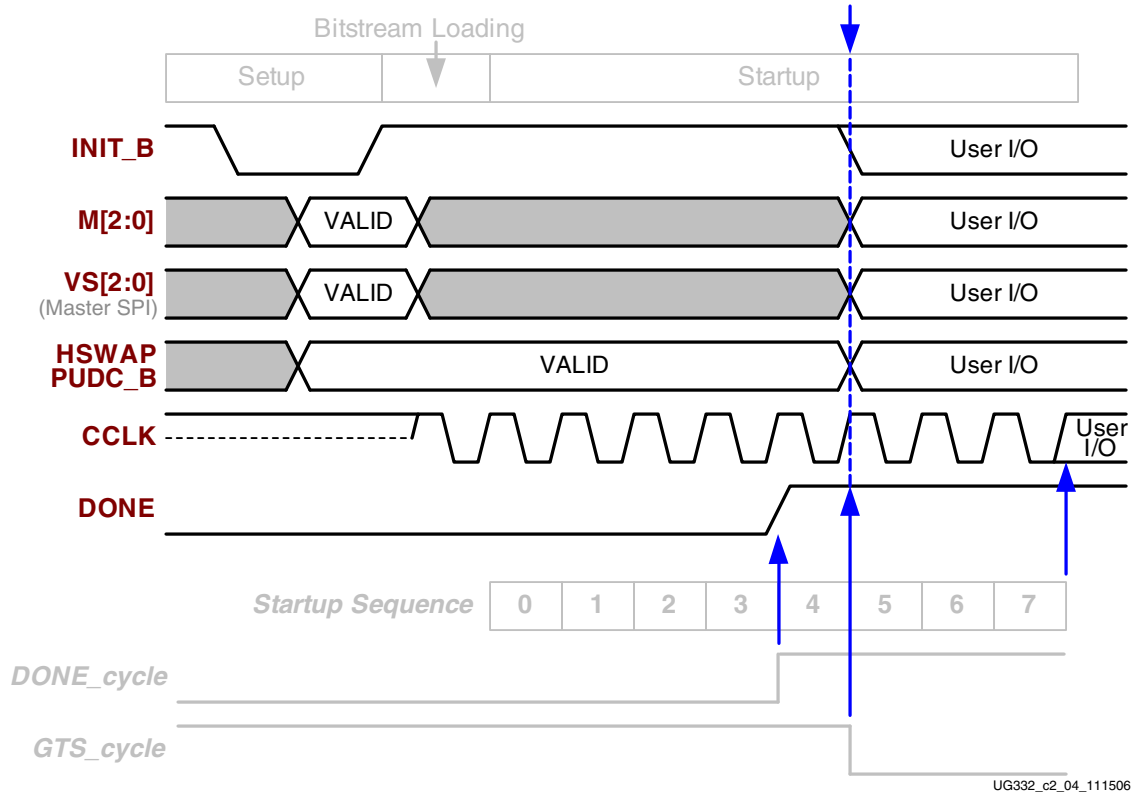


Figure 2-7: Stylized Configuration Waveforms Showing When Dual-Purpose Pins Become Active

V_{CCAUX} Level

In the Spartan-3A and Spartan-3A DSP platforms, the V_{CCAUX} level is programmable as either 2.5V (default) or 3.3V. The user specifies the value in the software through the CONFIG V_{CCAUX}=2.5 or CONFIG V_{CCAUX}=3.3 constraint. In the Spartan-3AN platform, the user must set CONFIG V_{CCAUX}=3.3 (default) for using the In-System Flash. The Spartan-3 and Spartan-3E families have a fixed V_{CCAUX} at 2.5V.

Master Serial Mode

The Master Serial configuration mode leverages the purpose-designed [Xilinx® Platform Flash PROMs](#) to configure Spartan®-3 generation FPGAs. Master Serial mode uses the serial interface offered on XCFxxS serial PROMs and the serial interface option on XCFxxP serial/parallel PROMs.

Xilinx Platform Flash PROMs offer the following system advantages.

- Simple interface. Fewest number of FPGA pins used during configuration.
- Low cost per configuration bit.
- Highest bandwidth between PROM and FPGA for any serial PROM, resulting in fastest configuration time.
- Small package footprint.
- In-system programmable and reprogrammable via an integrated JTAG interface.
- Fully supported by the Xilinx iMPACT programming software.
- Multiple I/O and JTAG voltage ranges for maximum system flexibility.
- Density migration within a common package footprint. See [Table 1-3, page 37](#).
- Sold and supported by Xilinx, with the long product lifetime and reliability associated with Xilinx products.

In Master Serial mode ($M[2:0] = <0:0:0>$), the Spartan-3 generation FPGA configures itself from an attached [Xilinx Platform Flash PROM](#), as illustrated in [Figure 3-1](#), [Figure 3-2](#), and [Figure 3-3](#).

The figures show optional components in gray and designated “NO LOAD”. For example, the Bitstream Generator option **ProgPin Pullup** internally connects a pull-up resistor between the PROG_B pin and V_{CCAUX} . An external 4.7 kΩ pull-up resistor to V_{CCAUX} is still recommended. The external pull-up provides a known pull-up value, and can be stronger than the internal pull-up alone, which the data sheet specifies at up to 12 kΩ

The FPGA supplies the **CCLK** output clock from its internal oscillator to the attached Platform Flash PROM. In response, the Platform Flash PROM supplies bit-serial data to the FPGA’s **DIN** input, and the FPGA accepts this data on each rising **CCLK** edge.

All the FPGA mode-select pins, $M[2:0]$, must be Low when sampled, which occurs when the FPGA’s **INIT_B** output initially goes High.

The FPGA’s **DOUT** pin is used in daisy-chain applications, described in “[Daisy-Chained Configuration](#),” [page 86](#). In a single-FPGA application, the FPGA’s **DOUT** pin is inactive, but pulled High via an internal resistor.

The Master Serial interface varies slightly between Spartan-3 generation FPGAs.

- [Figure 3-1, page 80](#) illustrates the Master Serial configuration interface for Spartan-3E and Spartan-3A/3A DSP FPGAs when the FPGA’s V_{CCAUX} supply is at 2.5V. Spartan-3E FPGAs

always have $V_{CCAUX} = 2.5V$. Spartan-3A and Spartan-3A DSP FPGAs support both $V_{CCAUX} = 2.5V$ or $3.3V$. Table 3-1, page 80 lists the FPGA/PROM connections.

- Figure 3-2, page 81 illustrates the Master Serial configuration interface for Extended Spartan-3A family FPGAs when $V_{CCAUX} = 3.3V$. Spartan-3AN FPGAs always have $V_{CCAUX} = 3.3V$. Table 3-1, page 80 lists the FPGA/PROM connections.
- Figure 3-3, page 82 illustrates the Master Serial configuration interface for Spartan-3 FPGAs.

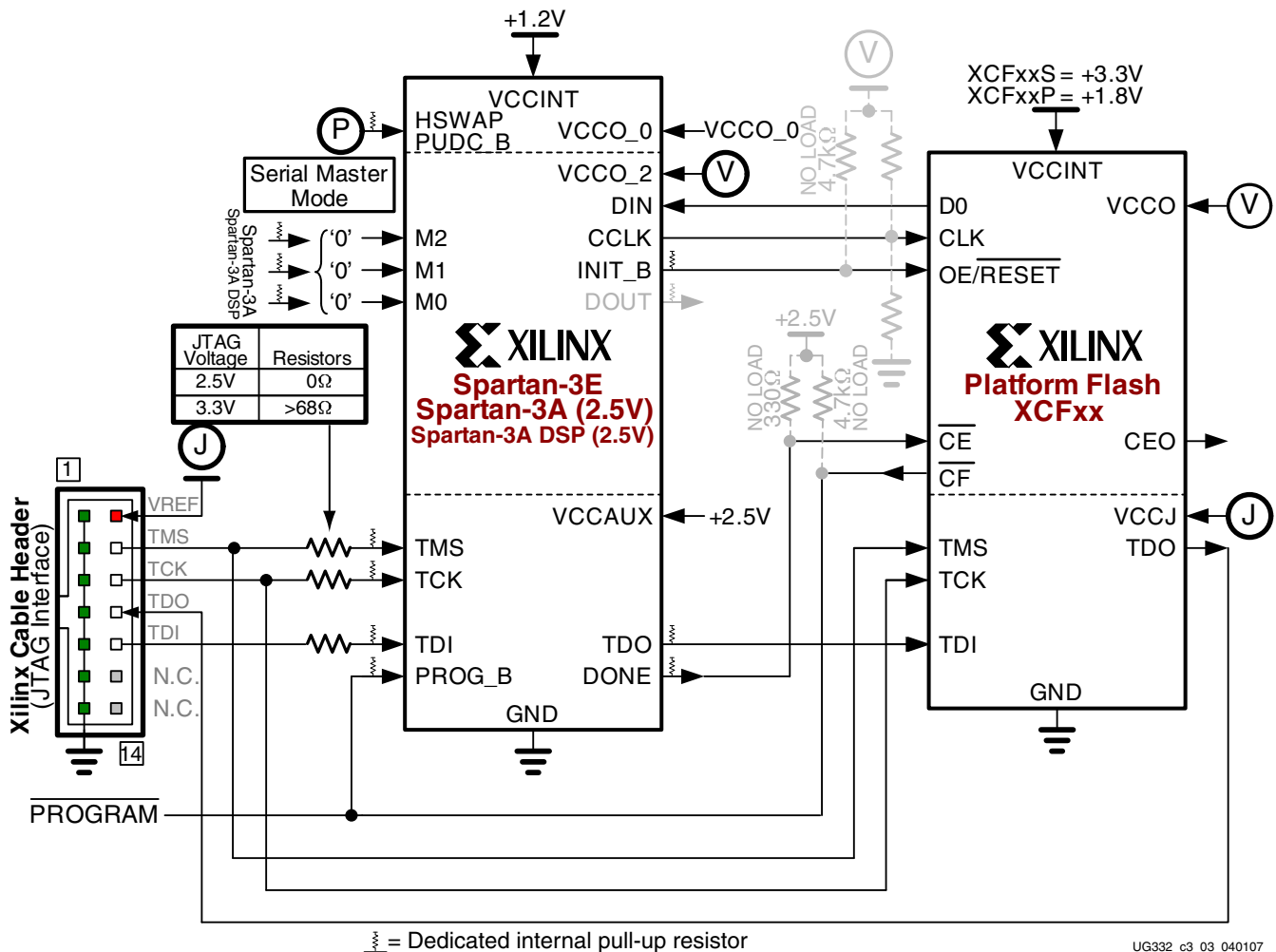


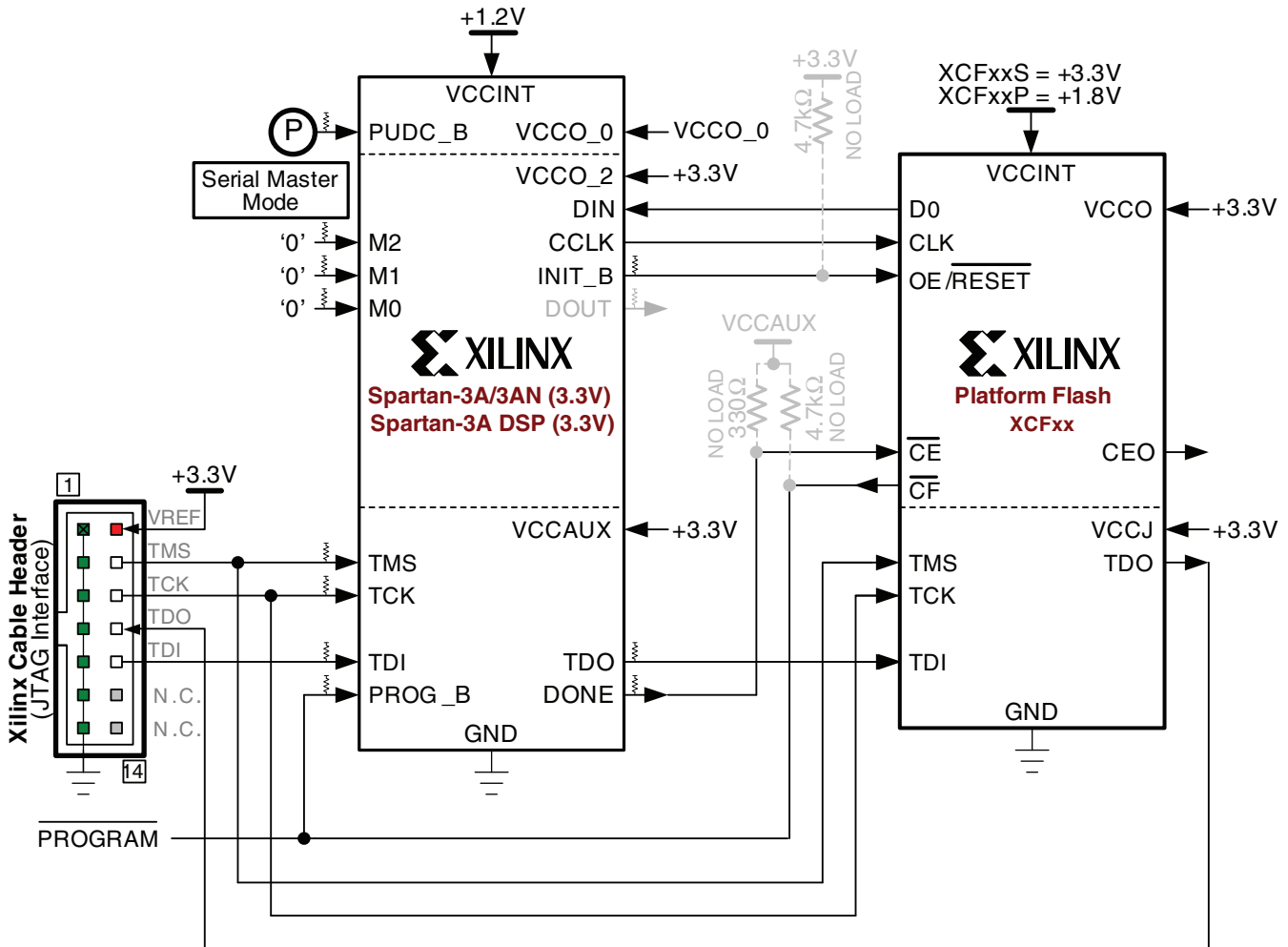
Figure 3-1: Master Serial Mode Using Platform Flash PROM (Spartan-3E or Spartan-3A/3A DSP FPGA, $V_{CCAUX} = 2.5V$)

Table 3-1: Spartan-3E/Spartan-3A/3A DSP FPGA Connections

FPGA Pin	Platform Flash PROM Pin	Comments
DIN	D0	
CCLK	CLK	Watch signal integrity on this trace. See “CCLK Design Considerations,” page 58.
INIT_B	OE/RESET	FPGA resets PROM during initialization, then enables the PROM data out during configuration.
DONE	CE	FPGA enables PROM during configuration. DONE output powered by FPGA V_{CCAUX} supply.

Table 3-1: Spartan-3E/Spartan-3A/3A DSP FPGA Connections

FPGA Pin	Platform Flash PROM Pin	Comments
PROG_B	\overline{CF}	
VCCO_2	V _{CCO}	Spartan-3E FPGA: 1.8V, 2.5V, or 3.3V Spartan-3A/3A DSP FPGA: 2.5V or 3.3V (not 1.8V due to V _{CCO2T} , which does not apply after configuration)
	VCCJ	PROM JTAG output voltage. If 3.3V, then protect the FPGA JTAG inputs with current-limiting resistors (>68Ω)



= Dedicated internal pull-up resistor

UG332_c3_15_052107

Figure 3-2: Master Serial Mode Using Platform Flash PROM (Extended Spartan-3A Family FPGA, V_{CCAUX} = 3.3V)

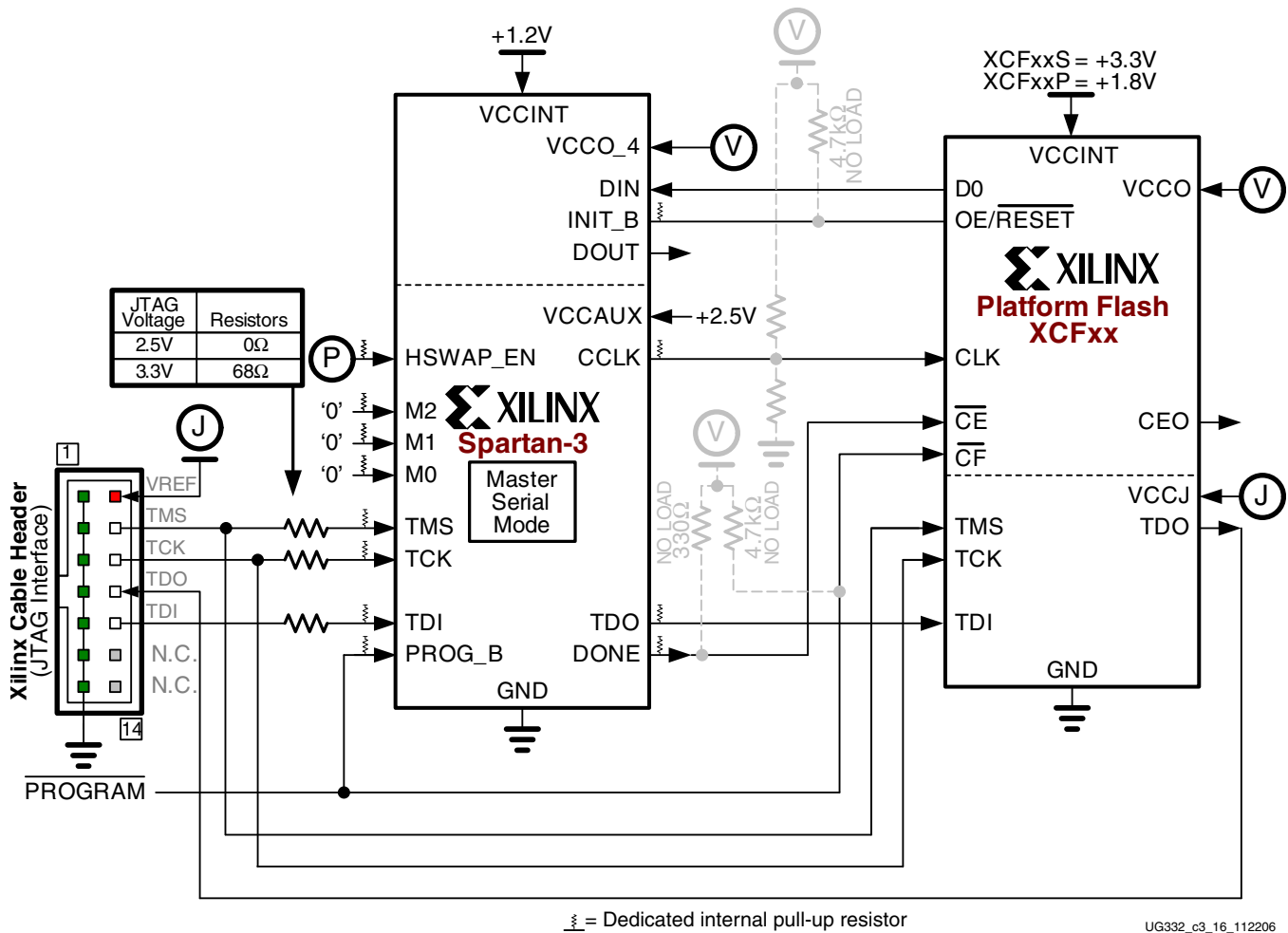


Figure 3-3: Master Serial Mode Using Platform Flash PROM (Spartan-3 FPGA)

Table 3-2: Spartan-3 FPGA Connections to Platform Flash PROM

FPGA Pin	Platform Flash PROM Pin	Comments
DIN	D0	
CCLK	CLK	Watch signal integrity on this trace. See “CCLK Design Considerations,” page 58. CCLK output powered by FPGA’s VCCAUX supply
INIT_B	OE/ $\overline{\text{RESET}}$	FPGA resets PROM during initialization, then enables the PROM’s data out during configuration.
DONE	$\overline{\text{CE}}$	FPGA enables PROM during configuration. DONE output powered by FPGA’s VCCAUX supply.
PROG_B	$\overline{\text{CF}}$	
VCCO_4	VCCO	1.8V, 2.5V, or 3.3V
	VCCJ	PROM’s JTAG output voltage. If 3.3V, then protect the FPGA JTAG inputs with current-limiting resistors (>68Ω)

Master Serial Mode Connections

Table 3-3 lists the various FPGA pins involved in Master Serial mode configuration.

Table 3-3: Master Serial Configuration Mode Connections

Pin Name	FPGA Direction	Description	During Configuration	After Configuration
Spartan-3E FPGA: HSWAP Extended Spartan-3A FPGA: PUDC_B Spartan-3 FPGA: HSWAP_EN (P)	Input	User I/O Pull-Up Control. When Low during configuration, enables pull-up resistors in all I/O pins to respective I/O bank V _{CCO} input. 0: Pull-ups during configuration 1: No pull-ups	Drive at valid logic level throughout configuration.	Spartan-3: Dedicated pin (don't care after configuration) Extended Spartan-3A: User I/O
M[2:0]	Input	Mode Select. Selects the FPGA configuration mode.	M2 = 0, M1 = 0, M0 = 0. Sampled when INIT_B goes High.	User I/O
DIN	Input	Serial Data Input.	Receives serial data from PROM's D0 output.	User I/O
CCLK	Output	Configuration Clock. Generated by FPGA internal oscillator. Frequency controlled by <i>ConfigRate</i> bitstream generator option. If CCLK PCB trace is long or has multiple connections, terminate this output to maintain signal integrity.	Drives PROM's CLK clock input.	Spartan-3: Dedicated pin. Spartan-3E Extended Spartan-3A: User I/O. Drive High or Low if not used.
DOUT	Output	Serial Data Output.	Not used in single-FPGA designs; DOUT is pulled up, not actively driving. In a daisy-chain configuration, this pin connects to DIN input of the next FPGA in the chain. See Figure 3-4, page 87.	User I/O
INIT_B	Open-drain bidirectional I/O	Initialization Indicator. Active Low. Goes Low at start of configuration during Initialization memory clearing process. Released at end of memory clearing, when mode select pins are sampled.	Connects to PROM's OE/RESET input. FPGA clears PROM's address counter at start of configuration, enables outputs during configuration. PROM also holds FPGA in Initialization state until PROM reaches Power-On Reset (POR) state. If CRC error detected during configuration, FPGA drives INIT_B Low.	User I/O. If unused in the application, drive INIT_B High to avoid a floating value. See INIT_B "After Configuration".

Table 3-3: Master Serial Configuration Mode Connections (Cont'd)

Pin Name	FPGA Direction	Description	During Configuration	After Configuration
DONE	Open-drain bidirectional I/O	FPGA Configuration Done. Low during configuration. Goes High when FPGA successfully completes configuration.	Connects to PROM's chip-enable (\overline{CE}) input. Enables PROM during configuration. Disables PROM after configuration.	When High, indicates that the FPGA successfully configured.
PROG_B	Input	Program FPGA. Active Low. When asserted Low for 500 ns or longer, forces the FPGA to restart its configuration process by clearing configuration memory and resetting the DONE and INIT_B pins once PROG_B returns High.	Must be High during configuration to allow configuration to start. Connects to PROM's \overline{CF} pin, allowing JTAG PROM programming algorithm to reprogram the FPGA.	Drive PROG_B Low and release to reprogram FPGA.

Voltage Compatibility

Platform Flash PROM

The Platform Flash PROM V_{CCINT} supply must be either 3.3V for the serial XCFxxS Platform Flash PROMs or 1.8V for the serial/parallel XCFxxP PROMs.

FPGA

Spartan-3E and Spartan-3A/3A DSP FPGAs with V_{CCAUX} at 2.5V

The Spartan-3E or Spartan-3A/3A DSP FPGA V_{CCO_2} supply input and the Platform Flash PROM V_{CCO} supply input must be the same voltage. A 2.5V-only interface is easiest as all signals are the same voltage. A 3.3V interface is also supported but the FPGA **PROG_B** and **DONE** pins require special attention as they are powered by the FPGA V_{CCAUX} supply, nominally 2.5V. For Spartan-3E FPGAs see application note [XAPP453: The 3.3V Configuration of Spartan-3 FPGAs](#), and for Spartan-3A/3A DSP FPGAs see application note [XAPP459: Interfacing Large-Swing Signals](#) for additional information.

Spartan-3 FPGAs

The Spartan-3 FPGA's V_{CCO_4} supply input and the Platform Flash PROM V_{CCO} supply input must be the same voltage. A 2.5V-only interface is easiest as all signals are the same voltage. A 3.3V interface is also supported but the FPGA **PROG_B**, **DONE**, and **CCLK** pins require special attention as they are powered by the FPGA V_{CCAUX} supply, nominally 2.5V. See application note [XAPP453: The 3.3V Configuration of Spartan-3 FPGAs](#) for additional information.

JTAG Interface

If the Platform Flash PROM is the last device in the chain, then the JTAG interface voltage is easily controlled by the PROM's **VCCJ** supply. If the FPGA's V_{CCAUX} supply is 2.5V and the JTAG chain is also 2.5V, the interface is simple. To create a 3.3V JTAG interface, even when the FPGA's V_{CCAUX} supply is 2.5V, connect **VCCJ** to 3.3V and provide current-limiting resistors on the FPGA's TDI, TMS, and TCK JTAG inputs.

For Spartan-3A/3A DSP FPGA, the V_{CCAUX} supply can be either 2.5V or 3.3V. If V_{CCAUX} is 3.3V, then a 3.3V JTAG interface is also easy. No current-limiting resistors are required.

See also “JTAG Cable Voltage Compatibility,” page 198.

Supported Platform Flash PROMs

Table 3-4 shows the smallest available Platform Flash PROM to program one Spartan-3 generation FPGA. A multiple-FPGA daisy-chain application requires a Platform Flash PROM large enough to contain the sum of the various FPGA bitstream sizes.

Table 3-4: Number of Bits to Program a Spartan-3 Generation FPGA and Smallest Platform Flash PROM

Family	FPGA	Number of Configuration Bits	Smallest Possible Platform Flash PROM
Spartan-3A (Spartan-3AN)	XC3S50A	437,312	XCF01S
	XC3S200A	1,196,128	XCF02S
	XC3S400A	1,886,560	XCF02S
	XC3S700A	2,732,640	XCF04S
	XC3S1400A	4,755,296	XCF08P or XCF04S + XCF02S
Spartan-3A DSP	XC3SD1800A	8,197,280	XCF08P or two XCF04S PROMs
	XC3SD3400A	11,718,304	XCF16P
Spartan-3E	XC3S100E	581,344	XCF01S
	XC3S250E	1,353,728	XCF02S
	XC3S500E	2,270,208	XCF04S
	XC3S1200E	3,841,184	XCF04S
	XC3S1600E	5,969,696	XCF08P or XCF04S + XCF02S
Spartan-3	XC3S50	439,264	XCF01S
	XC3S200	1,047,616	XCF01S
	XC3S400	1,699,136	XCF02S
	XC3S1000	3,223,488	XCF04S
	XC3S1500	5,214,784	XCF08P or XCF04S + XCF02S
	XC3S2000	7,673,024	XCF08P or 2 x XCF04S
	XC3S4000	11,316,864	XCF16P
	XC3S5000	13,271,936	XCF16P

There are two possible design solutions for FPGA designs that require 8 Mbit PROMs: use either a single 8 Mbit XCF08P parallel/serial PROM or two cascaded XCFxxS serial

PROMs as listed in [Table 3-4](#). The two XCFxxS PROMs have a 3.3V V_{CCINT} supply while the XCF08P requires a 1.8V V_{CCINT} supply. If the board does not already have a 1.8V supply available, the two cascaded XCFxxS PROM solution is recommended.

CCLK Frequency

In Master Serial mode, the FPGA's internal oscillator generates the configuration clock frequency. The FPGA provides this clock on its **CCLK** output pin, driving the PROM's CLK input pin. The FPGA starts configuration at its lowest frequency and increases its frequency for the remainder of the configuration process if so specified in the configuration bitstream. The maximum frequency is specified using the ConfigRate bitstream generator option. [Table 3-5](#) shows the maximum **ConfigRate** settings, approximately equal to the frequency measured in MHz, for various Platform Flash PROMs and I/O voltages. These values are determined using the minimum **CCLK** period from the appropriate Spartan-3E or Extended Spartan-3A family data sheet. The maximum **ConfigRate** for the serial XCFxxS PROMs is reduced at 1.8V. Extended Spartan-3A family FPGAs do not support a 1.8V configuration interface due to their higher VCCO_2 Power-On Reset voltage threshold. See "Power-On Reset (POR)," [page 240](#).

Table 3-5: Maximum ConfigRate Settings Using Platform Flash (Serial Mode, Commercial Range)

Platform Flash Part Number	I/O Voltage (V_{CCO_2} , V_{CCO})	Spartan-3E ConfigRate Setting	Extended Spartan-3A Family ConfigRate Setting
XCF01S	3.3V or 2.5V	25	33
XCF02S	1.8V	12	N/A
XCF04S			
XCF08P	3.3V or 2.5V	25	44
XCF16P	1.8V		N/A
XCF32P			

Daisy-Chained Configuration

If the application requires multiple FPGAs, each with different configurations, then configure the FPGAs using a daisy chain, as shown in [Figure 3-4, page 87](#). Use Master Serial mode ($M[2:0] = <0:0:0>$) for the FPGA connected to the Platform Flash PROM and Slave Serial mode ($M[2:0] = <1:1:1>$) for all other FPGAs in the daisy chain. After the master FPGA—the FPGA on the left in the diagram—finishes loading its configuration data from the Platform Flash, the master device supplies data using its **DOUT** output pin to the next device in the daisy chain, on the falling **CCLK** edge.

Also, to successfully configure a daisy chain, the **GTS_cycle** bitstream option must be set to a Startup phase after the **DONE_cycle** setting for all FPGAs in the chain. This is the software default setting. Optionally, set **GTS_cycle:Done**.

Ganged or Broadside Configuration

"Daisy-Chained Configuration" is designed to load multiple FPGAs, each with a different design and typically of different array size. However, some applications include multiple, identical FPGAs, all programmed with the same bitstream. Instead of daisy chaining the FPGAs and storing multiple copies of the same bitstream, "Ganged or Broadside

JTAG Interface

Spartan-3 generation FPGAs and the Platform Flash PROMs both have a four-wire IEEE 1149.1/1532 JTAG port. Both the FPGA and the PROM share the JTAG **TCK** clock input and the **TMS** mode select input. The devices may connect in either order on the JTAG chain with the TDO output of one device feeding the TDI input of the following device in the chain. The TDO output of the last device in the JTAG chain drives the JTAG connector.

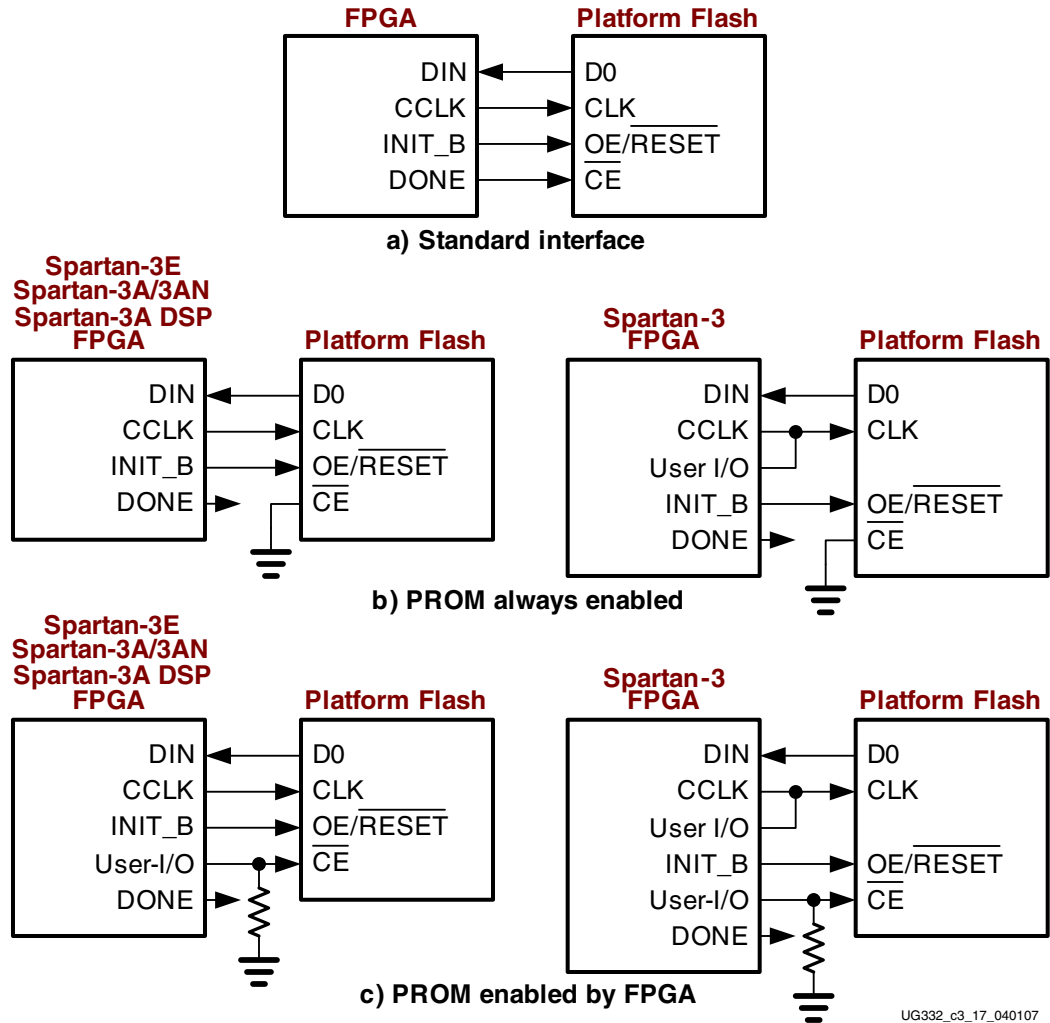
The JTAG interface on the FPGA is powered by the V_{CCAUX} supply. Consequently, the PROM's VCCJ supply input must also be 2.5V. To create a 3.3V JTAG interface, refer to [XAPP453: The 3.3V Configuration of Spartan-3 FPGAs](#) for additional information.

Storing Additional User Data in Platform Flash

Typically, there is some additional space leftover in the Platform Flash after storing the FPGA bitstream. If desired, the application can store additional data in the Platform Flash PROM and make it available to the FPGA after configuration.

The FPGA application does not have easy write-access to the PROM but read-access is relatively simple, as described in the referenced application notes below. For applications that also require easy write-access, consider using the Master SPI configuration interface, described in [Chapter 4, "Master SPI Mode"](#).

Use the available space in the Platform Flash PROM, or even the next larger PROM size, to hold additional nonvolatile application data such as MicroBlaze[®] processor code, or other user data such as serial numbers and Ethernet MAC IDs. Using a MicroBlaze application as an example, the FPGA configures from the Platform Flash PROM. Then using FPGA logic after configuration, the FPGA copies MicroBlaze code from Platform Flash into external DDR SDRAM for code execution, providing simple and cost-effective code shadowing.



UG332_c3_17_040107

Figure 3-6: Various Methods to Use Platform Flash PROM after Configuration

A few simple modifications are required to the standard interface. As shown in Figure 3-6a, the FPGA uses its DONE output to enable the Platform Flash PROM \overline{CE} input. However, once configured, the FPGA cannot re-enable the PROM because the DONE is a dedicated pin and the FPGA application cannot control it.

The simplest solution, shown in Figure 3-6b, is to connect the PROM's \overline{CE} input to ground. The PROM consumes slight more power if constantly enabled, but then the FPGA has direct access. On Spartan-3 FPGAs, the CCLK pin is a dedicated pin. To control the PROM, use an FPGA I/O in parallel with CCLK. Also be sure to set the *CclkPin:Pullnone* bitstream option.

Figure 3-6c shown an alternative solution. In this case, connect the PROM's \overline{CE} input to an FPGA I/O pin. The FPGA pin has a sufficiently large pull-down resistor to guarantee that \overline{CE} is Low during configuration. The exact size of the pull-down resistor depends on whether pull-up resistors are enabled during configuration and the I/O standard used in the application. See Table 2-15, page 66 for pull-down resistor values. After configuration, the FPGA can selective enable the PROM by driving the associated I/O pin High or Low.

See the following application notes for specific details on how to implement such an interface.

- XAPP482: MicroBlaze Platform Flash/PROM Boot Loader and User Data Storage http://www.xilinx.com/support/documentation/application_notes/xapp482.pdf
- XAPP694: Reading User Data from Configuration PROMs http://www.xilinx.com/support/documentation/application_notes/xapp694.pdf

Generating the Bitstream for a Master Serial Configuration

To create the FPGA bitstream for a Master Serial mode configuration, follow the steps outlined in “[Setting Bitstream Options, Generating an FPGA Bitstream](#),” page 42. For an FPGA configured in Master SPI mode, set the following bitstream generator options.

ConfigRate: CCLK Frequency

Set the *ConfigRate* option as described in “[CCLK Frequency](#),” page 86. Using the ISE® software Project Navigator, the Configuration Rate frequency is set in Step 7 in [Figure 1-7](#), page 44.

```
-g ConfigRate:25
```

StartupClk: CCLK

By default, the configuration Startup clock source is the internally generated CCLK. Keep the *StartupClk* bitstream generation option, shown as Step 13 in [Figure 1-8](#), page 45.

```
-g StartupClk:Cclk
```

DriveDone: Actively Drive DONE Pin

In a single FPGA design or for the Master FPGA in a multi-FPGA daisy chain, set the FPGA to actively drive the DONE pin after successfully completing the configuration process. Using ISE Project Navigator, check the **Drive Done Pin High** option, shown as Step 16 in [Figure 1-8](#), page 45.

```
-g DriveDone:Yes
```

GTS_cycle: Global Three-State Release Timing for Daisy Chains

If creating a multi-FPGA daisy chain, set the *GTS_cycle* option to be later than the *DONE_cycle* setting, which is the default setting for both. Alternatively, set *GTS_cycle:Done*. From ISE Project Navigator, the *GTS_cycle* setting is the **Enable Outputs (Output Events)** option, shown as Step 14 in [Figure 1-8](#), page 45.

Preparing a Platform Flash PROM File

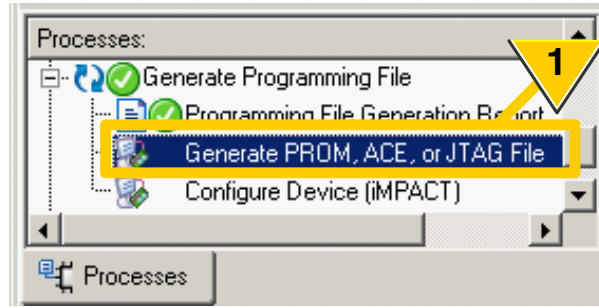
This section provides guidelines to create PROM files for Platform Flash PROM memories.

The Xilinx software tools, “**iMPACT**” or PROMGen, generate PROM files from the FPGA bitstream or bitstreams.

iMPACT

The following steps graphically describe how to create a PROM file using iMPACT from within the ISE Project Navigator.

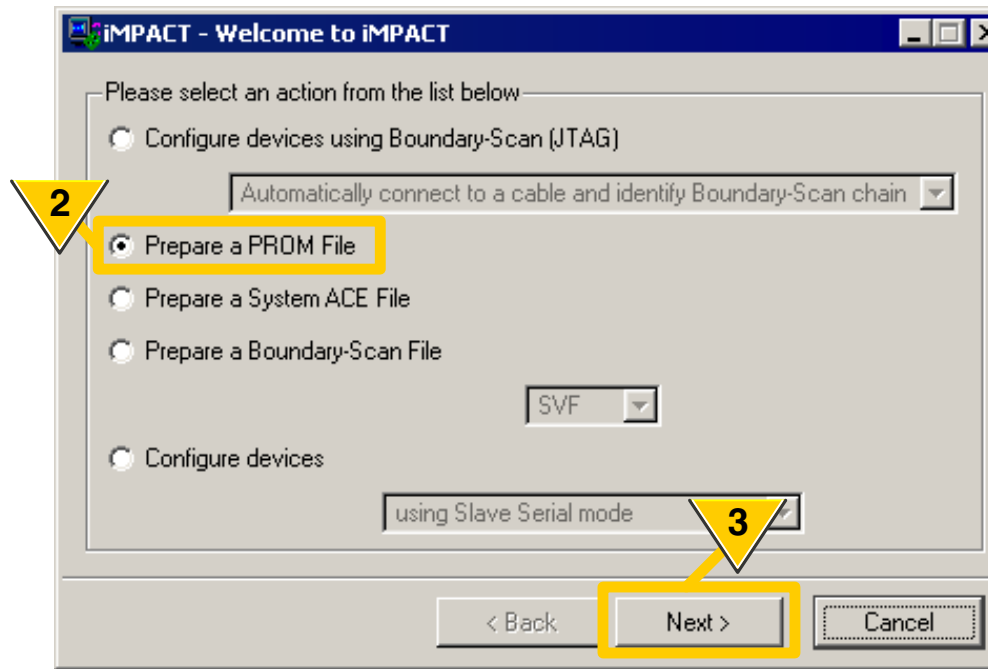
1. From within the ISE Project Navigator, double-click **Generate PROM, ACE, or JTAG File** from within the Process pane, as shown in [Figure 3-7](#).



UG332_c4_10_110206

Figure 3-7: Double-click **Generate PROM, ACE or JTAG File**

2. As shown in [Figure 3-8](#), select **Prepare a PROM File**.



UG332_c4_11_19

Figure 3-8: **Prepare a PROM File**

3. Click **Next**.

- As shown in [Figure 3-9](#), format the FPGA bitstream or bitstreams for a Xilinx PROM.

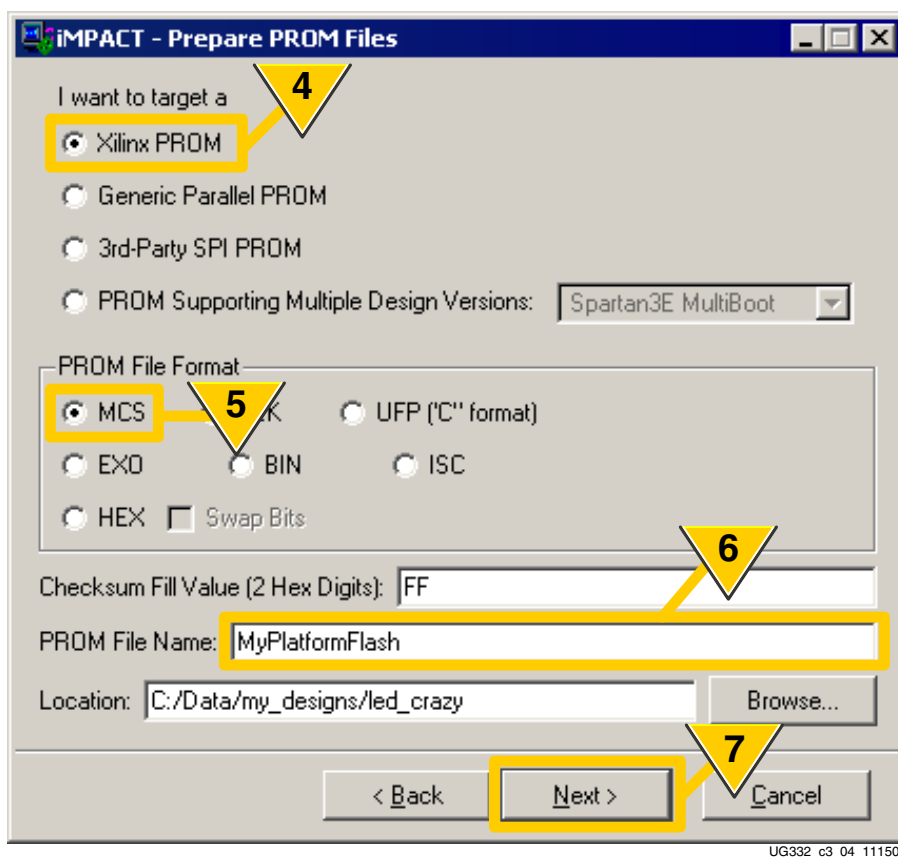


Figure 3-9: Set Options for Xilinx Platform Flash PROM

- Select a **PROM File Format**.
- Enter a **PROM File Name**.
- Click **Next**.
- As shown in [Figure 3-10](#), select the xcf (Platform Flash PROM) family from the drop-list.

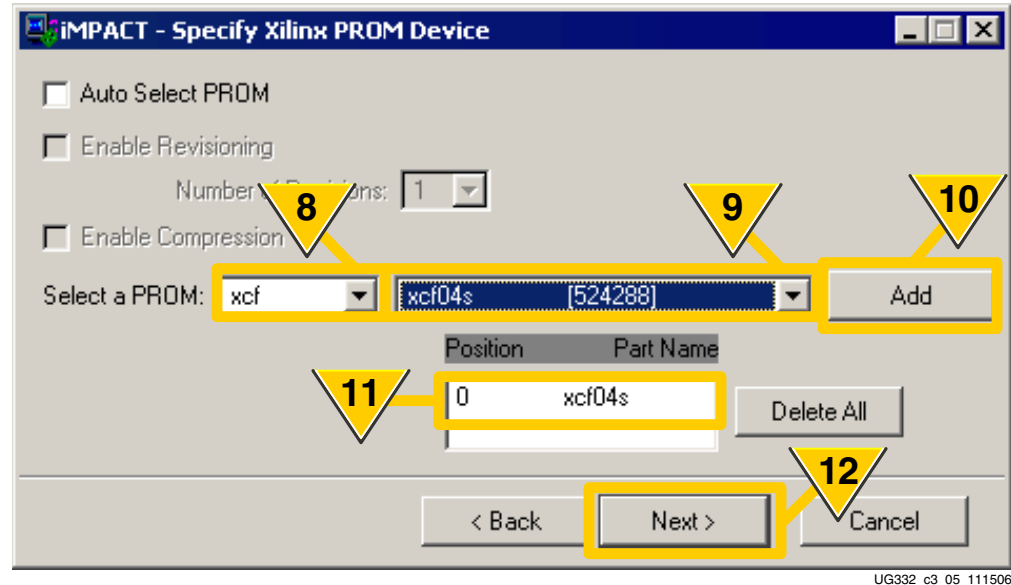


Figure 3-10: Select Platform Flash PROM

9. Select the desired Platform Flash part number. The example in Figure 3-10 shows an XCF04 PROM, which stores up to 4 Mbits, or 524,288 bytes.
10. Click **Add**. This example assumes that the FPGA is connected to a single Platform Flash PROM. However, multiple Platform Flash PROMs can also be cascaded to create a larger memory. If the application cascaded multiple PROMs, then click the **Add** button to include additional PROMs.
11. For a design that uses a single Platform Flash PROM, the PROM also is located in position 0. If the application used multiple, cascaded PROMs, each PROM part name and position would be listed.
12. Click **Next**.
13. As shown in Figure 3-11, review that the settings are correct to format the Platform Flash PROM. Click **Finish** to confirm the settings or **Back** to change the settings.

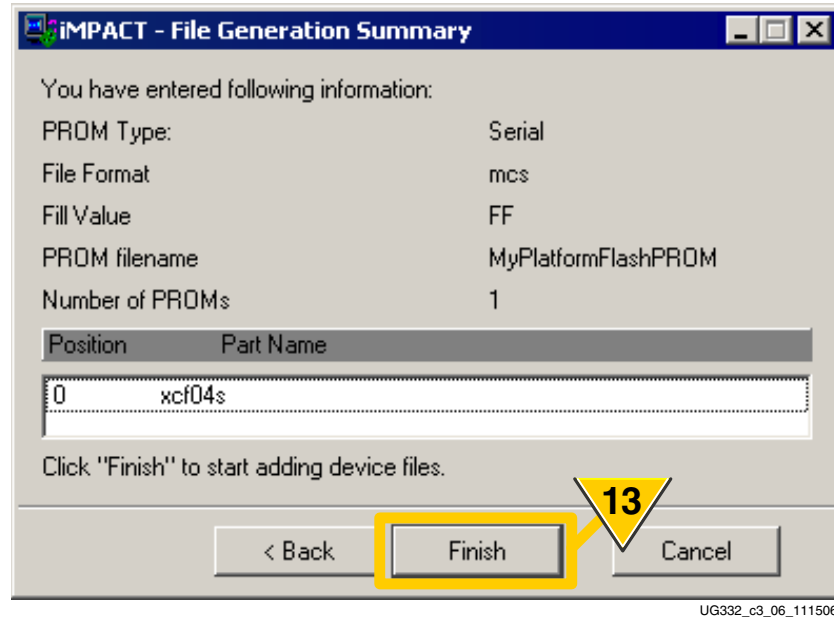


Figure 3-11: Review PROM Formatting Settings

14. As shown in Figure 3-12, click **OK** to start adding bitstream files.

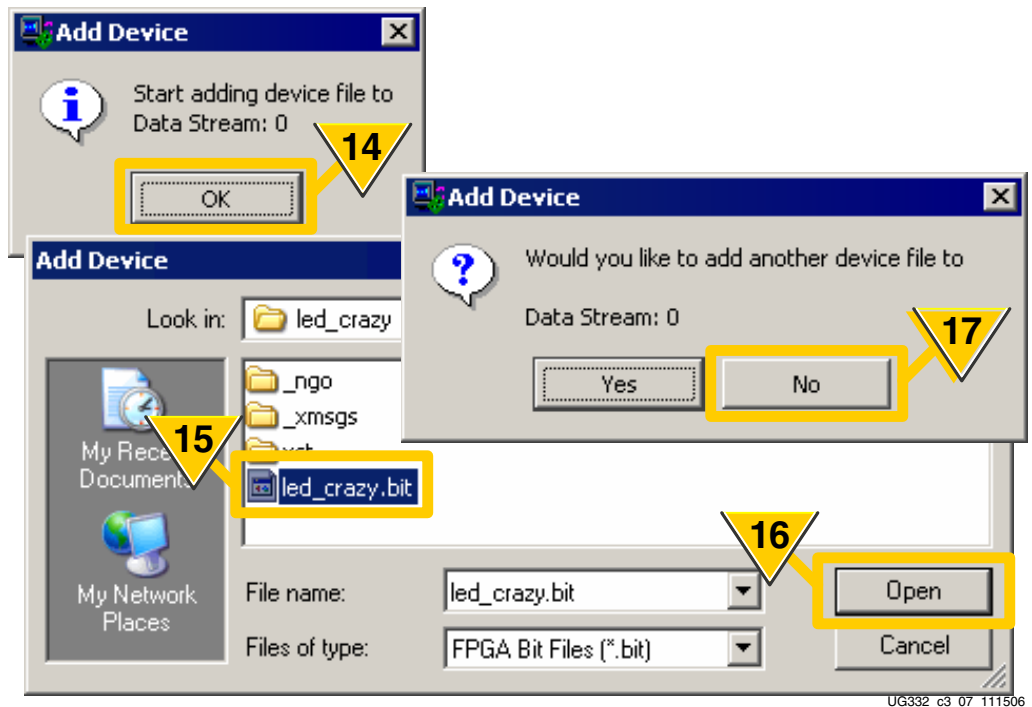


Figure 3-12: Add FPGA Configuration Bitstream File(s)

15. Locate and select the desired FPGA bitstream.
16. Click **Open**.

17. Click **No**. This example assumes that the Platform Flash PROM holds only a single FPGA bitstream. If creating a multi-FPGA configuration daisy chain, click **Yes** and select additional FPGA bitstreams.
18. As shown in [Figure 3-13](#), the iMPACT software graphically displays the Platform Flash PROM and associated FPGA bitstream(s).

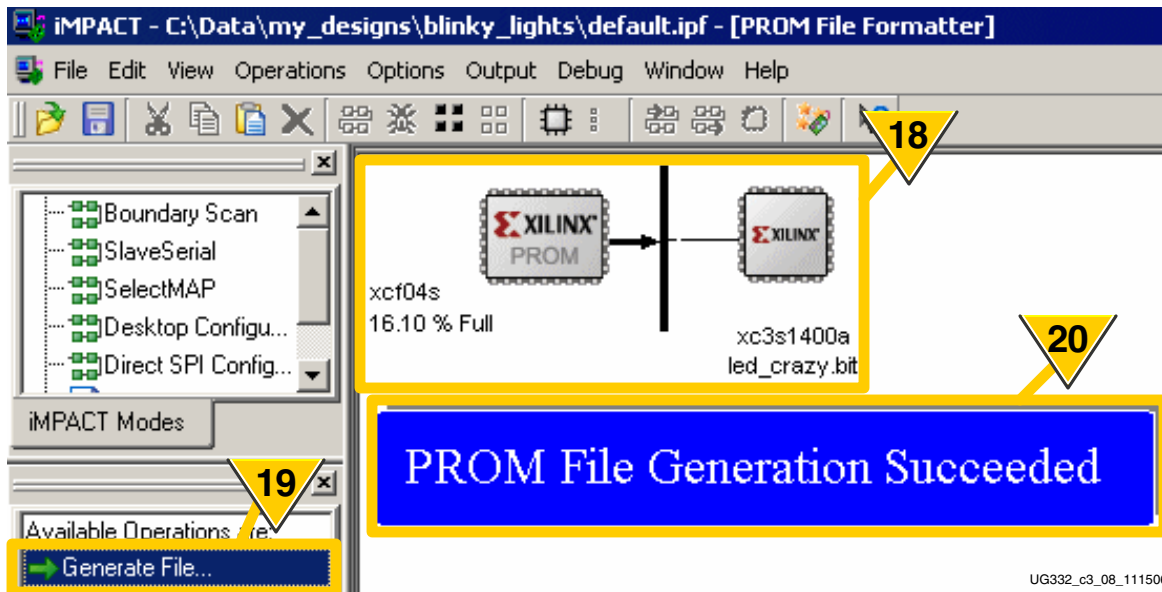


Figure 3-13: **Generate PROM File**

19. Click **Generate File**.
20. The iMPACT software indicates when the PROM file is successfully created.

Platform Flash In-System Programming via JTAG using iMPACT

Both the FPGA and the Platform Flash PROM are in-system programmable via the JTAG chain. Download support for prototyping purposes is provided by the Xilinx iMPACT programming software and the associated Xilinx Parallel Cable IV, or Platform Cable USB II programming cables.

Prepare Board for Programming

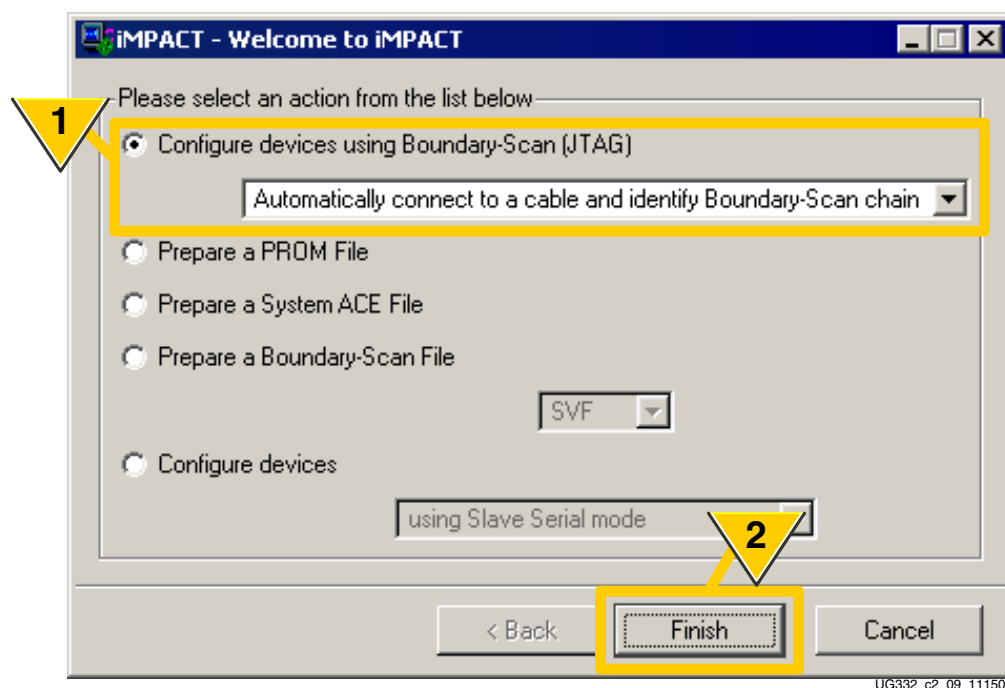
Before attempting to program the Platform Flash PROM, complete the following steps.

1. Ensure that the board is powered.
2. Ensure that the programming cable is properly connected both the board and to the computer or workstation.

Programming via iMPACT

The following steps describe how to program a Platform Flash PROM using the iMPACT software and a Xilinx programming cable.

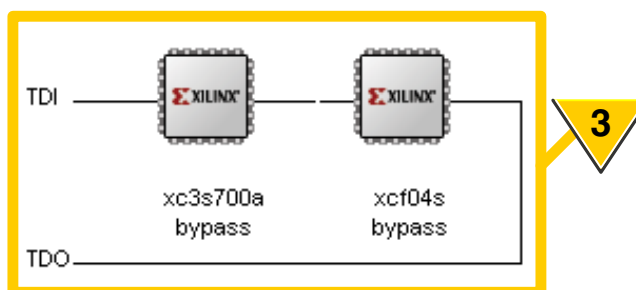
1. Click **Configure devices using Boundary-Scan (JTAG)** from within iMPACT, as shown in Figure 3-14. If the **Automatically connect...** option is selected, iMPACT will query the devices in the JTAG chain and automatically detect the chain topology.



UG332_c2_09_111506

Figure 3-14: Program Platform Flash PROM using JTAG

2. Click **Finish**.
3. As shown in Figure 3-15, the iMPACT software automatically detects the JTAG chain, if so enabled. This example application is similar to that shown in Figure 3-1. The FPGA is an XC3S700A, followed in the chain by an XCF04S Platform Flash PROM.



UG332_c3_10_111506

Figure 3-15: iMPACT Automatically Detects JTAG Chain

4. In this example, the XC3S700A precedes the XCF04S Platform Flash PROM in the chain. The FPGA does not need to be programmed in order to program the Platform Flash PROM. The iMPACT software prompts for the FPGA bitstream, as shown in Figure 3-16. Click **Bypass** to skip programming the FPGA.

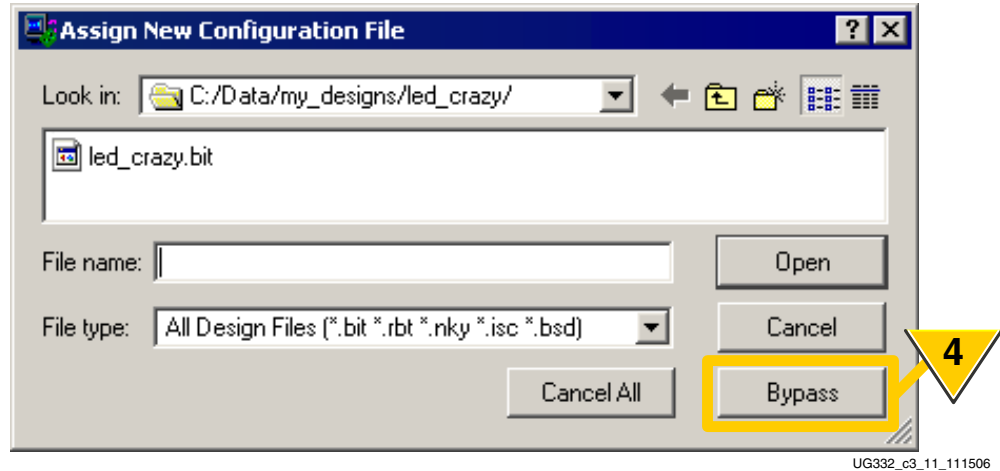


Figure 3-16: Bypass Programming the FPGA

5. As shown in Figure 3-17, select the PROM data file to be programmed to the Platform Flash PROM.

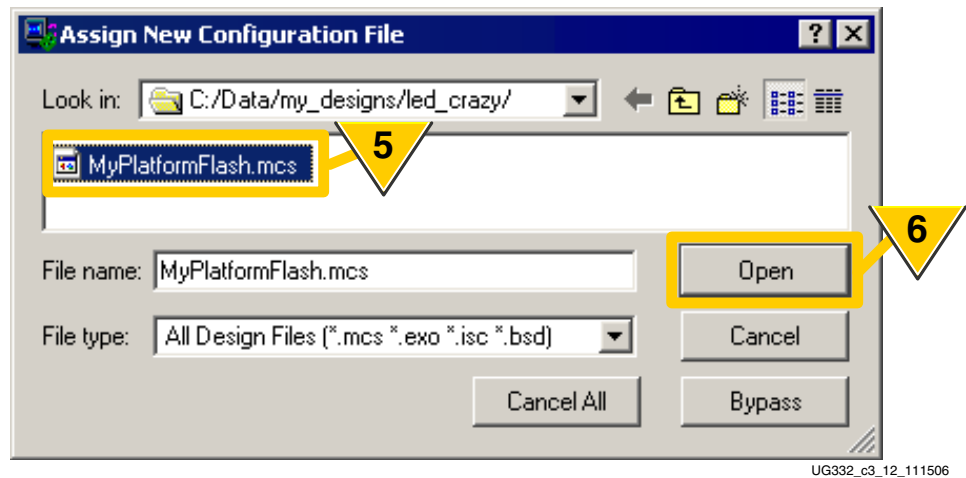


Figure 3-17: Select the Platform Flash Programming File

6. Click **Open**.
7. As shown in Figure 3-18, the iMPACT software updates the screen image, showing the files to be loaded to each device in the JTAG chain. To program the Platform Flash PROM, first click to highlight the XCF04S PROM.

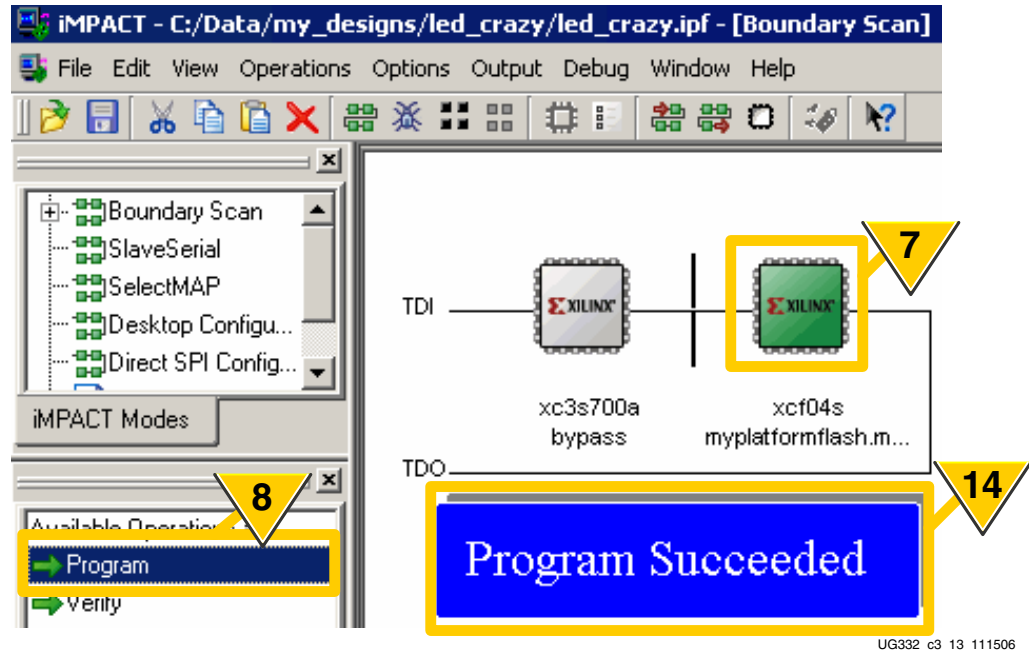


Figure 3-18: Program the Platform Flash PROM

8. Double-click **Program**.
9. Click **Programming Properties**, as shown in Figure 3-19.

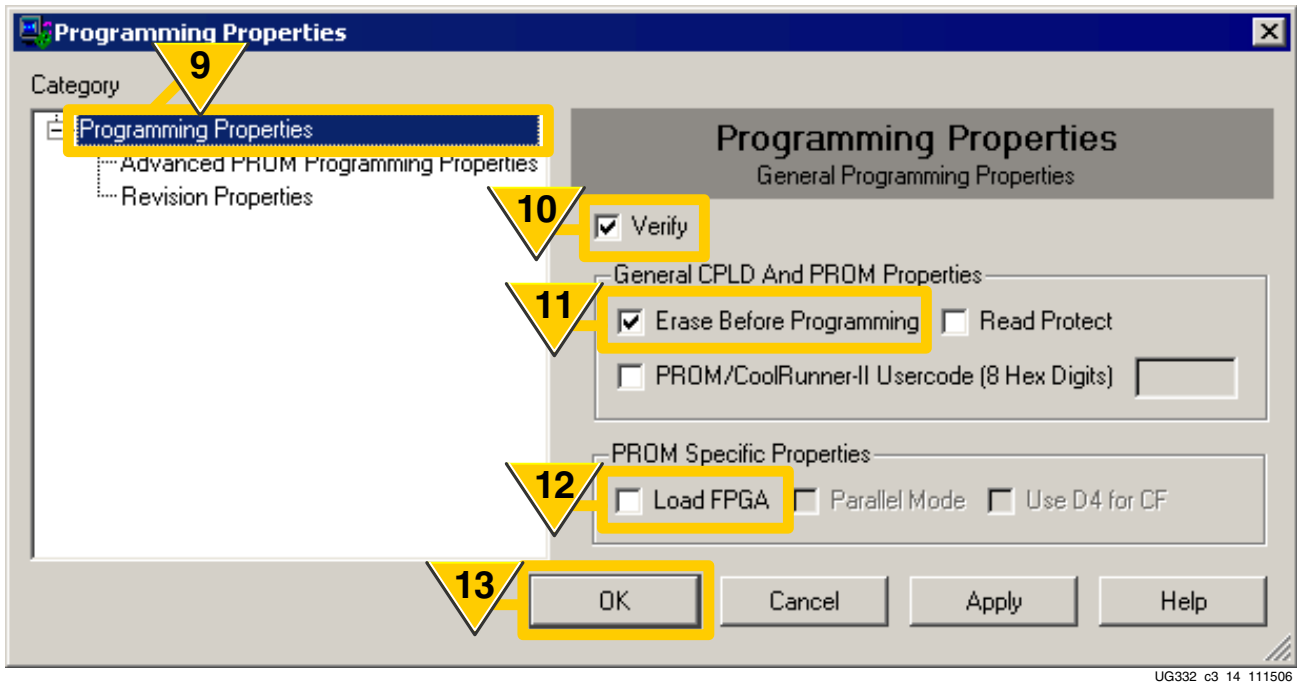


Figure 3-19: PROM Programming Options

10. Check **Verify**. Unchecking Verify will reduce programming but iMPACT can only guarantee correct programming on a verified PROM.

11. Check **Erase Before Programming**. Required for reprogramming. Unchecking the Erase option reduces programming time for a blank device.
12. Check **Load FPGA** to force the FPGA to automatically reconfigure with the new PROM data after PROM programming is complete.
13. Click **OK**.
14. The iMPACT software indicates successful programming, as shown in [Figure 3-18](#).

Production Programmers

The Xilinx Platform Flash PROMs are supported by a variety of third-party production programmers. These programmers are the best option for high-volume applications and many offer gang-programming options.

[Table 3-6](#) provides links to vendors that provide Platform Flash programming support. The links indicate the specific programmer model numbers, software versions, and any programming adapters required.

Table 3-6: Xilinx Platform Flash Production Programmers

Platform Flash Family	Part Numbers	Production Programmers
XCFxxS	XCF01S XCF02S XCF04S	www.xilinx.com/support/programr/dev_sup.htm#XCF00SP
XCFxxP	XCF08P XCF16P XCF32P	www.xilinx.com/support/programr/dev_sup.htm#XCF00SP

Additional Information

- **DS123: Platform Flash In-System Programmable Configuration PROMs**
www.xilinx.com/support/documentation/data_sheets/ds123.pdf

Master SPI Mode

The SPI serial Flash configuration mode is ideal for applications with the following attributes.

- SPI Flash PROMs are already being used in the system.
- The FPGA application needs to store data in nonvolatile memory or to access data from randomly-accessible, byte-addressable, nonvolatile memory.
- High-volume “consumer” applications with a production run of about a few years or less. For embedded applications with a five year or longer production lifetime, also consider Master Serial mode using Xilinx® Platform Flash, which has a longer, more stable supply lifetime than commodity Flash.

In Master SPI mode ($M[2:0] = <0:0:1>$), the Spartan®-3E or Extended Spartan-3A family FPGA configures itself from an attached industry-standard SPI serial Flash PROM, as illustrated in [Figure 4-1](#) and [Figure 4-2](#). The figure shows optional components in gray and designated “NO LOAD”. The FPGA supplies the **CCLK** output clock from its internal oscillator and drives the clock input of the attached SPI Flash PROM.

More information on configuration from SPI Flash PROMs can be found in the following application note.

- **XAPP951: Configuring Xilinx FPGAs with SPI Serial Flash**
http://www.xilinx.com/support/documentation/application_notes/xapp951.pdf

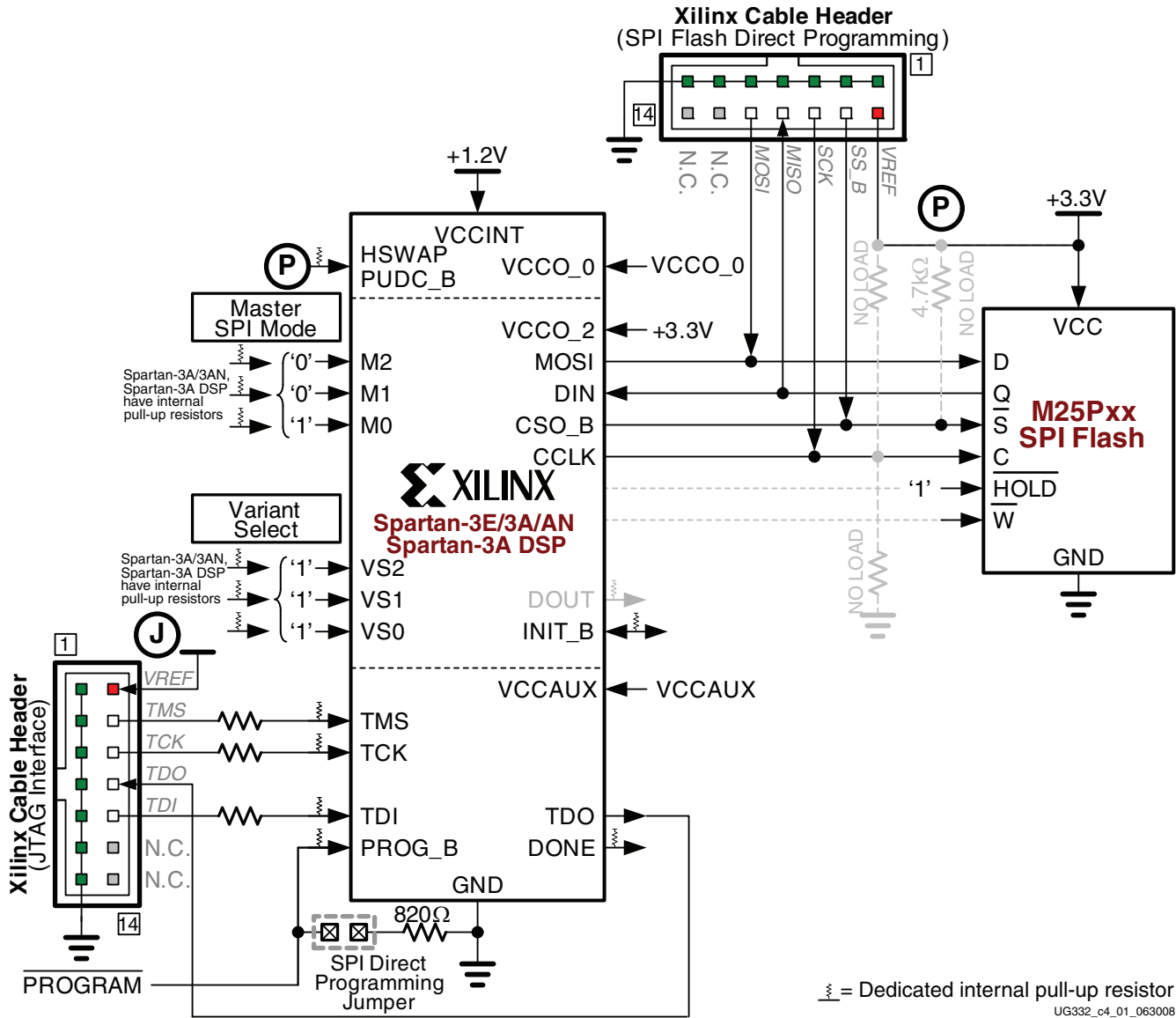
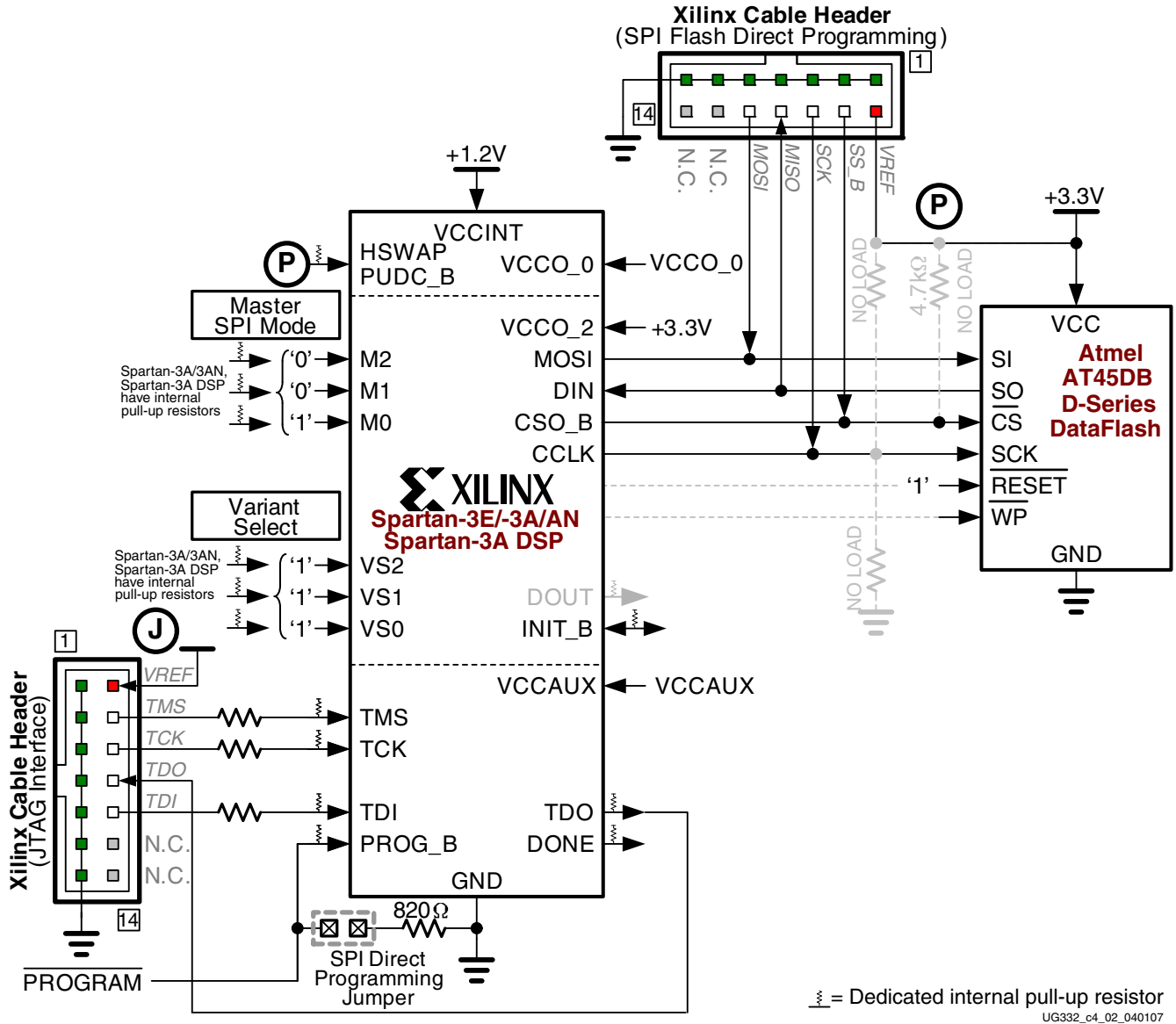


Figure 4-1: SPI Flash Configuration Interface for M25P-compatible Devices

Although SPI is a fairly standard and ubiquitous four-wire interface, various available SPI Flash PROMs use different command protocols. The FPGA's variant select pins, **VS[2:0]**, define how the FPGA communicates with the SPI Flash, including which SPI Flash command the FPGA issues to start the read operation and the number of dummy bytes inserted before the FPGA expects to receive valid data from the SPI Flash. **Table 4-2** shows the available SPI Flash PROMs tested or expected to operate with Spartan-3E/3A FPGAs. Other compatible devices might work but have not been hardware verified by Xilinx. All other **VS[2:0]** values are reserved for future use. Consult the data sheet for the desired SPI Flash device to determine its suitability.

Figure 4-1 shows the general connection diagram for SPI Flash PROMs that support the 0x0B FAST READ commands, which are most modern 25-series PROMs. The example shown is an **STMicro (Numonyx) M25Pxx** PROM.

Figure 4-2 shows the connection diagram for Atmel DataFlash serial PROMs, which also use an SPI-based protocol. Xilinx recommends using 'C'- or 'D'-series DataFlash devices. Figure 4-6, page 119 demonstrates how to configure multiple FPGAs with different configurations, all stored in a single SPI Flash. The diagram uses standard SPI Flash memories but the same general technique applies for Atmel DataFlash.



Note: Grayed out pull-up resistor on CSO_B is only necessary when HSWAP/PUDC_B is High (I/O Pull-Ups Not Enabled)

Figure 4-2: SPI Flash Configuration Interface for Atmel DataFlash Devices

Master SPI Mode Differences between Spartan-3 Generation FPGA Families

The Master SPI configuration mode is available using either the Extended Spartan-3A or Spartan-3E FPGA families. It is not provided on the Spartan-3 FPGA family, as summarized in [Table 4-1](#).

Table 4-1: Master SPI Mode Support within Spartan-3 Generation FPGAs

	Spartan-3 FPGA	Spartan-3E FPGA	Spartan-3A/3AN Spartan-3A DSP FPGA
Supports multi-FPGA daisy-chain configurations	Master SPI mode is not available on Spartan-3 FPGAs	Step 1 only	Yes
Supports MultiBoot configuration		No	Yes
Watchdog Timer retry		No	Yes
CCLK directionality during Master SPI mode		I/O	Output only for improved signal integrity
M[2:0] and VS[2:0] pins have dedicated internal pull-up resistors during configuration		No Optional, controlled by HSWAP	Yes

Choosing a Compatible SPI Serial Flash

The Spartan-3E and Extended Spartan-3A family FPGAs are designed to support a wide range of SPI serial Flash memory devices. [Table 4-2, page 105](#) lists the Xilinx-tested PROMs that have in-system programming support using the iMPACT software. Many other SPI Flash PROMs are designed to be form, fit, and functionally equivalent and are listed in [Table 4-5, page 107](#). The Xilinx ISE[®] software generates compatible programming files but Xilinx has not tested these PROMs for complete compatibility. Similarly, the PROMs listed in [Table 4-5, page 107](#) are not supported by the iMPACT in-system programming software.

The criteria to select an SPI Flash PROM are listed below.

- Ideally, the end application should use a Xilinx-tested SPI PROM, listed in [Table 4-2](#). [Table 4-3, page 105](#) lists the specific SPI Flash PROM part numbers tested and supported within iMPACT for in-system programming using Xilinx programming cables.

Table 4-2: SPI Flash Memory Devices Officially Supported by Xilinx and Programmed Using iMPACT

SPI Flash		Xilinx iMPACT Support	Unique ID	Read Command			Density (bits)								
				Fast Read (0x0B)	Read (0x03)	Read Array (0xE8)									
Vendor	Family		FPGA VS[2:0] Setting			512K	1M	2M	4M	8M	16M	32M	64M	128M	
			1:1:1	1:0:1	1:1:0										
STMicro (Numonyx)	M25P	◆		◆	◆		◆	◆	◆	◆	◆	◆	◆	◆	
	M25PE	◆		◆	◆			◆	◆	◆	◆				
	M45PE	◆		◆	◆			◆	◆	◆	◆				
Atmel	AT45DB D-series	◆	◆	◆	◆			◆	◆	◆	◆	◆	◆		
	AT45DB B-series	◆				◆		◆	◆	◆	◆				

Notes:

1. Xilinx iMPACT Support indicates that Xilinx has physically tested compatibility for these SPI Flash memory devices and provides programming support in the iMPACT programming utility using Xilinx approved JTAG cables. The iMPACT software generates programming information that is compatible with all the devices listed.
2. Unique ID indicates that these SPI Flash memory device have factory-programmed unique identifier bits, useful for protecting FPGA applications or IP cores.

Table 4-3: SPI Serial Flash PROMs Supported by iMPACT

Vendor	STMicro (Numonyx)			Atmel	
Status	Recommended	Supported		Recommended	Supported
Density (bits)	M25Pxx	M25PExx	M45PExx	AT45DBxxxD	AT45DBxxxB
512K	M25P05A				
1M	M25P10A	M25PE10	M45PE10	AT45DB011D	AT45DB011B
2M	M25P20	M25PE20	M45PE20	AT45DB021D	AT45DB021B
4M	M25P40	M25PE40	M45PE40	AT45DB041D	AT45DB041B
8M	M25P80	M25PE80	M45PE80	AT45DB081D	AT45DB081B
16M	M25P16		M45PE16	AT45DB161D	AT45DB161B
32M	M25P32			AT45DB321D AT45DB321C	AT45DB321B
64M	M25P64			AT45DB642D	
128M	M25P128				

- The specific SPI serial memory must support a compatible read command offered by the FPGA. The specific command set is selected by defining the FPGA's VS[2:0] pins before configuration. Table 4-4 lists the commands supported on Spartan-3E and Extended Spartan-3A family FPGAs. The command setting defines which SPI Flash read command that the FPGA issues at the start of configuration, followed by a 24-bit address starting at 0, followed by the number of dummy bits required for the specific command.

- ◆ The **Fast Read** command (command code 0x0B) is supported on modern 25-series SPI serial Flash devices. Set **VS[2:0]** <1:1:1> to use this command. SPI Flash PROMs that support the Fast Read command also support the **Read** command.
- ◆ The **Read** command (command code 0x03) is a legacy command set, offered on all 25-series SPI serial Flash devices. Set **VS[2:0]** <1:0:1> to use this command.
- ◆ The **Read Array** command (command code 0xE8) is offered on all **Atmel** AT45-series DataFlash PROMs. Set **VS[2:0]** <1:1:0> to use this command.
- ◆ Some recent SPI Flash PROMs, like the **Atmel** AT45DB D-series PROMs support all three read commands.

Table 4-4: SPI Read Commands Supported by Spartan-3 Generation FPGAs

VS[2:0] Pins			Read Command	Hexadecimal Command Code	Address Bits	Dummy Bits
VS2	VS1	VS0				
1	1	1	Fast Read	0x0B	24 bits, all zeros	8 bits, all zeros
1	0	1	Read	0x03		None
1	1	0	Read Array	0xE8		32 bits, all zeros
Others			Reserved			

- The specific SPI serial memory must be large enough to contain one or more FPGA bitstreams plus any other nonvolatile memory requirements to support the FPGA application after configuration.
 - ◆ The size of an individual, uncompressed FPGA bitstream is provided in [Table 4-6, page 108](#), although the size requirements might be reduced by using “[Bitstream Format](#),” [page 39](#).
 - ◆ If using MultiBoot on an Extended Spartan-3A family FPGA, add the size of each MultiBoot configuration image. Essentially, it is the same as an individual FPGA image, but MultiBoot allows multiple selectable images within a single FPGA.
 - ◆ Using a daisy-chained configuration scheme, a single SPI Flash PROM can store multiple FPGA bitstreams. Add the bitstream sizes for each FPGA in the daisy chain.
 - ◆ If using the SPI PROM to store MicroBlaze™ code or other nonvolatile data for the FPGA application after configuration, add the sizes of each of these images.
 - ◆ Add any overhead requirements to align the data to page or sector boundaries as required by the selected Flash PROM device.
- For possible future migration to a larger FPGA or to allow possible upward migration for additional data, choose a SPI PROM family that offers larger, compatible densities.
- For Spartan-3E FPGA applications that require anti-cloning protection, choose an SPI PROM that provides a unique identifier (ID). See “[Spartan-3E FPGA: Leveraging Security Features in Select Commodity Flash PROMs](#),” [page 299](#). Extended Spartan-3A family FPGAs provide similar protection features using an SPI PROM. See “[Extended Spartan-3A Family FPGA: Imprinting or Watermarking the Configuration PROM with Device DNA](#),” [page 298](#).
- The Xilinx iMPACT software offers direct, in-system programming using Xilinx programming cables, starting with ISE 8.2i. However, the current software version only supports the **STMicro (Numonyx)** and **Atmel** devices indicated in [Table 4-2, page 105](#). Many 25-series PROMs are directly compatible with the STMicro (Numonyx) M25Pxx family and could be substituted in production.

Table 4-5: Other SPI Flash Memory Devices With Data Sheet Compatibility (Unverified by Xilinx, Unsupported in iMPACT)

SPI Flash		Xilinx iMPACT Support	Unique ID	Read Command			Density (bits)								
				Fast Read (0x0B)	Read (0x03)	Read Array (0xE8)									
Vendor	Family			FPGA VS[2:0] Setting			512K	1M	2M	4M	8M	16M	32M	64M	128M
Atmel	AT26			◆	◆				◆	◆	◆	◆			
	AT25			◆	◆		◆	◆	◆						
Spansion (AMD, Fujitsu)	S25FL			◆	◆				◆	◆	◆	◆			
Winbond (NexFlash)	NX25P			◆	◆			◆	◆	◆	◆				
	W25P			◆	◆			◆	◆	◆	◆	◆			
Intel (Numonyx)	S33		◆	◆							◆	◆	◆		
SST	SST25L			◆	◆				◆	◆					
	SST25V			◆	◆		◆	◆	◆	◆	◆				
Macronix	MX25			◆	◆		◆	◆	◆	◆	◆				
Chingis (PMC)	Pm25			◆	◆		◆	◆	◆	◆	◆				
AMIC	A25L			◆	◆				◆	◆	◆				
Eon	EN25			◆	◆		◆	◆	◆	◆	◆				

Notes:

1. Compatibility based on publicly available data sheets.
2. Unique ID indicates that these SPI Flash memory device have factory-programmed unique identifier bits, useful for protecting FPGA applications or IP cores.

SPI Flash PROM Density Requirements

Table 4-6 shows the smallest usable SPI Flash PROM to program a single Extended Spartan-3A family or Spartan-3E FPGA. Commercially available SPI Flash PROMs range in density from 1 Mbit to 128 Mbits. A multiple-FPGA daisy-chained application requires a SPI Flash PROM large enough to contain the sum of the FPGA file sizes. An application can also use a larger-density SPI Flash PROM to hold additional data beyond just FPGA configuration data. For example, the SPI Flash PROM can also store application code for a

[MicroBlaze™](#) RISC processor core integrated in the Spartan-3A or Spartan-3E FPGA. See “SPI Flash Interface after Configuration”.

Table 4-6: Number of Bits to Program an Extended Spartan-3A family or Spartan-3E FPGA and Smallest SPI Flash PROM

Family	FPGA	Number of Configuration Bits (Uncompressed)	Smallest Usable SPI Flash PROM
Spartan-3A/3AN	XC3S50A/AN	437,312	512 Kbit
	XC3S200A/AN	1,196,128	2 Mbit
	XC3S400A/AN	1,886,560	2 Mbit
	XC3S700A/AN	2,732,640	4 Mbit
	XC3S1400A/AN	4,755,296	8 Mbit
Spartan-3A DSP	XC3SD1800A	8,197,280	8 Mbit
	XC3SD3400A	11,718,304	16 Mbit
Spartan-3E	XC3S100E	581,344	1 Mbit
	XC3S250E	1,353,728	2 Mbit
	XC3S500E	2,270,208	4 Mbit
	XC3S1200E	3,841,184	4 Mbit
	XC3S1600E	5,969,696	8 Mbit

FPGA Connections to the SPI PROM

Table 4-7 shows the connections between the SPI Flash PROM and the FPGA’s SPI configuration interface. Each SPI Flash PROM vendor uses slightly different signal naming.

Table 4-8, page 110 provides a complete list of the FPGA pins involved in the Master SPI configuration mode.

Table 4-7: Example SPI Flash PROM Connections and Pin Naming


SPI Flash Pin	FPGA Connection	STMicro (Numonyx)	Winbond/ NexFlash	Silicon Storage Technology	Atmel DataFlash
Slave Data Input	MOSI	D	DI	SI	SI
Slave Data Output	DIN	Q	DO	SO	SO
Slave Select	CSO_B	\bar{S}	\bar{CS}	CE#	\bar{CS}
Slave Clock	CCLK	C	CLK	SCK	SCK
Write Protect 	Not required for FPGA configuration. Must be High to program SPI Flash. Optional connection to FPGA user I/O after configuration.	\bar{W}	\bar{WP}	WP#	\bar{WP}

Table 4-7: Example SPI Flash PROM Connections and Pin Naming (Cont'd)

SPI Flash Pin	FPGA Connection	STMicro (Numonyx)	Winbond/ NexFlash	Silicon Storage Technology	Atmel DataFlash
Hold (see Figure 4-1)	Not required for FPGA configuration but must be High during configuration and programming. Optional connection to FPGA user I/O after configuration. Not applicable to Atmel DataFlash.	$\overline{\text{HOLD}}$	$\overline{\text{HOLD}}$	HOLD#	N/A
Reset (see Figure 4-2)	Only applicable to Atmel DataFlash. Not required for FPGA configuration but must be High during configuration and programming. Optional connection to FPGA user I/O after configuration. Do not connect to FPGA's PROG_B as this potentially prevents direct programming of the DataFlash.	N/A	N/A	N/A	$\overline{\text{RESET}}$
Ready/ $\overline{\text{Busy}}$ (see Figure 4-2)	Only applicable to Atmel DataFlash and only available on certain packages. Not required for FPGA configuration. Output from DataFlash PROM. Optional connection to FPGA user I/O after configuration.	N/A	N/A	N/A	$\text{RDY}/\overline{\text{BUSY}}$

The mode select pins, [M\[2:0\]](#), and the variant select pins, [VS\[2:0\]](#) are sampled when the FPGA's [INIT_B](#) output goes High and must be at defined logic levels during this time. After configuration, when the FPGA's [DONE](#) output goes High, these pins are all available as full-featured user-I/O pins.

(P) Similarly, the FPGA's [HSWAP](#) or [PUDC_B](#) pin must be defined. Set Low to enable pull-up resistors on all user-I/O pins during configuration or High to disable the pull-up resistors. The [HSWAP](#) or [PUDC_B](#) control must remain at a constant logic level throughout FPGA configuration. After configuration, when the FPGA's [DONE](#) output goes High, the [HSWAP](#) or [PUDC_B](#) pin is available as full-featured user-I/O pin and is powered by the [VCCO_0](#) supply.

The FPGA's [DOUT](#) pin is used in daisy-chain applications, described in "[Daisy-Chained Configuration](#)," [page 119](#). In a single-FPGA application, the FPGA's [DOUT](#) pin is in-active, but pulled High via an internal resistor.

(W) The SPI Flash PROM's [Write Protect](#) and [Hold](#) controls are not used by the FPGA during configuration, although the [Hold](#) pin must be High during the configuration process. The PROM's [Write Protect](#) input must be High in order to write or program the Flash memory.

Table 4-8: Serial Peripheral Interface (SPI) Connections

Pin Name	FPGA Direction	Description	During Configuration	After Configuration
HSWAP PUDC_B (P)	Input	User I/O Pull-Up Control. When Low during configuration, enables pull-up resistors in all I/O pins to respective I/O bank V _{CCO} input. See “Pull-Up Resistors During Configuration,” page 62. 0: Pull-ups during configuration 1: No pull-ups	Drive at valid logic level throughout configuration.	User I/O
M[2:0]	Input	Mode Select. Selects the FPGA configuration mode. Extended Spartan-3A family FPGAs have dedicated internal pull-up resistors on these pins. See “Choose a Configuration Mode: M[2:0],” page 50.	M2 = 0, M1 = 0, M0 = 1. Sampled when INIT_B goes High. Extended Spartan-3A family FPGAs have internal pull-up resistors to V _{CCO_2} .	User I/O
VS[2:0]	Input	Variant Select. Instructs the FPGA how to communicate with the attached SPI Flash PROM. Extended Spartan-3A family FPGAs have dedicated internal pull-up resistors on these pins.	Must be at the logic levels shown in Table 4-2 . Sampled when INIT_B goes High. Extended Spartan-3A family FPGAs have internal pull-up resistors to V _{CCO_2} .	User I/O
MOSI	Output	Master SPI Serial Data Output. Connect to the SPI Flash PROM’s Slave Data Input pin.	FPGA sends SPI Flash memory read commands and starting address to the PROM’s serial data input.	User I/O
DIN	Input	Master SPI Serial Data Input. Connect to the SPI Flash PROM’s Slave Data Output pin.	FPGA receives serial data from PROM’s serial data output.	User I/O
CSO_B	Output	Master SPI Chip Select Output. Active Low. Connect to the SPI Flash PROM’s Slave Select input.	If HSWAP or PUDC_B = 1, connect this signal to a 4.7 k Ω pull-up resistor to 3.3V.	Drive CSO_B High after configuration to disable the SPI Flash and reclaim the MOSI, DIN, and CCLK pins. Optionally, re-use this pin and MOSI, DIN, and CCLK to continue communicating with SPI Flash. See “SPI Flash Interface after Configuration,” page 116.

Table 4-8: Serial Peripheral Interface (SPI) Connections (Cont'd)

Pin Name	FPGA Direction	Description	During Configuration	After Configuration
CCLK	Output	Configuration Clock. Generated by FPGA internal oscillator. Connect to the SPI Flash PROM's Slave Clock input. Frequency controlled by ConfigRate bitstream generator option. If CCLK PCB trace is long or has multiple connections, terminate this output to maintain signal integrity. See " Configuration Clock: CCLK ," page 56.	Drives PROM's clock input.	User I/O. Drive High or Low if not used. Avoid excessive loading on CCLK to maintain best signal integrity for configuration.
DOUT	Output	Serial Data Output. Used in multi-FPGA daisy-chain configurations.	Not used in single-FPGA designs; DOUT is pulled up, not actively driving. In a daisy-chain configuration, this pin connects to DIN input of the next FPGA in the chain.	User I/O
INIT_B	Open-drain bidirectional I/O	Initialization Indicator. Active Low. Goes Low at start of configuration during Initialization memory clearing process. Released at end of memory clearing, when mode select pins are sampled. See " Initializing Configuration Memory, Configuration Error: INIT_B ," page 61.	Active during configuration. If SPI Flash PROM requires more than 2 ms to awake after powering on, hold INIT_B Low until PROM is ready. See " Power-On Precautions if System 3.3V Supply is Last in Sequence ," page 112. If CRC error detected during configuration, FPGA drives INIT_B Low. See " CRC Checking during Configuration ," page 309.	User I/O. If unused in the application, drive INIT_B High to avoid a floating value. See INIT_B " After Configuration ".
DONE	Open-drain bidirectional I/O	FPGA Configuration Done. Low during configuration. Goes High when FPGA successfully completes configuration. See " DONE Pin ," page 52.	Low indicates that the FPGA is not yet configured.	Pulled High via external pull-up. When High, indicates that the FPGA successfully configured.
PROG_B	Input	Program FPGA. Active Low. When asserted Low for 500 ns or longer, forces the FPGA to restart its configuration process by clearing configuration memory and resetting the DONE and INIT_B pins once PROG_B returns High.	Must be High to allow configuration to start.	Drive PROG_B Low and release to reprogram FPGA. Hold PROG_B to force FPGA I/O pins into Hi-Z, allowing direct programming access to SPI Flash PROM pins.
VCCO_2	Voltage supply input	Voltage Supply Input to I/O Bank 2. Supplies interface pins to SPI Flash PROM.	3.3V. Ensure that either the VCCO_2 supply ramps faster than V _{CCINT} or V _{CCAUX} or that the PROM wakes-up sufficiently fast. See " Power-On Precautions if System 3.3V Supply is Last in Sequence ," page 112.	3.3V

Voltage Compatibility

Available SPI Flash PROMs use a single 3.3V supply voltage. All of the FPGA's SPI Flash interface signals are within I/O Bank 2. Consequently, the FPGA's V_{CCO_2} supply voltage must also be 3.3V to match the SPI Flash PROM.

Also, see “Power-On Precautions if System 3.3V Supply is Last in Sequence,” page 112.

See also “JTAG Cable Voltage Compatibility,” page 198.

Power-On Precautions if System 3.3V Supply is Last in Sequence

Spartan-3E and Extended Spartan-3A family FPGAs have a built-in power-on reset (POR) circuit. The FPGA waits for its three power supplies — V_{CCINT} , V_{CCAUX} , and V_{CCO} to I/O Bank 2 (V_{CCO_2}) — to reach their respective power-on thresholds before beginning the configuration process. See “Power-On Reset (POR),” page 240 for more information.

The SPI Flash PROM is powered by the same voltage supply feeding the FPGA's V_{CCO_2} voltage input, typically 3.3V. SPI Flash PROMs specify that they cannot be accessed until their V_{CC} supply reaches its minimum data sheet voltage, followed by an additional delay. For some devices, this additional delay is as little as 10 μ s as shown in Table 4-9. For other vendors, this delay is as much as 20 ms.

Table 4-9: Example Minimum Power-On to Select Times for Various SPI Flash PROMs

Vendor	SPI Flash PROM Part Number	Data Sheet Minimum Time from V_{CC} min to Select = Low		
		Symbol	Value	Units
STMicro (Numonyx)	M25Pxx	T_{VSL}	10	μ s
Spansion	S25FLxxxA	t_{PU}	10	ms
NexFlash	NX25xx	T_{VSL}	10	μ s
Macronix	MX25Lxxxx	t_{VSL}	10	μ s
Silicon Storage Technology	SST25LFxx	$T_{PU-READ}$	10	μ s
Programmable Microelectronics Corporation	Pm25LVxxx	T_{VCS}	50	μ s
Atmel Corporation	AT45DBxxxD	t_{VCSL}	50	μ s
	AT45DBxxxB	—	20	ms

Notes:

1. Memory vendors are continuously improving their products and specifications. Please check with the memory vendor's data sheets for up-to-date values.

In many systems, the 3.3V supply feeding the FPGA's V_{CCO_2} input is valid before the FPGA's other V_{CCINT} and V_{CCAUX} supplies, and consequently, there is no issue. However, if the 3.3V supply feeding the FPGA's V_{CCO_2} supply is last in the sequence, a potential race occurs between the FPGA and the SPI Flash PROM, as shown in Figure 4-3.

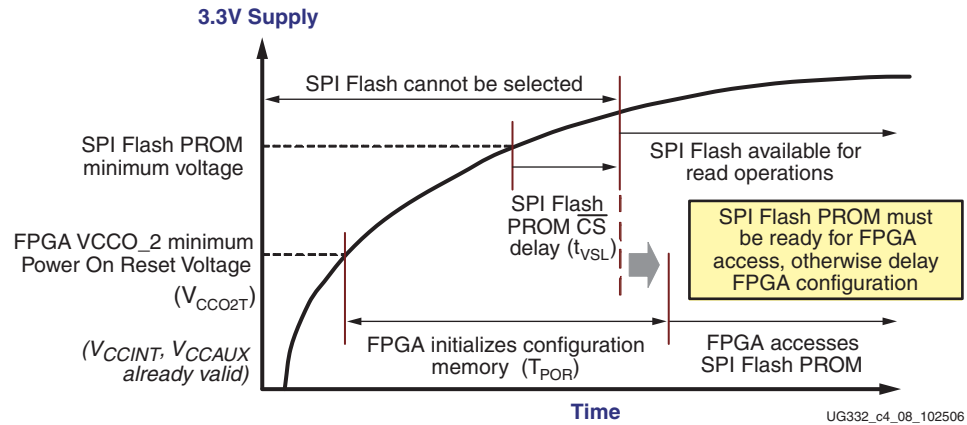


Figure 4-3: SPI Flash PROM/FPGA Power-On Timing if 3.3V Supply is Last in Power-On Sequence

If the FPGA's V_{CCINT} and V_{CCAUX} supplies are already powered and valid, then the FPGA waits for V_{CCO_2} to reach its minimum threshold voltage before starting configuration. This threshold voltage is labeled as V_{CCO2T} in the Spartan-3E or Extended Spartan-3A family data sheet. The range of values is listed in [Table 4-10](#) and are substantially lower than the SPI Flash PROM's minimum voltage. Once all three FPGA supplies reach their respective Power-On Reset (POR) thresholds, the FPGA starts the configuration process and begins initializing its internal configuration memory. After initialization completes, the FPGA deasserts `INIT_B`, selects the SPI Flash PROM, and starts sending the appropriate read command. The SPI Flash PROM must be ready for read operations at this time. The FPGA typically delays configuration long enough for the configuration source to be ready. If the configuration source is not ready when the FPGA begins configuration, the Configuration Watchdog Timer will allow the FPGA to automatically re-attempt configuration.

There are a few potential solutions if the 3.3V supply is last in the sequence and does not ramp fast enough, or if the SPI Flash PROM cannot be ready when required by the FPGA.

- Change the power sequence order so that the 3.3V V_{CCO_2} is powered and valid before the FPGA's V_{CCINT} or V_{CCAUX} supply.
- Choose a different SPI Flash PROM family or vendor, one with a faster power-on timing specification. For example, while the [Atmel AT45DBxxxB](#) family has 20 ms power-on requirement, the compatible [AT45DBxxxD](#) family requires just 30 μ s.
- Delay the FPGA configuration process by holding either the FPGA's `PROG_B` input or `INIT_B` input Low. Release the FPGA when the SPI Flash PROM is ready. For example, a simple R-C delay circuit attached to the `INIT_B` pin forces the FPGA to wait for a preselected amount of time. Alternately, a Power Good signal from the 3.3V supply or a system reset signal accomplishes the same purpose. Use an open-drain or open-collector output when driving `PROG_B` or `INIT_B`.

Table 4-10: Spartan-3E and Extended Spartan-3A Family DSP Power-On Reset Timing and Thresholds

Symbol	Description	Spartan-3E	Spartan-3A/3AN Spartan-3A DSP	Units
V_{CCO2T}	VCCO_2 voltage at which Power-On Reset (POR) circuit is released, assuming V_{CCINT} and V_{CCAUX} supplies are already applied and valid.	0.4 to 1.0	0.8 to 2.0	V
T_{POR}	The time from when the FPGA's Power-On Reset (POR) circuit is released to the rising transition of the INIT_B pin	Up to 7	Up to 18	ms

Extended Spartan-3A Family and Configuration Watchdog Timer

Extended Spartan-3A family FPGAs include a configuration watchdog timer (CWDT) which makes SPI Flash configuration more robust, even when the 3.3V supply is applied last.

In Master SPI mode, the CWDT ensures that the FPGA reads a valid synchronization word from the SPI Flash PROM within the first $2^{16}-1$ cycles of **CCLK**. The synchronization word is part of the FPGA configuration bitstream. If the FPGA does not find the synchronization word, the CWDT forces the FPGA to automatically resend the SPI Flash read command and to retry the configuration process. The CWDT retries to successfully configure from SPI Flash three times before failing. If the FPGA fails to configure, it then drives the **INIT_B** pin Low, indicating a failure.

CCLK Frequency

In SPI Flash mode, the FPGA's internal oscillator generates the configuration clock frequency. The FPGA provides this clock on its **CCLK** output pin, driving the PROM's **Slave Clock** input pin. The FPGA begins configuring using its lowest frequency setting. If so specified in the configuration bitstream, the FPGA increases the **CCLK** frequency to the specified setting for the remainder of the configuration process. The maximum frequency is specified using the **ConfigRate** bitstream generator option. The maximum frequency supported by the FPGA configuration logic depends on the timing for the SPI Flash device. Without examining the timing for a specific SPI Flash PROM, use **ConfigRate** = 12 or lower. SPI Flash PROMs that support the FAST READ command support higher data rates. Some such PROMs support up to **ConfigRate** = 25 and beyond but require careful data sheet analysis. See "Serial Peripheral Interface (SPI) Configuration Timing," page 138 for more detailed timing analysis.

Table 4-11 lists the various **ConfigRate** setting options and the corresponding clock-to-output requirement, T_V , for the SPI Flash PROM. The T_V value is determined according to the equation in Table 4-16, page 141. Extended Spartan-3A family FPGAs have more **ConfigRate** settings than Spartan-3E FPGAs, hence the shaded cells under the Spartan-3E column. Unless a **ConfigRate** setting is specified when generating the bitstream, the Spartan-3E FPGA uses the default, slowest setting of **ConfigRate** = 1, which lengthens the overall configuration time. The Extended Spartan-3A family FPGAs use a default setting of **ConfigRate** = 6.

Table 4-11: FPGA *ConfigRate* Setting and Corresponding SPI Flash PROM Clock-to-Output Requirements (T_V)

<i>ConfigRate</i> Bitstream Setting	SPI Flash Maximum T_V Specification				Units
	Spartan-3E		Spartan-3A/3AN Spartan-3A DSP		
	Commercial	Industrial	Commercial	Industrial	
1 (Spartan-3E default)	≤ 265	≤ 224	≤ 588	≤ 553	ns
3	≤ 127	≤ 106	≤ 189	≤ 178	
6 (Extended Spartan-3A family default)	≤ 58	≤ 47	≤ 91.3	≤ 85.6	
12	≤ 23.5	≤ 18.3	≤ 41.9	≤ 39	
13			≤ 37.1	≤ 34.8	
17			≤ 27.2	≤ 25.3	
22			≤ 18.6	≤ 17.2	
25	≤ 6.1	≤ 3.5	≤ 15.3	≤ 14.4	
27			≤ 13.9	≤ 13	
33			≤ 10.1	≤ 9.2	
44			≤ 5.3	≤ 4.9	

SPI Flash Interface after Configuration

After the FPGA successfully completes configuration, all of the pins connected to the SPI Flash PROM are available as user-I/O pins.

If Not Using SPI Flash after Configuration

If not using the SPI Flash PROM after configuration, drive **CSO_B** High to disable the PROM, as shown in [Figure 4-4](#). The **MOSI**, **DIN**, and **CCLK** pins are then available as general-purpose I/O pins in the FPGA application, although avoid additional loading on **CCLK** if possible to maintain best signal integrity.

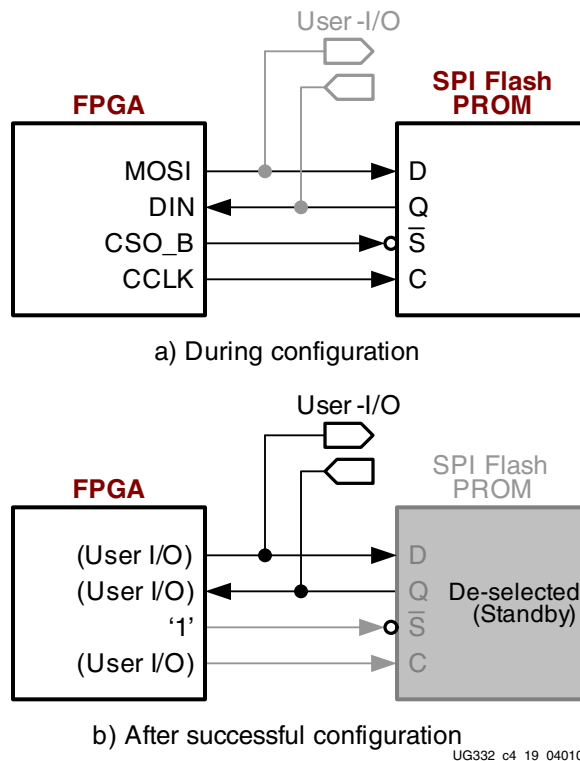


Figure 4-4: If Not Using SPI after Configuration, Drive CSO_B Pin High

De-selecting **CSO_B** also places the SPI PROM in the lower-power Standby mode. See “[Deassert CSO_B to Enter Standby Mode,](#)” page 142.

If Using SPI Flash Interface after Configuration

Because all the interface pins are user I/O after configuration, the FPGA application can continue to use the SPI Flash interface pins to communicate with the SPI Flash PROM, as shown in Figure 4-5. SPI Flash PROMs offer random-accessible, byte-addressable, read/write, nonvolatile storage to the FPGA application.

Caution! Allow the FPGA configuration logic to use the CCLK pin to complete configuration and startup before using it to control the SPI Flash interface. Although most dual-purpose pins become I/O at the GTS cycle, CCLK must wait until the End of Startup (EOS). Delay access by a couple clock cycles after configuration to avoid conflicts. See “Startup” in Chapter 12.

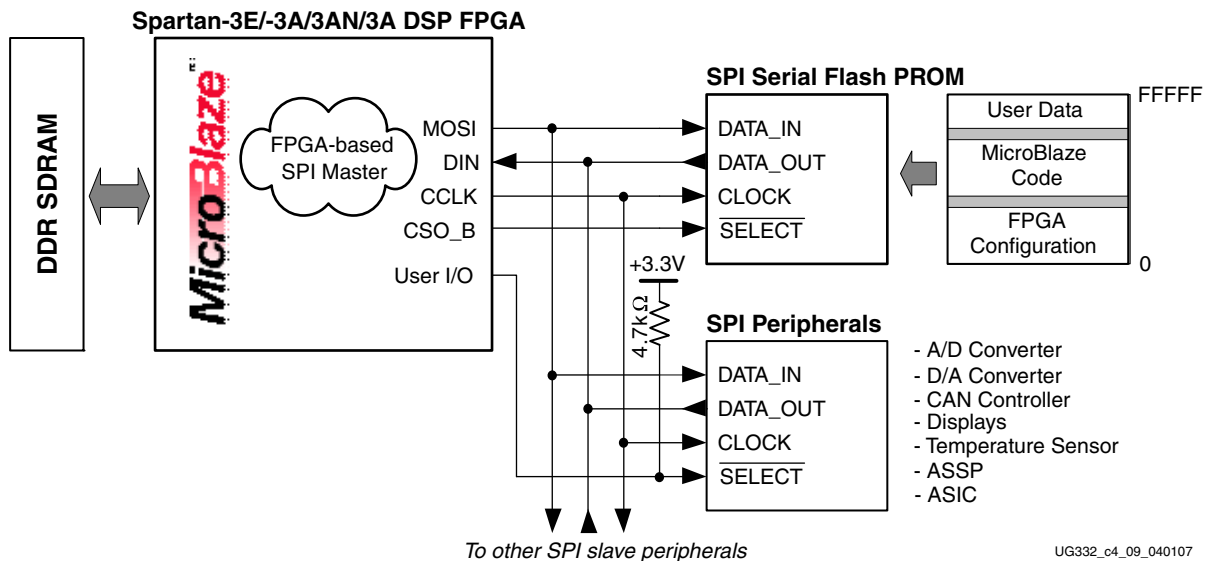


Figure 4-5: Using the SPI Flash Interface After Configuration

SPI Master Interface using FPGA Logic

The FPGA does not contain a dedicated SPI interface, except for configuration. Consequently, to access the SPI Flash or other SPI devices after configuration, the FPGA application must contain an SPI bus master interface. Xilinx provides SPI interface cores, as described below.

- For an application that already includes a [MicroBlaze processor core](#), the Xilinx [Embedded Development Kit \(EDK\)](#) includes an SPI interface that connects to the MicroBlaze OPB bus. Depending on the options used, the SPI interface core uses between 147 to 203 slices.
 - ◆ **OPB Serial Peripheral Interface Product Specification**
http://www.xilinx.com/support/documentation/ip_documentation/opb_spi.pdf
- For general applications, the 8-bit [PicoBlaze™ processor core](#) offers an easy-to-use solution that requires approximately 100 slices and a block RAM. Example design solutions are available for the [Spartan-3E FPGA Starter Kit](#) board.
 - ◆ **PicoBlaze STMicro SPI Flash Programmer**
www.xilinx.com/products/boards/s3estarter/reference_designs.htm#picoblaze_spi_flash_programmer
 - ◆ **PicoBlaze SPI-based D/A Converter Controller**
www.xilinx.com/products/boards/s3estarter/reference_designs.htm#picoblaze_dac_control

Accessing SPI Flash PROM

SPI Flash PROMs are available in densities ranging from 1 Mbit up to 128 Mbits. However, a single Spartan-3A/3E FPGA requires less than 6 Mbits. A Spartan-3A DSP FPGA requires a little more than 11 Mbits. If desired, use a larger SPI Flash PROM to contain additional nonvolatile application data, such as MicroBlaze processor code, or other user data such as serial numbers and Ethernet MAC IDs. In the example shown in [Figure 4-5](#), the FPGA configures from SPI Flash PROM. Then using FPGA logic after configuration, the FPGA copies MicroBlaze code from SPI Flash into external DDR SDRAM for code execution. Similarly, the FPGA application can store nonvolatile application data within the SPI Flash PROM.

The FPGA configuration image, or initial configuration image for an Extended Spartan-3A family MultiBoot application, is always stored at starting address 0. Store any additional data beginning in the next available SPI Flash PROM sector or page. Do not mix configuration data and user data in the same sector or page.

After configuration, the FPGA application can exploit any special features of the attached SPI serial Flash PROM. For example, the Atmel AT45DB-series PROMs support a slightly-modified serial interface called Rapid-S. The FPGA cannot configure using this mode but after configuration, the FPGA application can use Rapid-S to increase overall data throughput. Similarly, the NexFlash/Winbond W25X-series PROMs support a feature called Dual-Output SPI that transmits two data bits per clock cycle but requires a special read command. The FPGA does not support this command for configuration, but the FPGA application can issue the command after configuration.

Accessing other SPI-compatible Peripherals

Similarly, the SPI bus can be expanded to additional SPI peripherals. Because SPI is a common industry-standard interface, various SPI-based peripherals are available, such as analog-to-digital (A/D) converters, digital-to-analog (D/A) converters, CAN controllers, and temperature sensors.

The [MOSI](#), [DIN](#), and [CCLK](#) pins are common to all SPI peripherals. Connect the select input on each additional SPI peripheral to one of the FPGA user I/O pins. If [HSWAP](#) or [PUDC_B](#) = 0 during configuration, the FPGA holds the select line High. If [HSWAP](#) or [PUDC_B](#) = 1, connect the select line to +3.3V via an external 4.7 k Ω pull-up resistor to avoid spurious read or write operations. After configuration, drive the select line Low to select the desired SPI peripheral.

During the configuration process, [CCLK](#) is controlled by the FPGA and limited to the frequencies generated by the FPGA. After configuration, the FPGA application can use other clock signals to drive the [CCLK](#) pin and can further optimize SPI-based communication.

Caution! Avoid excessive loading on the [CCLK](#) pin. Excessive loading will degrade the signal integrity on this crucial signal. Use the recommended design practices described in “[CCLK Design Considerations](#),” page 58.

Refer to the individual SPI peripheral data sheet for specific interface and communication protocol requirements.

Caution! Although many devices claim to have an SPI interface, the timing and even signal polarity vary between devices and between vendors. Check the data sheet for the specific device to determine compatibility.

Daisy-Chained Configuration

If the application requires multiple FPGAs with different configurations, then configure the FPGAs using a daisy chain, as shown in [Figure 4-6, page 119](#). Use SPI Flash mode ($M[2:0] = <0:0:1>$) for the FPGA connected to the SPI PROM and Slave Serial mode ($M[2:0] = <1:1:1>$) for all other FPGAs in the daisy chain. After the master FPGA—the FPGA on the left in the diagram—finishes loading its configuration data from the SPI Flash PROM, the master FPGA supplies data to the next FPGA in the daisy chain via the **DOUT** output pin, clocked on the falling **CCLK** edge.

Also, to successfully configure a daisy chain, the *GTS_cycle* bitstream option must be set to a Startup phase after the *DONE_cycle* setting for all FPGAs in the chain. This is the software default setting. Optionally, set *GTS_cycle:Done*.

The 0-ohm resistors at the output of each FPGA's **INIT_B** and **DONE** pin is recommended for debugging purposes. Should there be a configuration error, the FPGAs can be individually isolated. The jumper on the master FPGA's **DONE** pin is recommended for future in-system programming support as well as for debugging purposes.

The pull-up resistors shown in gray are optional, but should be provided in the board design. The resistors themselves do not need to be stuffed during board manufacturing. As described in [Table 2-13, page 65](#), the dedicated pull-up resistors on Spartan-3 generation FPGAs are sufficiently strong to pull-up the corresponding signal pin. The Thevenin termination resistors on **CCLK** are also optional, but also recommended in the board design.

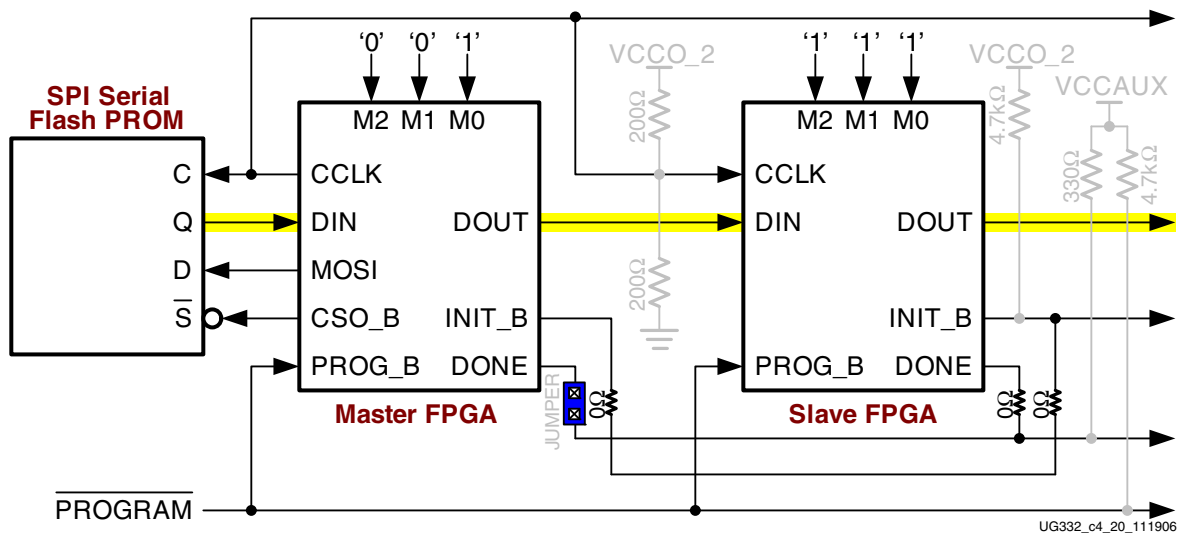


Figure 4-6: Daisy Chaining from SPI Flash Mode

Caution! SPI mode daisy chains are supported for Spartan-3E FPGAs only in Stepping 1 silicon versions. SPI mode daisy chains are supported on all Spartan-3E Automotive grade devices, which are all based on Stepping 1 silicon, and all Extended Spartan-3A family FPGA versions.

Ganged or Broadside Configuration

“Daisy-Chained Configuration” is designed to load multiple FPGAs, each with a different design and typically of different array size. However, some applications include multiple, identical FPGAs, all programmed with the same bitstream. Instead of daisy chaining the FPGAs and storing multiple copies of the same bitstream, “Ganged or Broadside Configuration” supports programming multiple, identical FPGAs with the same bitstream.

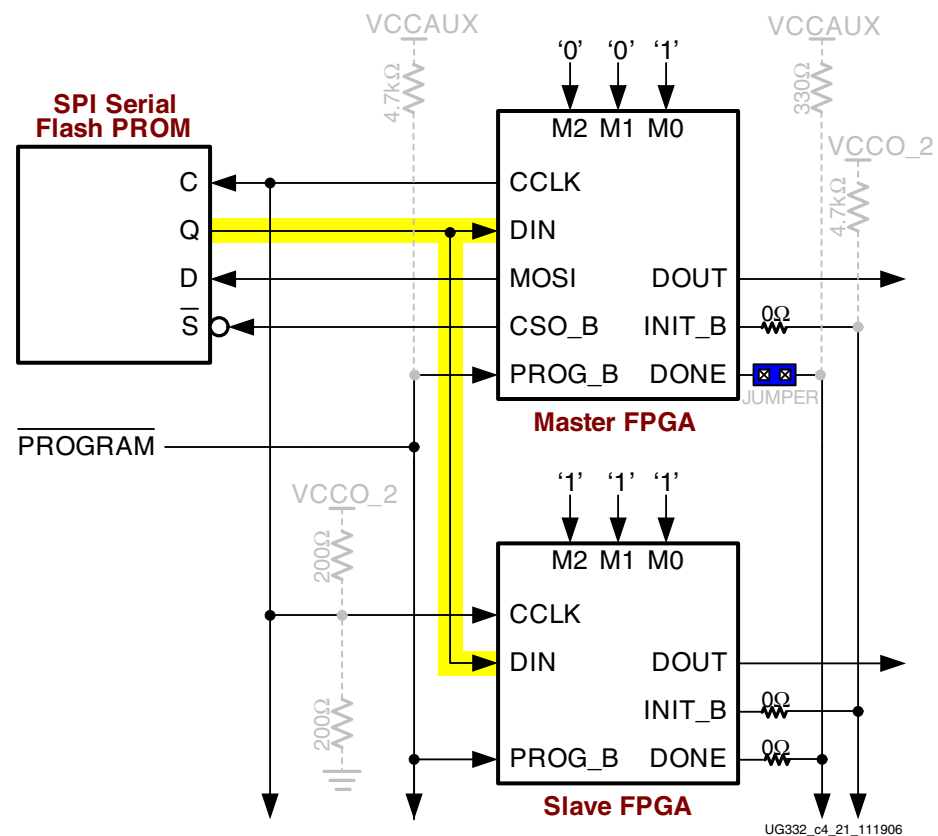


Figure 4-7: Multiple, Identical FPGAs Programmed with the Same Bitstream

Programming Support

In production applications, the SPI Flash PROM is usually preprogrammed before it is mounted on the printed circuit board. The [Xilinx ISE development software](#) produces industry-standard programming files that can be used with third-party gang programmers. Consult your specific SPI Flash vendor for recommended production programming solutions.

There are multiple programming methods for the attached SPI memory as described below.

Starting with ISE 9.1i, Service Pack 2 and later, the iMPACT programming software supports two different methods to program an attached SPI Flash PROM, as summarized in [Table 4-12](#).

Using the Direct Programming Method, the programming cable communicates directly to the SPI Flash PROM. The FPGA is not involved in the programming process and the FPGA

I/O pins that connect to the PROM must be in their high-impedance state (Hi-Z) during programming. Hold the FPGA's PROG_B input Low to place the I/Os in Hi-Z; the FPGA's DONE pin remains Low.

Using the Indirect Programming Method, the programming cable connects to the FPGA's JTAG port. The iMPACT software first programs the FPGA with a special design that performs the actual SPI PROM programming and uses the JTAG interface as a serial communications port. During the process, the FPGA's DONE output is High because the FPGA is configured with the programming application. All pins that are not connected to the SPI Flash PROM or the JTAG interface have an internal pull-up resistor to the VCCO voltage supply associated with the pin.

For the Spartan-3AN family, iMPACT supports programming of the internal Flash and does not support indirect programming of external Flash.

Table 4-12: Summary of SPI Flash PROM Programming Options

	Direct Method	Indirect Method
Detailed Instructions	"Direct SPI Programming using iMPACT," page 131	"Indirect SPI Programming using iMPACT," page 134
ISE Version Required	ISE 9.1i or later	ISE 9.1i, Service Pack 2 or later for Spartan-3A/3A DSP FPGAs; ISE 10.1 for Spartan-3E FPGAs
Interface/Cable Connection	Directly to SPI PROM	FPGA's JTAG Port
DONE Pin Status during Programming	Low	High (FPGA is configured with special programming design)
Required PROG_B Control	PROG_B = Low	N/A
Status of non-SPI Pins during Programming	High-impedance because PROG_B = Low	Pulled High using internal pull-up resistor to associated VCCO supply input

Third-Party Programmer (Off-board Programming)

Off-board programming, before board assembly, using a third party programmer is likely the preferred method for high-volume production. Most [Xilinx distributors](#) offer programming services or can arrange for such services. Check the PROM vendor's web site for a list of approved and qualified third-party device programmers. See ["Preparing an SPI PROM File," page 126](#) to properly format the programming file.

Direct, SPI In-System Programming

For systems requiring in-system programming support, there are different options for production and prototyping phases. For production programming, some third party PROM programmers utilize a socket adapter with attached wires to program the SPI flash memory in-system. For prototype programming, the Xilinx iMPACT software provides direct, in-system programming support for limited set of [STMicro \(Numonyx\)](#) and [Atmel](#) SPI Flash memories.

Requirements for iMPACT Direct Programming Support

The following are required to successfully perform in-system programming on the attached SPI serial Flash PROM.

- A Xilinx programming cable
 - ◆ **Platform Cable USB II**
<http://www.xilinx.com/products/devkits/HW-USB-II-G.htm>
 - ◆ **Parallel Cable IV**
<http://www.xilinx.com/products/devkits/HW-PC4.htm>
- A compatible cable connector on-board
- Properly installed Xilinx ISE 8.2i software (or later)

Programmable Cable Connections

All modern Xilinx programming cables use a standard 14-pin ribbon cable and associated socket. The socket connections appear in [Figure 4-1, page 102](#) and [Figure 4-2, page 103](#), along with a detail pinout table in [Table 4-13](#). The mechanical dimensions are provided in [Figure 9-6, page 208](#) and vendor part numbers provided in [Table 9-7, page 208](#).

As shown in [Table 4-13](#), one side of the socket connects entirely to GND for better signal integrity. The other side of the cable includes the V_{REF} voltage connection and the four SPI Flash control signals. When used for SPI programming, the programming cable behaves as an SPI Master, controlling all transactions on the SPI bus.

Table 4-13: Xilinx Download Header Signal Description for In-System SPI Flash PROM Programming

Signal	Socket Pin (top view)		Direction	Signal Connections to SPI PROM, System	“Flying Lead” Label/ Wire Color ⁽¹⁾
GND	1	2	←	V_{REF} : Connect to 3.3V (<i>VCCO_2</i>), which is common to the FPGA and SPI PROM. The voltage reference must be regulated and must not have a current limiting series resistor.	VREF (red)
GND	3	4	→	SPI Slave Select : Connect to the SPI PROM’s <i>Slave Select</i> input.	TMS/PROG (green)
GND	5	6	→	SPI Clock : Connect to the SPI PROM’s <i>Slave Clock</i> input.	TCK/CCLK (yellow)
GND	7	8	←	SPI Master Input/Slave Output : Connect to the SPI PROM’s <i>Slave Data Output</i> .	TDO/DONE (magenta)
GND	9	10	→	SPI Master Output /Slave Input : Connect to the SPI PROM’s <i>Slave Data Input</i> .	TDI/DIN (white)
GND	11	12	–	Reserved . Do not connect.	–
GND	13	14	↔	D.N.C. Do not connect. Although the cable leads label this as <i>INIT</i> , do not connect it to the FPGA’s <i>INIT_B</i> pin.	–/INIT (gray)

Notes:

1. The “Flying Lead” adapter is only required if using stake pins instead of the recommended 14-pin socket.

The specified surface-mount cable connector requires only 0.162 square inches of board space. The Xilinx iMPACT programming solution is only qualified for system prototyping so the socket can be removed from the production bill of materials to save cost.

Alternatively, the Xilinx programming cables optionally support “flying leads” that push on to standard 0.1-inch stake pins. However the ribbon cable and associated socket have superior signal integrity and provide fast programming speeds. Also ensure that the programming cable leads are connected correctly. The SPI programming capability is new for the Xilinx programming cables and existing cables may have different signal labels, as indicated in [Table 4-13](#).

Forcing FPGA SPI Bus Pins to High-impedance During Programming

Because the programming cable acts as an SPI bus Master, the FPGA’s SPI pins must be floating, or high-impedance (Hi-Z). This requirement also applies for third party programmers that directly program the SPI Flash PROM. Ensure that the FPGA *MOSI*, *DIN*, *CSO_B*, and *CCLK* pins are all high impedance (floating, Hi-Z), allowing the programmer to have full and direct control over the SPI PROM. There are three different methods to place the FPGA SPI signals in high-impedance, listed below.

- Option 1** Hold the FPGA's **PROG_B** pin Low throughout the programming process. The FPGA is unconfigured during the programming process and automatically loads the new SPI Flash PROM image when **PROG_B** is released High.
- Option 2** Change the FPGA's mode pins to JTAG mode ($M[2:0] = <1:0:1>$) and pulse the FPGA's **PROG_B** pin. Do not perform any JTAG operations. All FPGA I/O pins are forced to their high-impedance state. The FPGA is unconfigured during the programming process. The FPGA's **M[2:0]** pins must be returned to the SPI Flash setting and **PROG_B** pin must be pulsed Low before the FPGA reloads the new SPI Flash PROM image.
- Option 3** Within a functioning FPGA application, use an internal control signal that three-states the **MOSI**, **DIN**, **CCLK**, and **CSO_B** pins. The FPGA remains configured with the current configuration. Pulse the **PROG_B** pin Low or, on Extended Spartan-3A family FPGAs, issue a MultiBoot reconfiguration operation with a start address of zero.

If using **Option 1** or **Option 2**, be aware that pull-up resistors to **VCCO_2** are enabled on the FPGA's SPI pins if the FPGA's **HSWAP** or **PUDC_B** pin is Low. Using **Option 3**, the FPGA's SPI pins are fully controlled by the FPGA application.

Direct, In-system SPI Programming Using FPGA as Intermediary

This method is typically used to update the SPI serial Flash, using the FPGA as the actual programmer. The advantage is that the FPGA's flexibility allows the FPGA to connect to practically any digital interface to receive the programming data. The FPGA-based "programmer" can be included as part of the application or, alternatively, downloaded temporarily into the FPGA using the FPGA's JTAG interface.

The [Spartan-3E FPGA Starter Kit](#) includes a design example that programs the attached STMicro (Numonyx) M25P16 SPI Flash using an RS-232 connection to a PC or workstation.

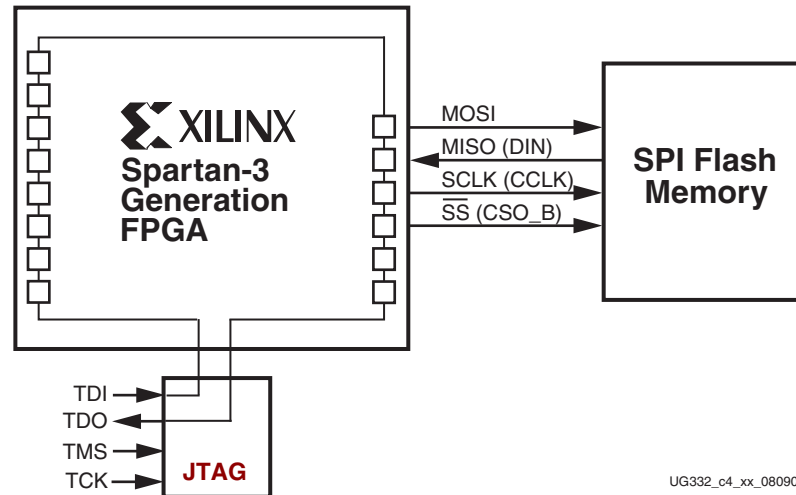
- **PicoBlaze RS-232 to STMicro SPI Flash Programmer**
www.xilinx.com/products/boards/s3estarter/reference_designs.htm#picoblaze_spi_flash_programmer

The [Spartan-3A FPGA Starter Kit](#) includes a design example that programs the attached Atmel AT45DB161D DataFlash PROM using an RS-232 connection to a PC or workstation.

- **PicoBlaze RS-232 to Atmel DataFlash Programmer**
www.xilinx.com/products/boards/s3astarter/reference_designs.htm#atmel_spi_flash_programmer

Indirect, In-System SPI Programming Using FPGA JTAG Chain

The FPGA has JTAG test capabilities which include the standard PRELOAD and EXTEST commands. When using these commands, it is possible to drive and sample the pins of the FPGA with the JTAG chain and thereby stimulate the pins of the SPI memory via the associated FPGA pins and the traces routed on the PCB. This method, shown in [Figure 4-8](#), is supported by many third-party JTAG tool vendors. However, this method is often much slower than the "Direct, SPI In-System Programming" technique.



UG332_c4_xx_080906

Figure 4-8: Using FPGA's JTAG Test Chain to Program Attached SPI Flash

The advantage to this approach is that it requires minimal wiring for in-system programming and that the SPI Flash PROM can be programmed during other JTAG-based board test operations.

For easier development, Xilinx recommends including the JTAG programming cable socket shown in [Figure 4-1, page 102](#) and [Figure 4-2, page 103](#). The FPGA configuration can be downloaded directly into the FPGA for development purposes without requiring that the SPI Flash PROM be programmed.

For more information on the JTAG interface, see [Chapter 9, "JTAG Configuration Mode and Boundary-Scan,"](#) especially "Programming Cables and Headers," page 207.

Generating the Bitstream for a Master SPI Configuration

To create the FPGA bitstream for a Master SPI configuration, follow the steps outlined in "Setting Bitstream Options, Generating an FPGA Bitstream," page 42. For an FPGA configured in Master SPI mode, set the following bitstream generator options.

ConfigRate: CCLK Frequency

Set the *ConfigRate* option as described in "CCLK Frequency," page 114. Using ISE Project Navigator, the Configuration Rate frequency is set in Step 7 in [Figure 1-7, page 44](#).

```
-g ConfigRate:12
```

StartupClk: CCLK

By default, the configuration Startup clock source is the internally generated CCLK. Keep the *StartupClk* bitstream generation option, shown as Step 13 in [Figure 1-8, page 45](#).

```
-g StartupClk:Cclk
```

DriveDone: Actively Drive DONE Pin

In a single FPGA design or for the Master FPGA in a multi-FPGA daisy chain, set the FPGA to actively drive the DONE pin after successfully completing the configuration process.

Using ISE Project Navigator, check the **Drive Done Pin High** option, shown as Step 16 in [Figure 1-8, page 45](#).

```
-g DriveDone:Yes
```

DONE_cycle: Daisy Chains with Spartan-3E Master

If a Spartan-3E FPGA is the Master FPGA in an SPI-based daisy chain, ensure that **DONE_cycle** is set for cycle 5 or earlier. From ISE Project Navigator, the **DONE_cycle** setting is the **Done (Output Events)** option, shown as Step 14 in [Figure 1-8, page 45](#).

```
-g DONE_cycle:4
```

GTS_cycle: Global Three-State Release Timing for Daisy Chains

If creating a multi-FPGA daisy chain, set the **GTS_cycle** option to be later than the **DONE_cycle** setting, which is the default setting for both. Alternatively, set **GTS_cycle:Done**. From ISE Project Navigator, the **GTS_cycle** setting is the **Enable Outputs (Output Events)** option, shown as Step 14 in [Figure 1-8, page 45](#).

Preparing an SPI PROM File

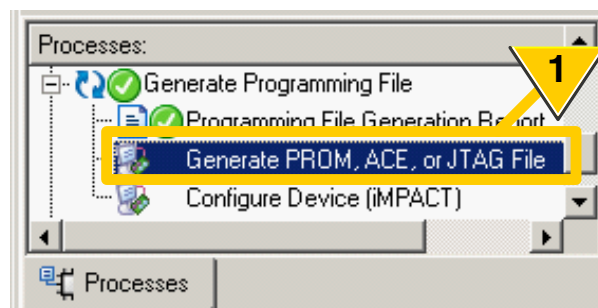
This section provides guidelines to create PROM files for SPI Flash memories.

The Xilinx software tools, “iMPACT” or “PROMGen”, generate SPI-formatted PROM files from the FPGA bitstream or bitstreams. SPI Flash memory devices serially output data bytes with the most-significant bit (MSB) first while Xilinx PROMs output data least-significant bit (lsb) first. Consequently, a PROM file formatted for an SPI Flash memory device is bit-reversed within each byte, directly opposite from the bit ordering for a standard Xilinx PROM file. When using PROMGen, the **-spi** option is **required** for proper formatting.

iMPACT

The following steps graphically describe how to create an SPI-formatted PROM file using iMPACT from within the ISE Project Navigator. To create a Spartan-3A/3A DSP MultiBoot image for an SPI Flash memory, see “[Generating an Extended Spartan-3A Family MultiBoot PROM Image using iMPACT](#),” page 278.

1. From within the ISE Project Navigator, double-click **Generate PROM, ACE, or JTAG File** from within the Process pane, as shown in [Figure 4-9](#).



UG332_c4_10_110206

Figure 4-9: Double-click **Generate PROM, ACE or JTAG File**

- As shown in [Figure 4-10](#), select **Prepare a PROM File**.

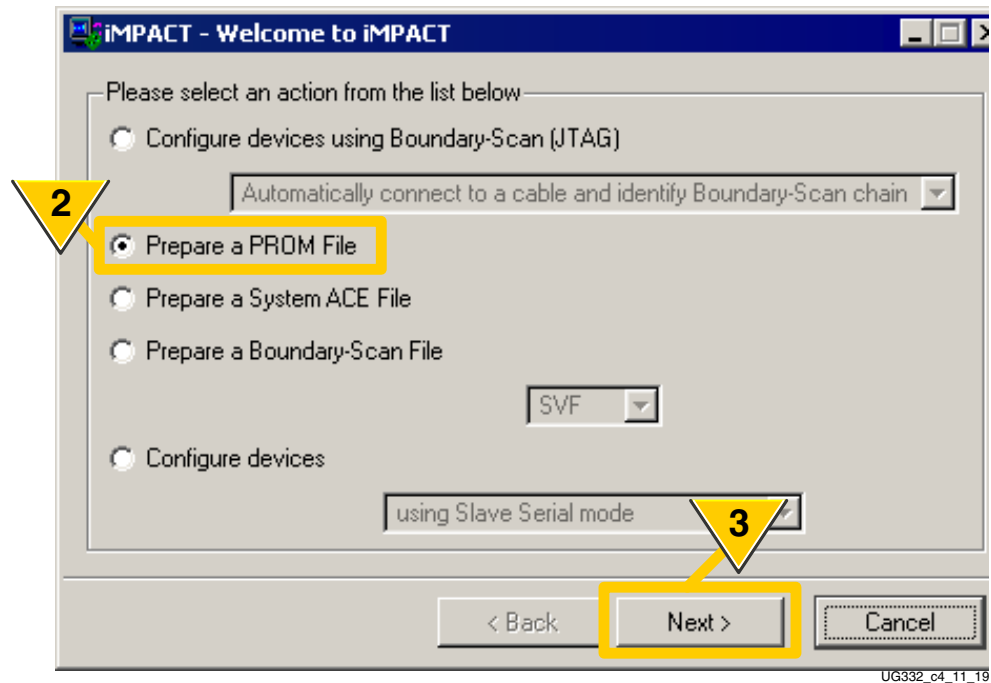


Figure 4-10: Prepare a PROM File

- Click **Next**.
- As shown in [Figure 4-11](#), format the FPGA bitstream or bitstreams for a **3rd-Party SPI PROM**. This option automatically invokes the `-spi` option for generating the PROM file.

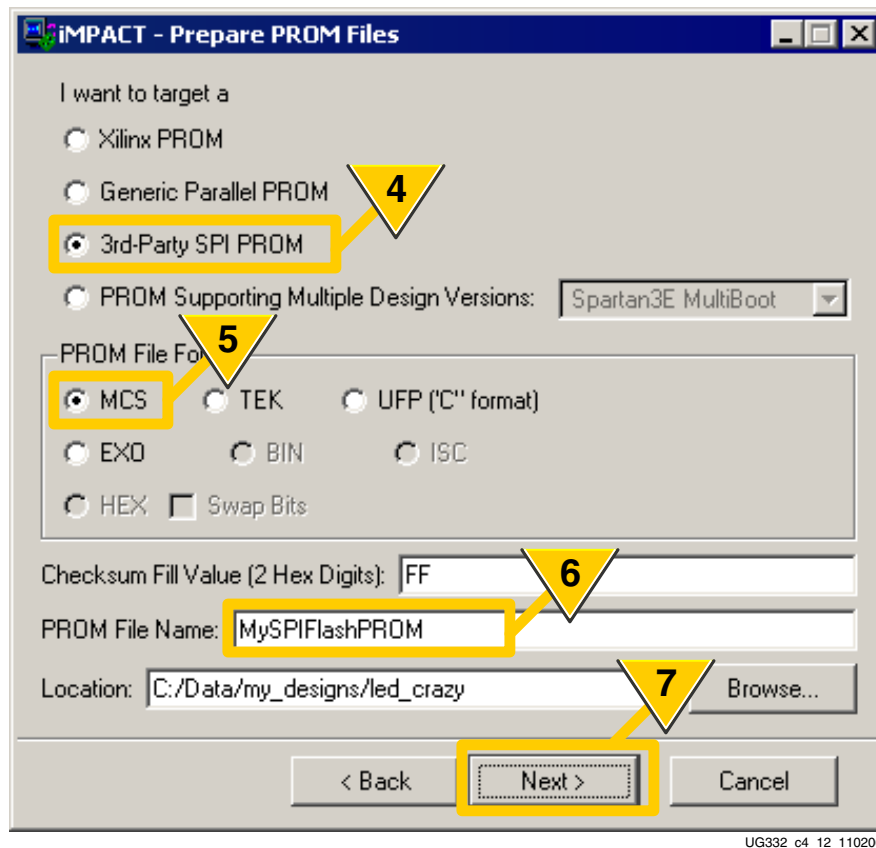


Figure 4-11: Set Options for a 3rd-Party SPI PROM

5. Select a **PROM File Format**.
6. Enter a **PROM File Name**.
7. Click **Next**.
8. As shown in Figure 4-12, select the **SPI PROM Density** of the targeted device, measured in bits.

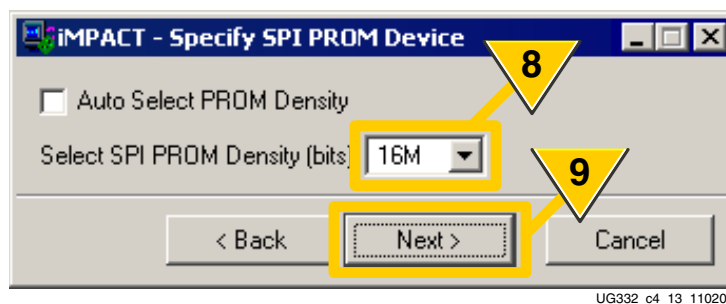


Figure 4-12: Select SPI PROM Density

9. Click **Next**.
10. As shown in Figure 4-13, review that the settings are correct to format the SPI PROM. Click **Finish** to confirm the settings or **Back** to change the settings.

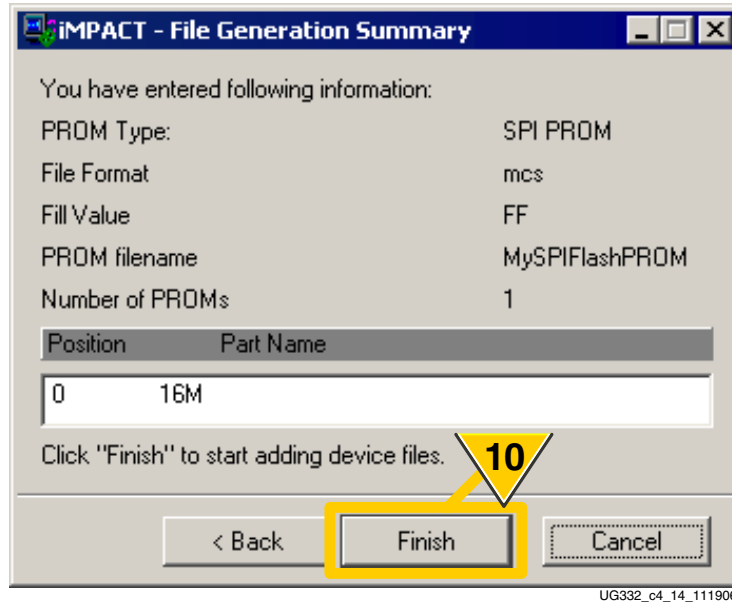


Figure 4-13: Review PROM Formatting Settings

- As shown in Figure 4-14, click OK to start adding FPGA configuration bitstreams to the PROM image.

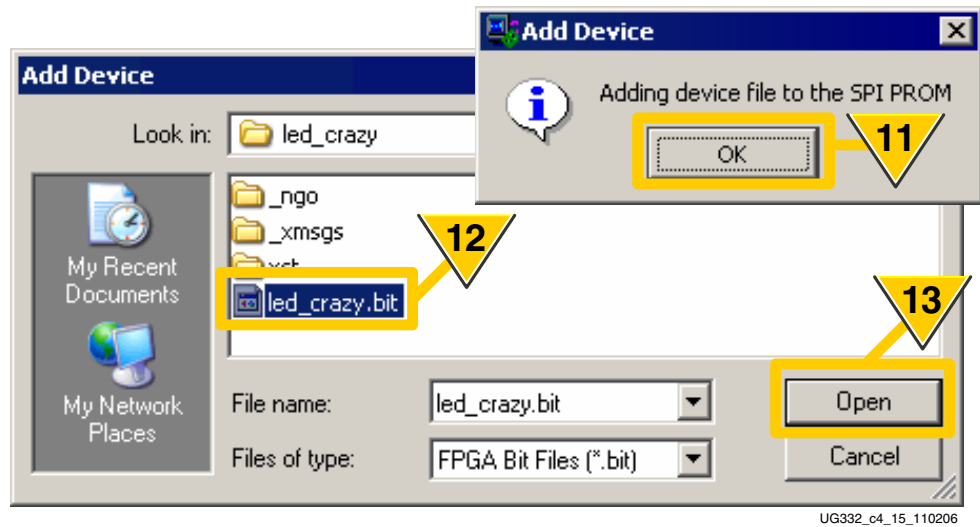


Figure 4-14: Add FPGA Configuration Bitstream File(s)

- Locate and select the desired FPGA bitstream.
- Click **Open**.
- As shown in Figure 4-15, the iMPACT software graphically displays the SPI PROM and associated FPGA bitstream(s).

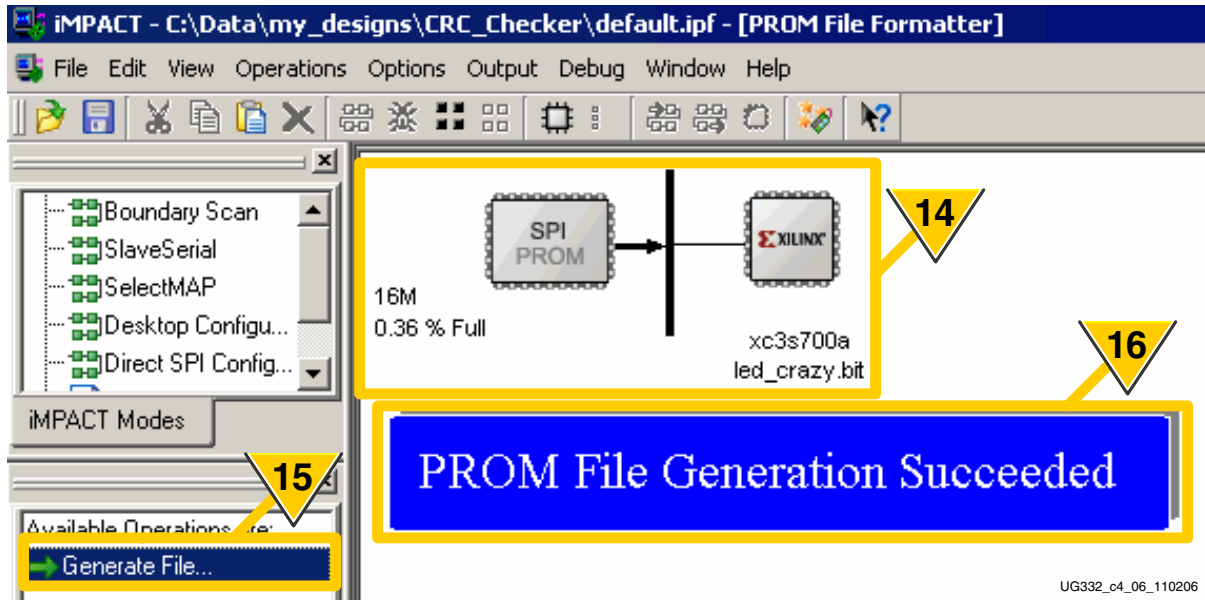


Figure 4-15: Generate PROM File

15. Click **Generate File**.
16. The iMPACT software indicates when the PROM file is successfully created.

PROMGen

PROMGen is a command-line utility that provides an alternate means to create an SPI PROM programming file. PROMGen can be invoked from within a command window or from within a script file.

Table 4-14 shows the relevant options for SPI Flash PROM formatting.

Table 4-14: PROM Generator Command Options

PROMGen Option	Description
-spi	REQUIRED FOR SPI FLASH PROMS! Specifies the correct bit ordering required to configure from an SPI Flash memory device.
-p <format>	PROM output file format. Specifies the file format required by the SPI programming software. Refer to the third party programmer documentation for details.
-s <size>	Specifies the PROM size in <u>kilobytes</u> . The PROM size must be a power of 2, and the default setting is 64 kilobytes.
-u <address>	Loads the .bit file from the specified starting address in an upward direction. This option must be specified immediately before the input bitstream file.

The example PROMGen command, provided below, generates an SPI-formatted PROM file with the following characteristics.

- Formatted for an SPI Flash PROM by specifying the **-spi** option.

- Formatted using the Intel MCS format by specifying the `-p mcs` option. The output filename is specified by the `-o <promdata>.mcs` option, where `<promdata>` is a user-specified file name.
- Formatted for a 16Mbit SPI PROM by specifying the `-s 2048` option. PROMGen specifies sizes in Kbytes.
- The specified FPGA bitstream is loaded in the upward direction, starting at address 0 by specifying the `-u 0` option.
- The FPGA bitstream to be formatted for the PROM is specified as the last option, `<inputfile>.bit`, where `<inputfile>` is the user-specified file name used when generating the FPGA bitstream.

```
promgen -spi -p mcs -o <promdata>.mcs -s 2048 -u 0 <inputfile>.bit
```

Direct SPI Programming using iMPACT

Starting with version 8.2i, the iMPACT programming software supports direct, in-system programming for SPI serial Flash PROMs. The SPI Flash memory devices that are tested and supported is indicated under the “Xilinx iMPACT Support” column in [Table 4-3, page 105](#).

Prepare Board for Programming

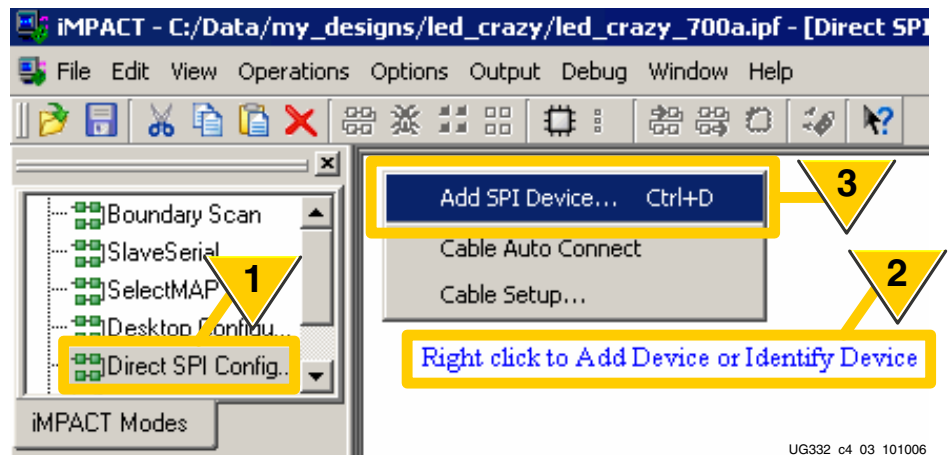
Before attempting to program the SPI PROM, complete the following steps.

1. Ensure that the board is powered.
2. Ensure that the FPGA pins that connect to the SPI Flash are high-impedance (Hi-Z). See [“Forcing FPGA SPI Bus Pins to High-impedance During Programming,” page 123](#).
3. Ensure that the programming cable is properly connected both the board and to the computer or workstation. See [“Programmable Cable Connections,” page 122](#).

Programming via iMPACT

The following steps describe how to program the SPI PROM using the iMPACT software and a Xilinx programming cable.

1. Click **Direct SPI Configuration** from within iMPACT, as shown in [Figure 4-16](#).



UG332_c4_03_101006

Figure 4-16: iMPACT Supports Direct Programming for SPI Serial Flash Memories.

2. Right-click in the area indicated.
3. Select **Add SPI Device**.
4. Select a previously-formatted PROM file, as shown in [Figure 4-17](#).

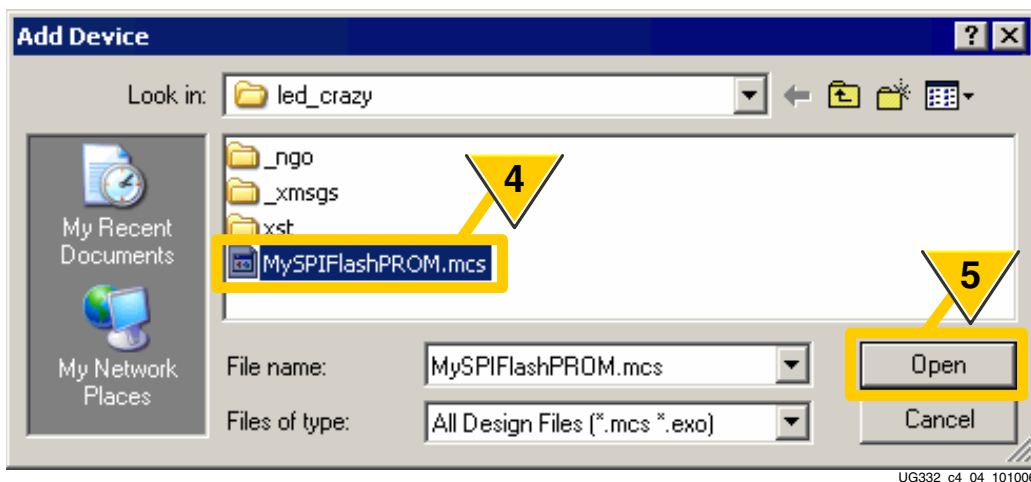


Figure 4-17: Select a Previously-formatted PROM File

5. Click **Open**.
6. Select the **Part Name** for a supported SPI serial Flash, as shown in [Figure 4-18](#).

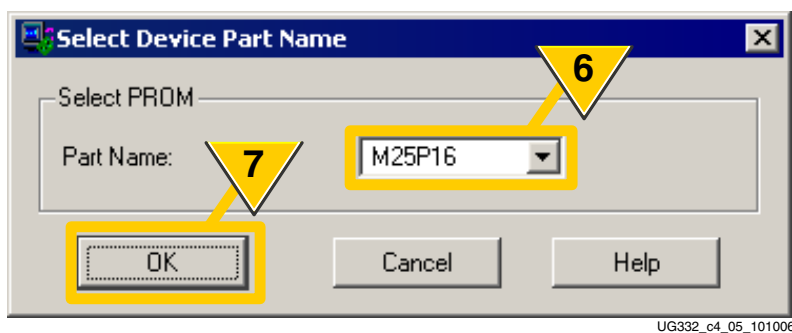


Figure 4-18: Select a Supported SPI Flash Memory Device.

7. Click **OK**.
8. The iMPACT software displays the selected SPI Flash PROM, as shown in [Figure 4-19](#).

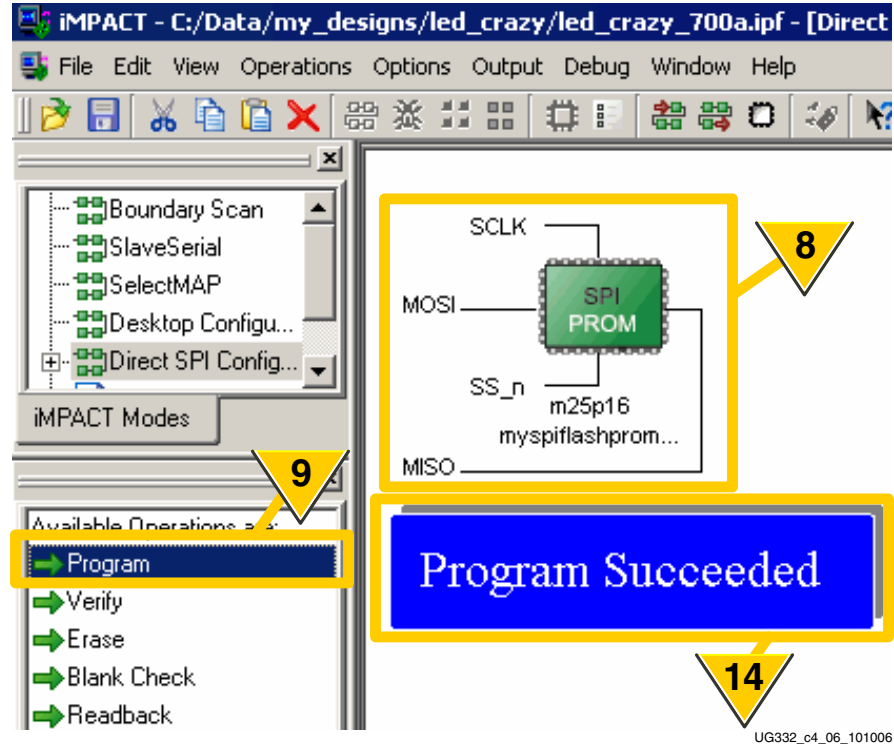


Figure 4-19: Directly Program Supported SPI Flash PROM.

9. Click **Program**.

Note: Step 14 occurs later.

10. Click the **Programming Properties** option under **Category**, as shown in Figure 4-20.

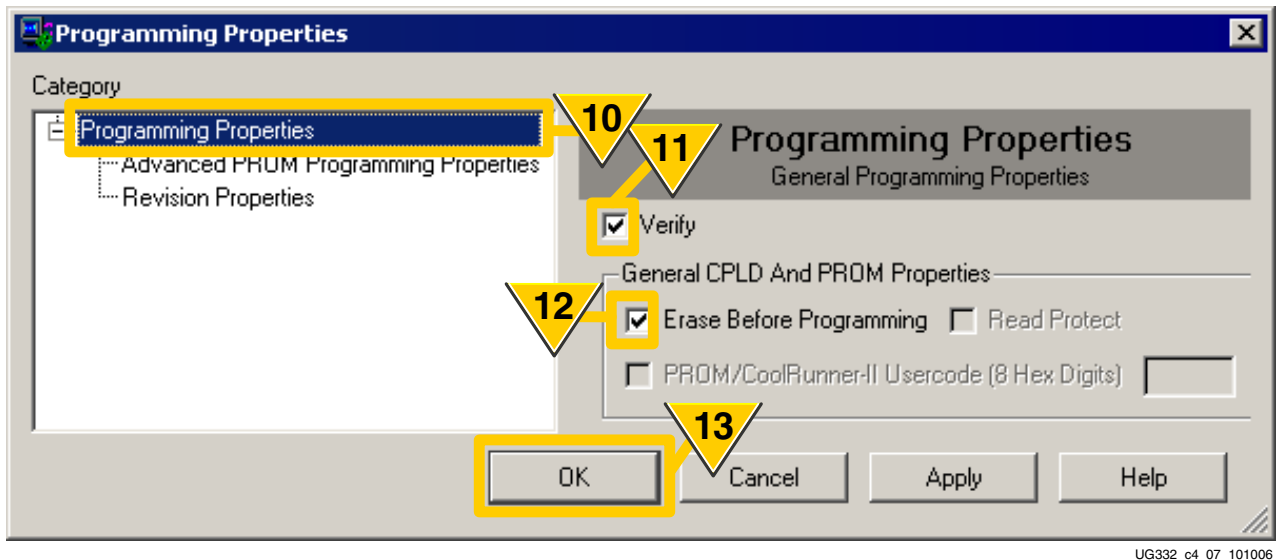


Figure 4-20: SPI PROM Programming Options

11. Check **Verify**. Unchecking Verify reduces programming time but the iMPACT software can only guarantee correct programming for a verified PROM.

12. Check **Erase Before Programming**. Unchecking the Erase option reduces programming time. However, Xilinx recommends erasing the PROM when downloading a new FPGA bitstream.
13. Click **OK**.
14. The iMPACT software indicates successful programming, as shown in [Figure 4-19](#).

Indirect SPI Programming using iMPACT

Indirect programming support is available starting with Xilinx ISE 9.1i, Service Pack 2 and later releases for the Spartan-3A and Spartan-3A DSP FPGAs. The Spartan-3E FPGAs are supported starting with the ISE 10.1 release. iMPACT supports programming of the internal Flash in Spartan-3AN FPGAs. In Indirect mode, the iMPACT software programs the memory attached to the FPGA through the FPGA's JTAG port. For details, see the following application note:

- **XAPP974: Indirect Programming of SPI Flash Serial PROMs with Spartan-3A FPGAs**
http://www.xilinx.com/support/documentation/application_notes/xapp974.pdf

During the programming process, the FPGA is configured with a special programming application. Consequently, the FPGA's DONE pin will go High during the programming process.

Programming Setup

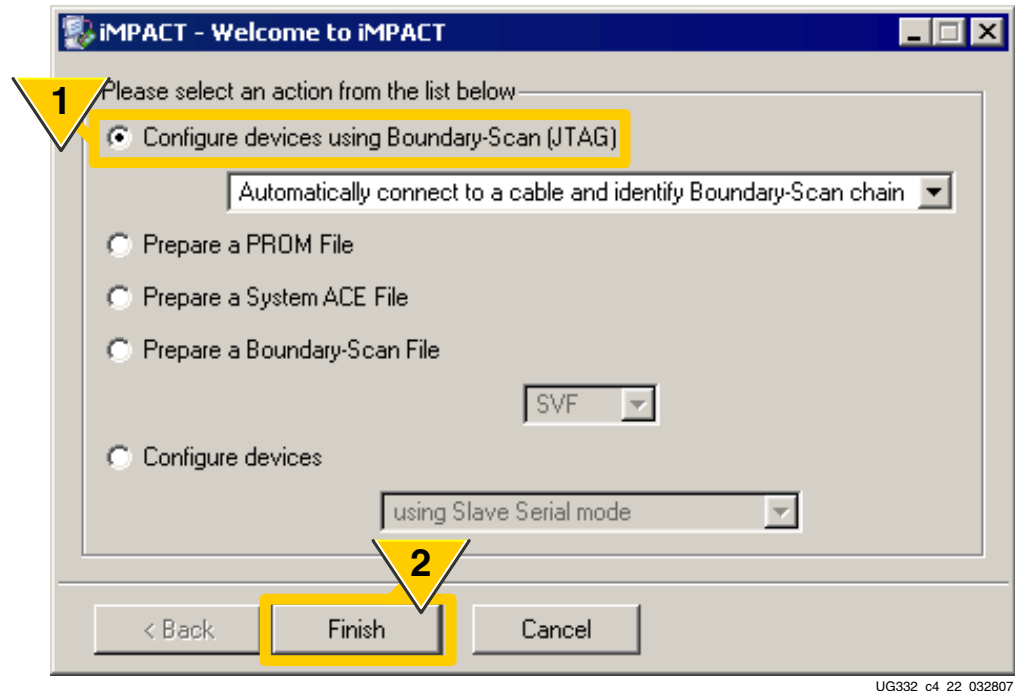
To program the attached and selected SPI PROM using the Indirect method, configure the board as described below.

1. Disconnect power to the board.
2. Set the FPGA mode select pins for Master SPI mode.
3. Connect the JTAG programming cable to the FPGA's JTAG port.
4. Re-apply power to the board.

Using iMPACT

To program the attached and selected SPI PROM using the iMPACT software and the Indirect programming method, follow the steps outlined below. This specific example uses the Spartan-3A FPGA Starter Kit board, which has an XC3S700A FPGA connected to an XCF04S Platform Flash PROM on the JTAG chain.

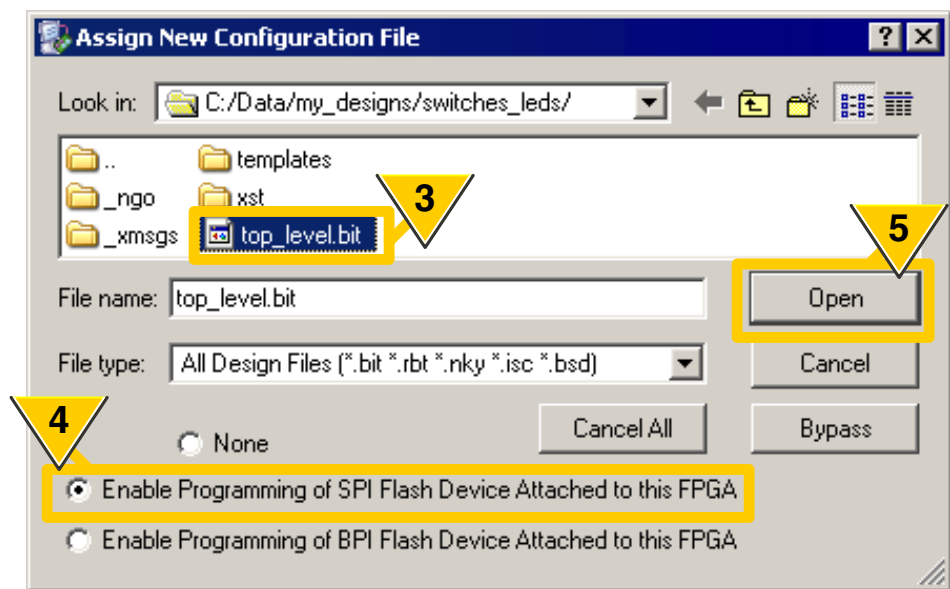
1. Invoke iMPACT and select **Configure devices using Boundary Scan (JTAG)**, as shown in [Figure 4-21](#).



UG332_c4_22_032807

Figure 4-21: Indirect Programming Method Uses JTAG

2. Select **Finish**.
3. (iMPACT 9.1i only) Select the FPGA bitstream file (*.bit) to be programmed into the FPGA, as shown in Figure 4-22. This step is superfluous but required for iMPACT 9.1i. This step is eliminated as of iMPACT 9.2i. This file is *not* the special FPGA-based SPI programming application.

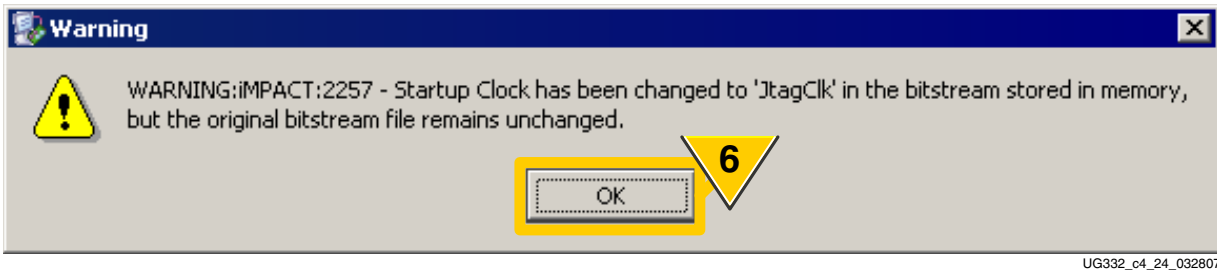


UG332_c4_23_032807

Figure 4-22: Select the FPGA Bitstream File and Enable SPI Programming

4. Select **Enable Programming of SPI Flash Device Attached to this FPGA**.

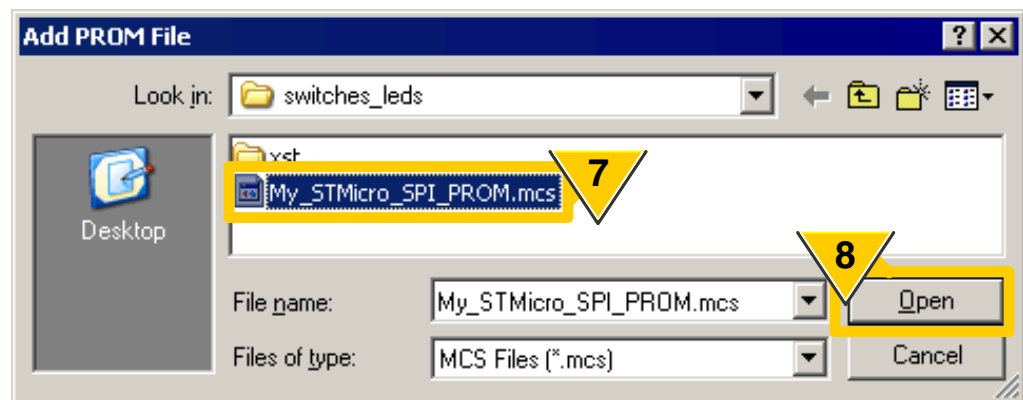
5. Click **Open**.
6. The iMPACT software warns that it changed the Startup clock source over to the JTAG clock pin, TCK. The SPI Flash image is not affected. This warning is safely ignored.



UG332_c4_24_032807

Figure 4-23: iMPACT Uses the JTAG Clock Input TCK for Startup Clock when Programming via JTAG

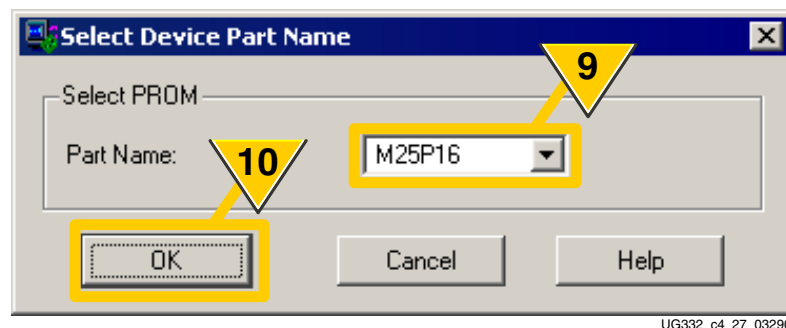
7. As shown in Figure 4-24, select the programming file for the attached SPI Flash PROM.



UG332_c4_26_032907

Figure 4-24: Select the SPI PROM Programming File

8. Click **Open**.
9. Select the part number for the attached SPI Flash PROM, as shown in Figure 4-25.



UG332_c4_27_032907

Figure 4-25: Select SPI Flash PROM Type

10. Click **OK**.
11. Select **Bypass** when prompted for the Platform Flash PROM programming file, as shown in Figure 4-26.

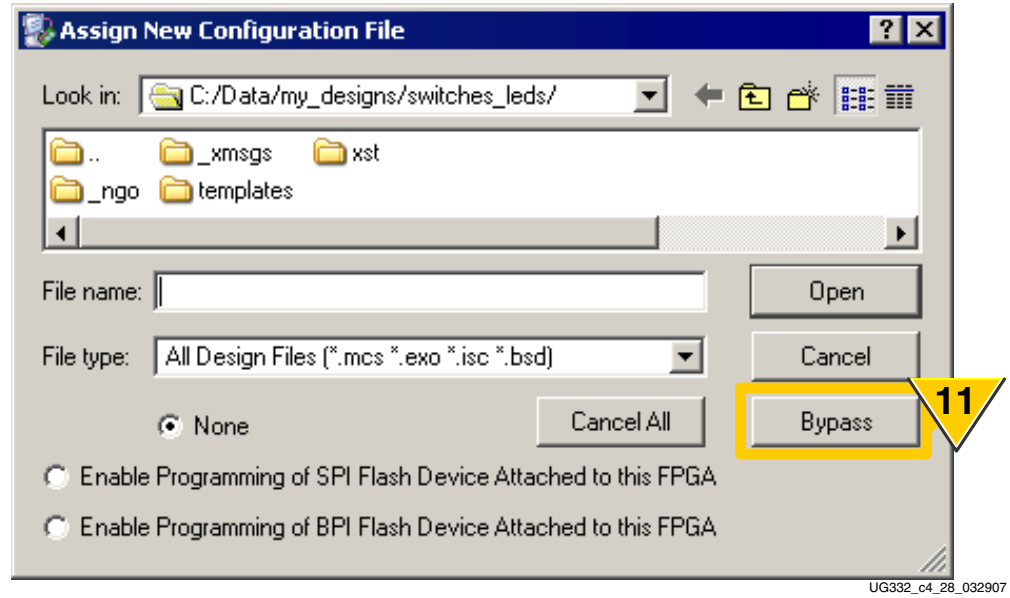


Figure 4-26: Bypass the Platform Flash PROM

12. As shown in Figure 4-27, the iMPACT software then displays the JTAG chain for the XC3S700A Spartan-3A FPGA followed by the XCF04S Platform Flash PROM. Click to highlight the FLASH memory attached to the XC3S700A FPGA. This action enables the command options shown in Step 13.

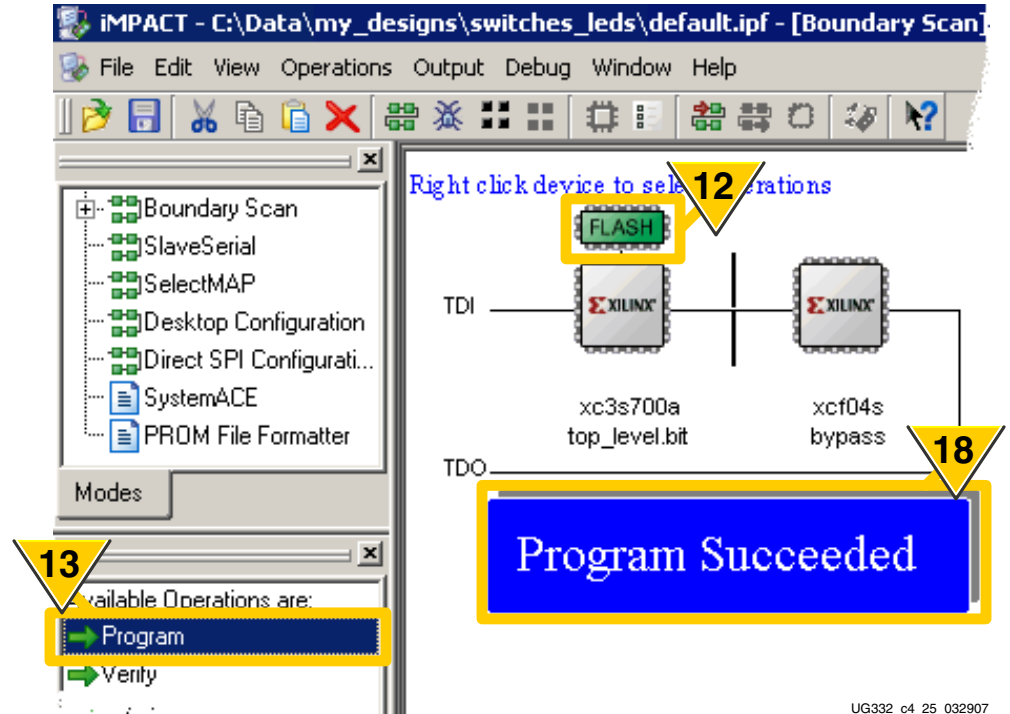


Figure 4-27: iMPACT Presents JTAG Chain, Shows Attached Flash PROM

13. Double-click **Program**.

Note: Step 18 occurs later.

14. Click the **Programming Properties** option under **Category**, as shown in [Figure 4-28](#).

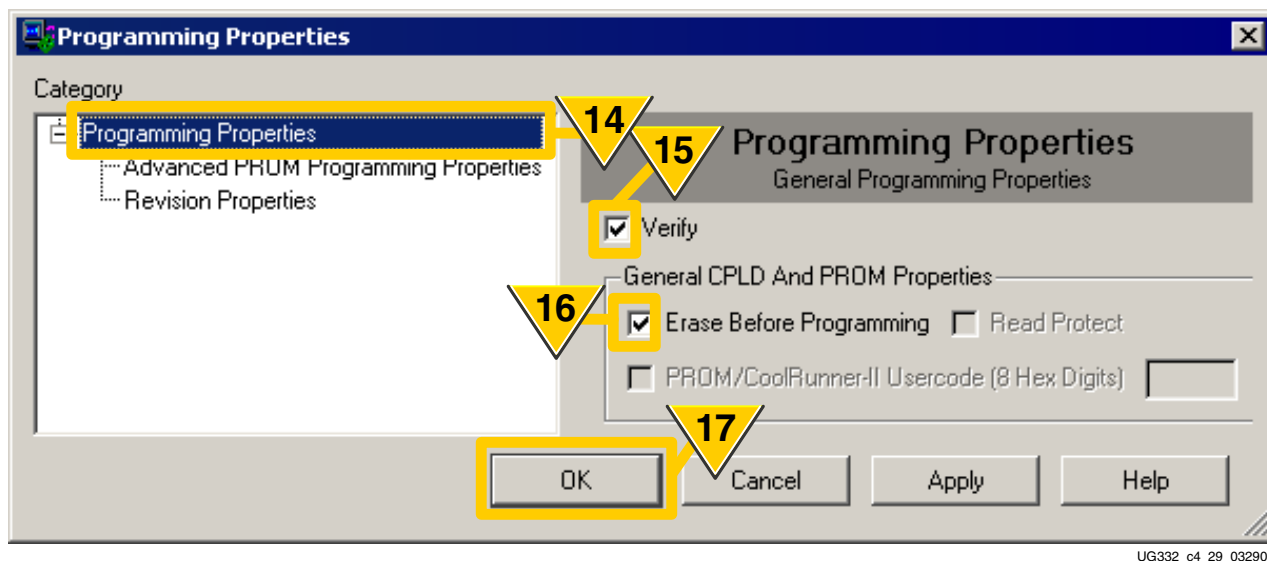
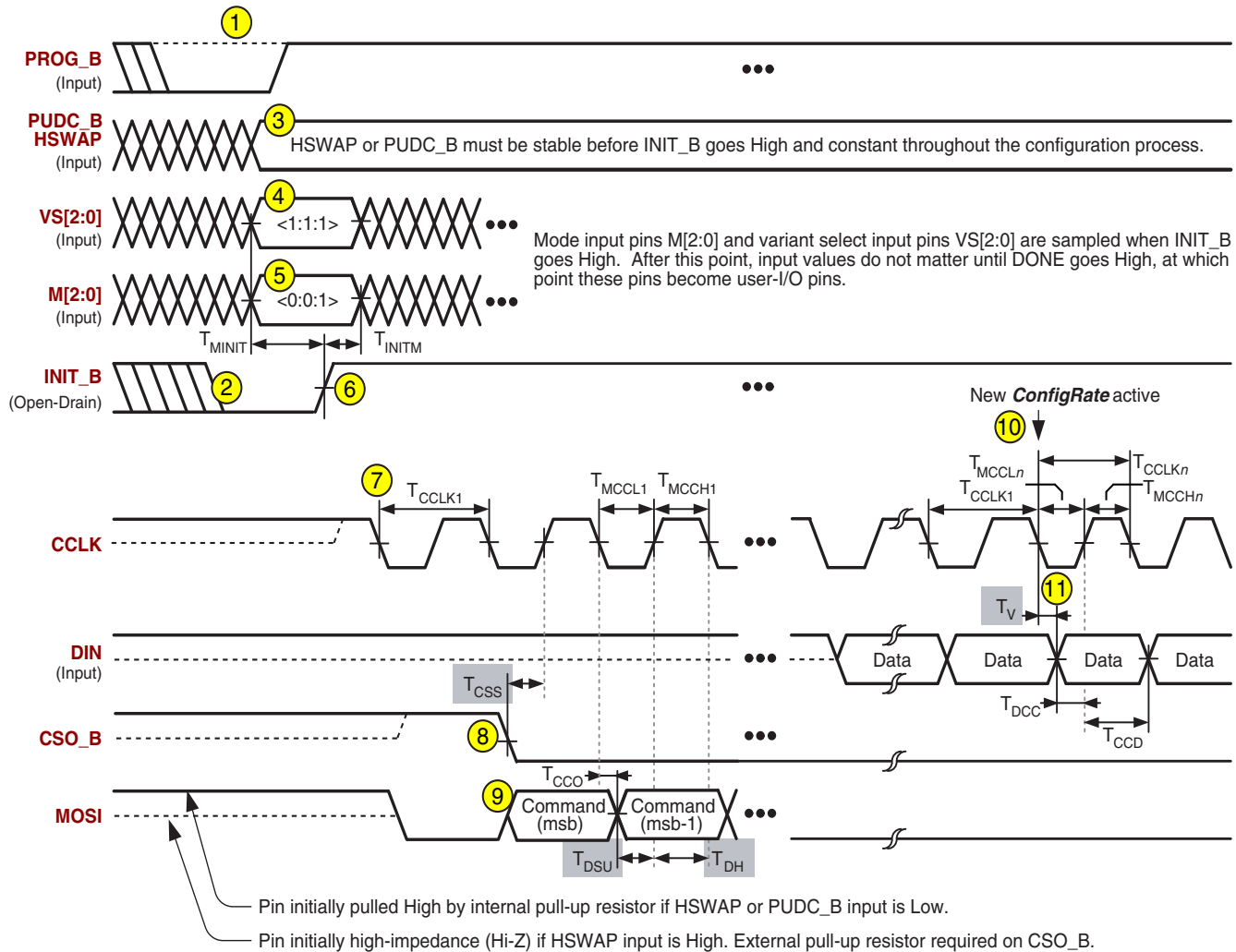


Figure 4-28: SPI PROM Programming Options

15. Check **Verify**. Unchecking Verify reduces programming time but the iMPACT software can only guarantee correct programming for a verified PROM.
16. Check **Erase Before Programming**. Unchecking the Erase option reduces programming time. However, Xilinx recommends erasing the PROM when downloading a new FPGA bitstream.
17. Click **OK**.
18. The iMPACT software indicates successful programming, as shown in [Figure 4-28](#). The FPGA is configured with the new programming file.

Serial Peripheral Interface (SPI) Configuration Timing

[Figure 4-29](#) provides example waveforms for Master SPI configuration. The following items correspond to the numbered markers in [Figure 4-29](#). The symbols for the FPGA timing parameters are listed in [Table 4-15](#). The required SPI Flash PROM timing and the dependencies on FPGA timing is provided in [Table 4-16](#), [page 141](#).



Shaded values indicate specifications on attached SPI Flash PROM.

UG332_c4_17_110206

Figure 4-29: Waveforms for Serial Peripheral Interface (SPI) Configuration

1. The FPGA powers on, releasing the internal Power-On Reset (POR) circuit or the **PROG_B** input returns High.
2. The FPGA begins clearing its internal configuration memory. The FPGA actively drives the **INIT_B** output Low.
3. Ensure that **HSWAP** or **PUDC_B** is at a stable logic level throughout the configuration process. The value on this input pin defines whether pull-up resistors are enabled during configuration. Some applications may depend on the pull-up resistors to define the **VS[2:0]** variant-select pins and to hold **CSO_B** High before the FPGA actively drives it Low.
4. The **VS[2:0]** variant-select pins must be defined and stable before the **INIT_B** pin returns High. The value on **VS[2:0]** defines the specific read command that the FPGA issues to the SPI serial PROM. See [Table 4-2, page 105](#).
5. The **M[2:0]** mode-select pins must be defined for Master SPI mode (<0:0:1>) and stable before the **INIT_B** pin returns High.

6. After the FPGA completes clearing the internal configuration memory, the FPGA release the `INIT_B` pin, allowing it to float High via the dedicated internal pull-up resistor to `VCCO_2`.
7. After the `INIT_B` pin returns High, the FPGA begins toggling the `CCLK` output, which controls all the configuration timing. The `CCLK` output initially starts at its lowest, default frequency, approximately 1 MHz.
8. The SPI Flash requires a High-to-Low transition on the `CSO_B` output. The FPGA actively drives the `CSO_B` output High for one `CCLK` cycle before asserting the `CSO_B` pin Low. This begins the SPI bus transaction.
9. Based on the `VS[2:0]` pin values sampled when `INIT_B` pin returned High, the FPGA begins issuing a SPI Flash read command. The FPGA sends the command, most-significant bit first. The FPGA subsequently sends a 24-bit address, all zeros, and the appropriate number of dummy bits, also zero, for the select Flash memory. The FPGA clocks out the command, address, and dummy bits on the `MOSI` output, clocked on the falling edge of `CCLK`.
10. Within the first 384 bits of the configuration bitstream, the FPGA loads the `ConfigRate` setting for the remainder of the configuration process. The `ConfigRate` setting defines the `CCLK` frequency. All interface timing must be evaluated for the specific setting. See “`CCLK Frequency`,” page 114 and “`ConfigRate: CCLK Frequency`,” page 125.
11. The SPI Flash PROM provides data on the falling edge of `CCLK`. This PROM data must be valid and setup on the FPGA’s `DIN` serial data input before next rising edge of `CCLK`.

Table 4-15 lists the various FPGA timing parameters associated with the SPI configuration interface.

Table 4-15: FPGA Timing Symbols for Serial Peripheral Interface (SPI) Configuration Mode

Symbol	Description
T_{CCLK1}	Initial <code>CCLK</code> clock period
T_{CCLKn}	<code>CCLK</code> clock period after FPGA loads <code>ConfigRate</code> setting
T_{MINIT}	Setup time on the <code>VS[2:0]</code> variant-select pins and the <code>M[2:0]</code> mode-select pins before the rising edge of <code>INIT_B</code>
T_{CCLKL1}	Minimum <code>CCLK</code> Low time at the initial, default <code>ConfigRate</code> setting
T_{CCLKLn}	Minimum <code>CCLK</code> Low time at the <code>ConfigRate</code> setting specified in the FPGA bitstream.
T_{INITM}	Hold time on the <code>VS[2:0]</code> variant-select pins and the <code>M[2:0]</code> mode-select pins before the rising edge of <code>INIT_B</code>
T_{CCO}	<code>MOSI</code> output valid delay after <code>CCLK</code> falling clock edge
T_{DCC}	Setup time on the <code>DIN</code> data input before <code>CCLK</code> rising clock edge
T_{CCD}	Hold time on the <code>DIN</code> data input after <code>CCLK</code> rising clock edge

Table 4-16 shows the relationship between the SPI Flash PROM timing specifications and the FPGA’s configuration timing specifications. For example, the SPI Flash clock-to-output time, T_V , must be less than or equal the FPGA minimum `CCLK` Low time and the specified `ConfigRate` setting, T_{CCLKLn} , minus the FPGA’s setup time on the `DIN` input, T_{DCC} . See the T_V parameter highlighted in Figure 4-29, page 139.

All communication from the FPGA to the SPI Flash PROM, i.e., sending the read command, address, and dummy bits, all occurs at the default **CCLK ConfigRate** setting, which is the slowest setting in the Spartan-3E FPGAs, **TCCLK1**, which equates to approximately 1 MHz. The Extended Spartan-3A family families default to 6 MHz.

Table 4-16: Configuration Timing Requirements for Attached SPI Serial Flash

Symbol	Description	Requirement	Units
T_{CCS}	SPI serial Flash PROM chip-select time	$T_{CCS} \leq T_{MCCL1} - T_{CCO}$	ns
T_{DSU}	SPI serial Flash PROM data input setup time	$T_{DSU} \leq T_{MCCL1} - T_{CCO}$	ns
T_{DH}	SPI serial Flash PROM data input hold time	$T_{DH} \leq T_{MCCH1}$	ns
T_V	SPI serial Flash PROM data clock-to-output time	$T_V \leq T_{MCCLn} - T_{DCC}$	ns
f_C or f_R	Maximum SPI serial Flash PROM clock frequency (also depends on specific read command used)	$f_C \geq \frac{1}{T_{CCLKn(min)}}$	MHz

Notes:

1. These requirements are for successful FPGA configuration in SPI mode, where the FPGA provides the **CCLK** frequency. The post-configuration requirements may be different, depending on the application loaded into the FPGA and the resulting clock source.
2. Subtract additional printed circuit board routing delay as required by the application.

Multi-Package Layout

Most of the SPI PROM vendors have a multi-package migration scheme that allows a design to migrate to larger or smaller memory densities.

The multi-package layout provides ...

- **Density migration between smaller- and larger-density SPI Flash PROMs.** Not all SPI Flash memory densities are available in all packages. The SPI Flash migration strategy follows nicely with the pinout migration provided by Xilinx FPGAs. Should the application need more nonvolatile storage, there is always a convenient, upward density migration path in the SPI Flash PROM, up to 128Mbits.
- **Consistent configuration PROM layout when migrating between FPGA densities.** Within the Extended Spartan-3A family FPGAs and within the Spartan-3E FPGA family, a particular FPGA package option spans different density levels while maintaining footprint compatibility. The SPI Flash multi-package layout allows comparable flexibility in the associated configuration PROM. Ship the optimally-sized SPI Flash memory for the specific FPGA mounted on the board.
- **Supply security.** If a certain SPI Flash density is not available in the desired package, switch to a different package style or to a different density to secure availability. Likewise, multiple vendors support the STMicroelectronics (Numonyx) footprint.

An example package layout for the M25Pxx SPI serial Flash family, from the [Spartan-3E FPGA Starter Kit Board](#), is provided in [Figure 4-30](#). The multi-package layout supports the 8-lead 8x6 mm MLP package, the 8-pin SOIC package and the 16-pin SOIC package. Pin 1 for the 8-pin SOIC and MLP packages is located in the top-left corner. However, pin 1 for the 16-pin SOIC package is located in the top-right corner, because the package is rotated

90°. The 16-pin SOIC package also has four pins at the center each side that do not connect on the board. These pins must be left unconnected, *i.e.* floating.

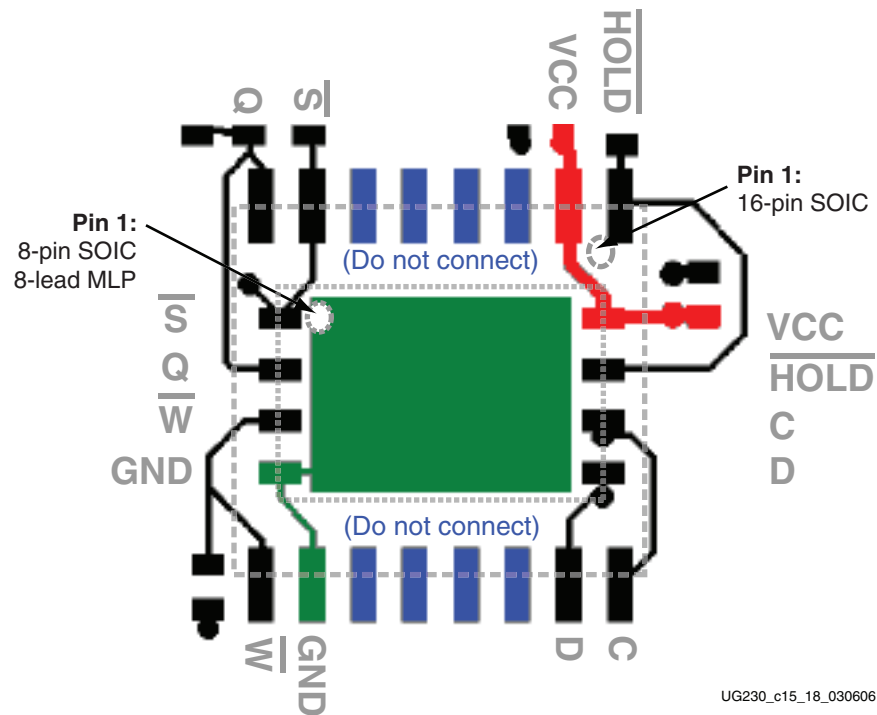


Figure 4-30: Multi-Package Layout for the M25Pxx Family on Spartan-3E Starter Kit

Saving Power

Most SPI Flash memories support multiple power-saving options. The simplest and most useful is the **Standby Mode**, which reduces power simply by de-selecting the SPI Flash memory. Within the FPGA application, drive the **CSO_B** pin High.

Deassert CSO_B to Enter Standby Mode

The SPI Flash memory automatically enters Standby power mode when the memory's active-Low **Slave Select** line is deasserted High. After configuration or when not accessing the SPI Flash, the application must drive the **CSO_B** pin High.

Master BPI Mode

Overview

The master Byte-wide Peripheral Interface (BPI) configuration mode is available for either the Spartan®-3A/3AN/3A DSP and Spartan-3E FPGA families. It is not supported on the Spartan-3 FPGA family although there is a similar mode that leverages Xilinx® Parallel Platform Flash PROMs (see [Chapter 6, “Master Parallel Mode”](#)).

In BPI mode, a Spartan-3E or Extended Spartan-3A family FPGA configures itself from a standard parallel NOR Flash PROM, as illustrated in [Figure 5-1, page 144](#) for Spartan-3E FPGAs and [Figure 5-2, page 145](#) for Extended Spartan-3A family FPGAs. The figures show optional components in gray and designated “NO LOAD”.

The BPI configuration interface is primarily designed to support standard parallel NOR Flash PROMs and the interface supports both byte-wide (x8) and byte-wide/word-wide (x8/x16) PROMs. In a pinch, the interface also functions with word-only (x16) PROMs, but the upper byte in a portion of the PROM remains unused. For FPGA configuration, the BPI interface does not require any specific Flash PROM features, such as a boot block or a specific sector size.

The BPI interface also works equally well with other asynchronous memories that use a similar SRAM-style interface such as the following, many of which have faster access times.

- Xilinx Parallel [Platform Flash PROMs](#) (XCFxxP)
- SRAM
- NVRAM (non-volatile RAM)
- EEPROM
- EPROM
- Masked ROM

NAND Flash memory is a different technology and is commonly used in memory cards for digital cameras. Extended Spartan-3A family and Spartan-3E FPGAs do not configure directly from NAND Flash memories.

The FPGA's internal oscillator controls the interface timing and the FPGA supplies the clock on the CCLK output pin. However, the CCLK signal typically is not connected in single FPGA applications. The FPGA drives three pins Low during configuration (LDC[2:0]) and one pin High during configuration (HDC) to the PROM's control inputs.

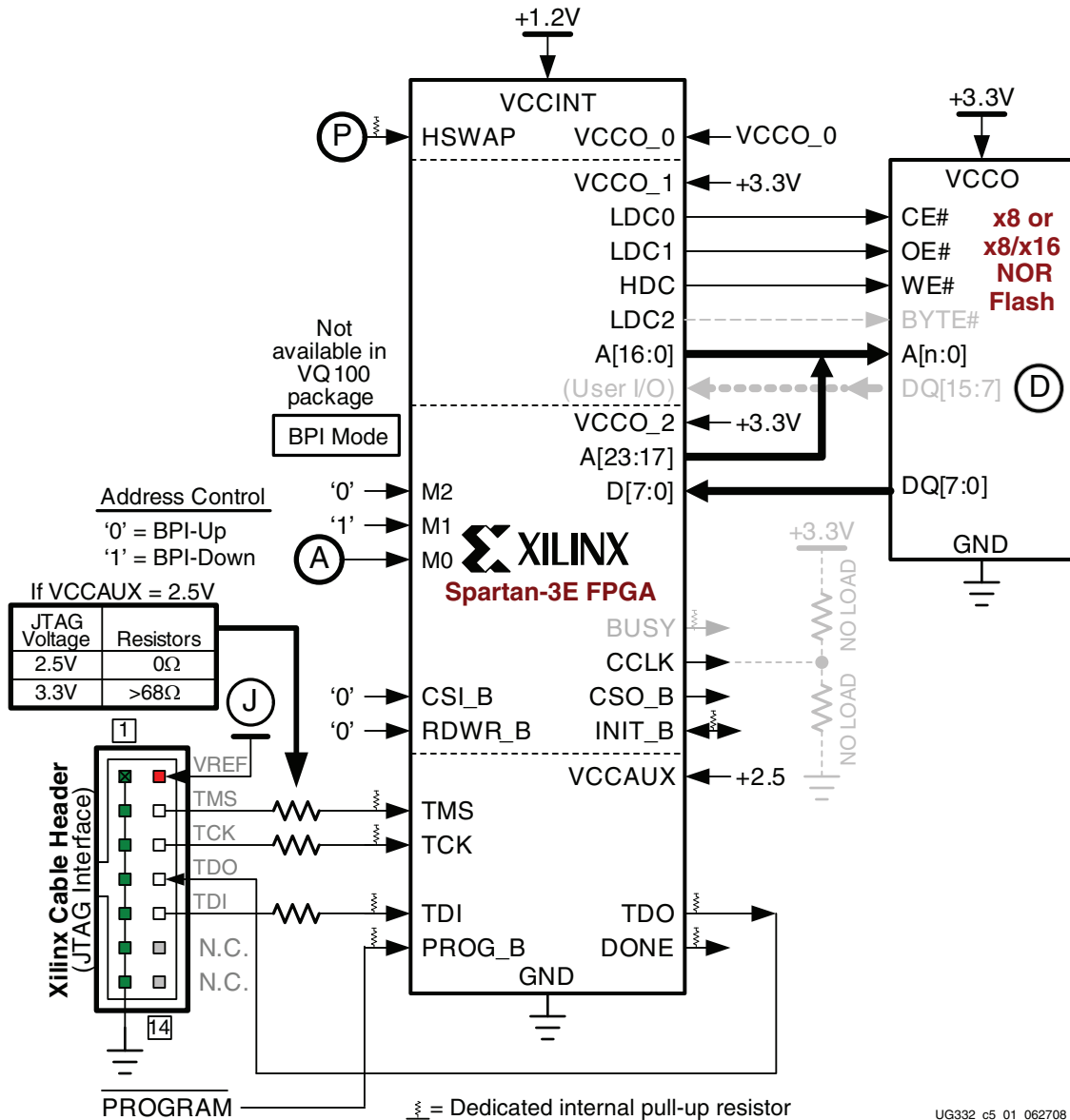


Figure 5-1: Spartan-3E FPGA Configured from Parallel NOR Flash

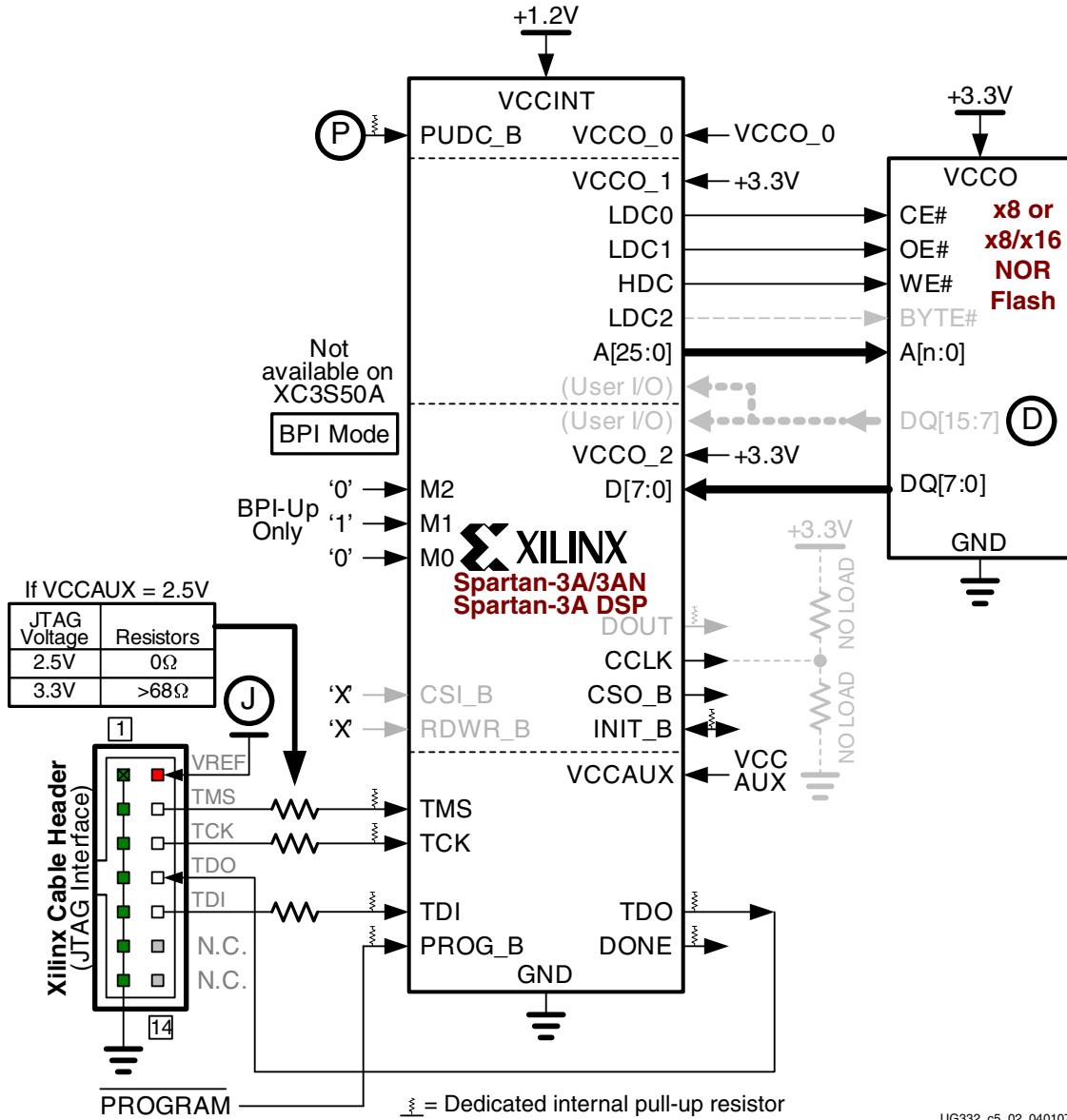


Figure 5-2: Extended Spartan-3A Family FPGA Configured from Parallel NOR Flash

Master BPI Mode Differences between Spartan-3 Generation FPGA Families

Table 5-1 summarizes the BPI configuration mode differences between various Spartan-3 generation FPGA families. BPI mode is only available on the Spartan-3E and Extended Spartan-3A family FPGA families. The Extended Spartan-3A family BPI mode supports up to 26 address lines, capable of addressing up to 512 Mbits (64 KBytes).

Table 5-1: BPI Configuration Mode Differences between Spartan-3 Generation FPGA Families

	Spartan-3 FPGA	Spartan-3E FPGA	Spartan-3A/3AN Spartan-3A DSP FPGA
BPI Up mode supported (start at 0, increment addresses)	BPI Mode not available on Spartan-3 FPGA family	Yes	Yes
BPI Down mode supported (start at highest location, decrement addresses)		Yes	No
Maximum number of address lines supplied by FPGA		24	26
FPGA I/O Banks used for address lines		Banks 1 and 2	Bank 1 only
Address lines independent of Right-edge Clock inputs (RHCLKs)		No	Yes
Parallel daisy chains supported		Yes	Yes
Serial daisy chains supported		No	Yes
Supports MultiBoot configuration		Yes	Yes
Watchdog Timer retry		No	Yes
Number of interface timing options, controlled by <i>ConfigRate</i> setting (see Table 5-6)		3	12
CCLK directionality during Master BPI mode		I/O	Output only for improved signal integrity
<i>RDWR_B</i> and <i>CSL_B</i> required during configuration		Yes	No (don't care)
<i>M[2:0]</i> pins have dedicated internal pull-up resistors during configuration		No Optional, controlled by <i>HSWAP</i>	Yes

PROM Address Generation

Extended Spartan-3A family FPGAs always start configuration from address 0 with incrementing addresses, a mode called **BPI Up**. Extended Spartan-3A family FPGAs always set $M[2:0] = \langle 0:1:0 \rangle$ for BPI mode.

A As shown in [Figure 5-1, page 144](#), the Spartan-3E FPGA family supports two versions of BPI configuration, defined by the M0 mode select pin. As shown in [Table 5-2, page 147](#), when the M0 mode-select pin is Low, a Spartan-3E FPGA configures using **BPI Up** mode, starting at address 0 and incrementing the addresses presented on the $A[23:0]$ address pins. When the M0 mode-select pin is High, a Spartan-3E FPGA configures using the **BPI Down** mode, starting from the highest memory location ($A[23:0] = 0xFFFFFFFF$) and automatically decrementing the memory addresses. Extended Spartan-3A family FPGAs do not support **BPI Down** mode.

Addresses are generally incremented (or decremented for **BPI Down** mode) on every falling **CCLK** edge. The exception is when using Spartan-3A FPGAs as part of a serial daisy chain (see “[Serial Daisy Chaining \(Extended Spartan-3A Family FPGAs Only\)](#),” page 157).

Table 5-2: BPI Addressing Control

M2	M1	M0	Mode	Supported Families	Start Address	Addressing
0	1	0	BPI Up	Spartan-3A/3AN, Spartan-3A DSP, Spartan-3E FPGAs	0	Incrementing
		1	BPI Down	Spartan-3E FPGAs only	0xFF_FFFF	Decrementing

The Spartan-3E addressing flexibility allows the FPGA to share the parallel Flash PROM with an external or embedded processor. Depending on the specific processor architecture, the processor boots either from the top or bottom of memory. The FPGA is flexible and boots from the opposite end of memory from the processor. Only the processor or the FPGA can boot at any given time. The FPGA can configure first, holding the processor in reset or the processor can boot first, asserting the FPGA’s **PROG_B** pin.

Spartan-3E FPGAs generally provide up to 24 address lines to access an attached parallel memory. There are a few exceptions as described below.

- Spartan-3E FPGAs available in the TQ144 package only provide 20 address lines, which is more than sufficient for the smaller FPGA array sizes offered in the TQ144 package.
- Similarly, the XC3S100E FPGA in the CP132 package only has 20 address lines while the XC3S250E and XC3S500E FPGAs in the same package have 24 address lines.
- The BPI address pins are not provided on Spartan-3E FPGAs offered in the VQ100. Consequently, Spartan-3E FPGAs in the VQ100 package cannot configure from a parallel NOR Flash, but can configure using parallel Xilinx Platform Flash (XCFxxP).

Extended Spartan-3A family FPGAs generally provide up to 26 address lines to access an attached parallel memory. There are a few exceptions as described below.

- The XC3S50A FPGA does not support BPI mode.

As shown in [Figure 5-14, page 169](#), the mode select pins, **M[2:0]**, are sampled when the FPGA’s **INIT_B** output goes High and must be at defined logic levels during this time. After configuration, when the FPGA’s **DONE** output goes High, the mode pins are available as full-featured user-I/O pins.

P Similarly, the FPGA’s **HSWAP** pin must be Low to enable pull-up resistors on all user-I/O pins or High to disable the pull-up resistor. The **HSWAP** or **PUDC_B** control must remain at a constant logic level throughout FPGA configuration. After configuration, when the FPGA’s **DONE** output goes High, the **HSWAP** or **PUDC_B** pin is available as full-featured user-I/O pin and is powered by the **VCCO_0** supply.

On Spartan-3E FPGAs, the **RDWR_B** and **CSL_B** pins must be Low throughout the configuration process, although the start of configuration is delayed until **CSL_B** is asserted. After configuration, these pins also become user I/O. The **RDWR_B** and **CSL_B** are not used and are ignored on Extended Spartan-3A family FPGAs.

In a single-FPGA application, the FPGA’s **CSO_B** and **CCLK** pins are not used but are actively driving during the configuration process. The Spartan-3E **BUSY** pin, not available on Extended Spartan-3A family FPGAs, is not used but actively drives during configuration and is available as a user I/O after configuration.

After configuration, all of the interface pins except [DONE](#) and [PROG_B](#) are available as user I/Os.

Table 5-3: Byte-Wide Peripheral Interface (BPI) Connections

Pin Name	FPGA Direction	Description	During Configuration	After Configuration
HSWAP PUDC_B (P)	Input	User I/O Pull-Up Control. When Low during configuration, enables pull-up resistors in all I/O pins to respective I/O bank V_{CC0} input. 0: Pull-ups during configuration 1: No pull-ups	Drive at valid logic level throughout configuration.	User I/O
M[2:0] (A)	Input	Mode Select. Selects the FPGA configuration mode. Extended Spartan-3A family FPGAs have dedicated internal pull-up resistors on these pins. See “ Design Considerations for the HSWAP, M[2:0], and VS[2:0] Pins, ” page 75.	M2 = 0, M1 = 1. Set M0 = 0 to start at address 0, increment addresses. On Spartan-3E FPGAs, optionally set M0 = 1 to start at address 0xFFFFFFFF and decrement addresses. Sampled when INIT_B goes High.	User I/O
Spartan-3E FPGAs only: CSL_B	Input	Chip Select Input. Active Low	Must be Low throughout configuration. This input is ignored on Extended Spartan-3A family FPGAs.	User I/O
Spartan-3E FPGAs only: RDWR_B	Input	Read/Write Control. Active Low write enable. Read functionality typically only used after configuration, if bitstream option <i>Persist:Yes</i> .	Must be Low throughout configuration. This input is ignored on Extended Spartan-3A family FPGAs.	User I/O
LDC0	Output	PROM Chip Enable	Connect to PROM chip-select input (CS#). FPGA drives this signal Low throughout configuration.	User I/O. If the FPGA does not access the PROM after configuration, drive this pin High to deselect the PROM. A[23:0] , D[7:0] , LDC2 , LDC1 , and HDC then become available as user I/O.
LDC1	Output	PROM Output Enable	Connect to the PROM output-enable input (OE#). The FPGA drives this signal Low throughout configuration.	User I/O
HDC	Output	PROM Write Enable	Connect to PROM write-enable input (WE#). FPGA drives this signal High throughout configuration.	User I/O

Table 5-3: Byte-Wide Peripheral Interface (BPI) Connections (Cont'd)

Pin Name	FPGA Direction	Description	During Configuration	After Configuration
LDC2 ⓓ	Output	PROM Byte Mode	This signal is not used for x8 PROMs. For PROMs with a x8/x16 data width control, connect to PROM byte-mode input (BYTE#). See “Precautions Using x8/x16 Flash PROMs”. FPGA drives this signal Low throughout configuration.	User I/O. Drive this pin High after configuration to use a x8/x16 PROM in x16 mode.
Spartan-3E FPGAs: A[23:0] Spartan-3A Spartan-3AN Spartan-3A DSP FPGAs: A[25:0]	Output	Address	Connect to PROM address inputs. High-order address lines may not be available in all packages and not all may be required. Number of address lines required depends on the size of the attached Flash PROM. Spartan-3E FPGA address generation controlled by M0 mode pin. Addresses presented on falling CCLK edge.	User I/O
D[7:0]	Input	Data Input	FPGA receives byte-wide data on these pins in response the address presented on A[23:0] or A[25:0]. Data captured by FPGA on rising edge of CCLK. D0 is the MSB (see “BPI Data Ordering”)	User I/O.
CSO_B	Output	Chip Select Output. Active Low.	Not used in single-FPGA applications. In a daisy-chain configuration, this pin connects to the CSI_B pin of the next FPGA in the chain. If HSWAP or PUDC_B = 1 in a multi-FPGA daisy-chain application, connect this signal to a 4.7 kΩ pull-up resistor to VCCO_2. Actively drives Low when selecting a downstream device in the chain.	User I/O
Spartan-3E: FPGAs BUSY	Output	Busy Indicator.	Not used in single-FPGA designs; BUSY is pulled up, not actively driving.	User I/O.
Spartan-3A Spartan-3AN Spartan-3A DSP FPGAs: DOUT	Output	Serial Data Output. Used in Extended Spartan-3A family serial daisy chains.	Not used in single-FPGA designs; DOUT is pulled up, not actively driving. In an Extended Spartan-3A family serial daisy-chain configuration, this pin connects to DIN input of the next FPGA in the chain.	User I/O.

Table 5-3: Byte-Wide Peripheral Interface (BPI) Connections (Cont'd)

Pin Name	FPGA Direction	Description	During Configuration	After Configuration
CCLK	Output	Configuration Clock. Generated by FPGA internal oscillator. Frequency controlled by <i>ConfigRate</i> bitstream generator option. If CCLK PCB trace is long or has multiple connections, terminate this output to maintain signal integrity. See “ CCLK Design Considerations ,” page 58.	Not used in single FPGA applications but actively drives. In a daisy-chain configuration, drives the CCLK inputs of all other FPGAs in the daisy chain.	User I/O. Drive High or Low if not used.
INIT_B	Open-drain bidirectional I/O	Initialization Indicator. Active Low. Goes Low at start of configuration during the Initialization memory clearing process. Released at the end of memory clearing, when the mode select pins are sampled.	Active during configuration. If CRC error detected during configuration, FPGA drives INIT_B Low.	User I/O. If unused in the application, drive INIT_B High to avoid a floating value. See INIT_B “ After Configuration ”.
DONE	Open-drain bidirectional I/O	FPGA Configuration Done. Low during configuration. Goes High when FPGA successfully completes configuration.	Low indicates that the FPGA is not yet configured.	Pulled High via external pull-up. When High, indicates that the FPGA is successfully configured.
PROG_B	Input	Program FPGA. Active Low. When asserted Low for 500 ns or longer, forces the FPGA to restart its configuration process by clearing configuration memory and resetting the DONE and INIT_B pins once PROG_B returns High.	Must be High to allow configuration to start.	Drive PROG_B Low and release to reprogram FPGA. Hold PROG_B to force FPGA I/O pins into Hi-Z, allowing direct programming access to Flash PROM pins.

Voltage Compatibility

Ⓥ The FPGA’s parallel Flash interface signals are within I/O Banks 1 and 2. The majority of parallel Flash PROMs use a single 3.3V supply voltage. Consequently, in most cases, the FPGA’s VCCO_1 and VCCO_2 supply voltages must also be 3.3V to match the parallel Flash PROM. There are some 1.8V parallel Flash PROMs available and Spartan-3E FPGAs interface with these devices if the VCCO_1 and VCCO_2 supplies are also 1.8V. Extended Spartan-3A family FPGAs do not support 1.8V PROMs because of the Spartan-3A FPGA’s Power-On Reset (POR) voltage threshold, V_{CCO2T} , shown in the appropriate Extended Spartan-3A family data sheet and summarized in [Table 12-1, page 241](#).

Also, see “[Power-On Precautions if 3.3V Supply is Last in Sequence](#),” page 167.

See also “[JTAG Cable Voltage Compatibility](#),” page 198.

Compatible Parallel NOR Flash Families

The Spartan-3E and Extended Spartan-3A family BPI configuration interface operates with a wide variety of x8 or x8/x16 parallel NOR Flash devices. [Table 5-4](#) provides a few example Flash memory families that operate with the BPI interface. Xilinx has hardware

tested various family members from some vendors. Other devices appear to be compatible based on a data sheet analysis. Consult the manufacturer’s data sheet for the desired parallel NOR Flash device to determine the suitability of a specific device.

While most parallel NOR Flash have comparable memory read functions, different vendors may use different programming algorithms, which has no impact on FPGA configuration.

Table 5-4: Example Compatible Parallel NOR Flash Families

Flash Vendor	Flash Memory Family	Status
STMicroelectronics (Numonyx)	M29W	Hardware tested
Atmel	AT29 / AT49	Hardware tested
Spansion	S29	Data sheet compatible
Intel (Numonyx)	Embedded Flash (J3 v. D)	Hardware tested
Macronix	MX29	Data sheet compatible

Required Parallel Flash PROM Densities

Table 5-5 indicates the smallest usable parallel Flash PROM to program a single Extended Spartan-3A family or Spartan-3E FPGA. Parallel Flash memory devices are typically specified by bit density but the memory is addressed as bytes or half-words. Extended Spartan-3A family FPGAs present up to 26 address lines during configuration, although not all are address lines are required, depending on number of bytes required to hold the FPGA bitstream(s). Table 5-5 shows the minimum required number of address lines between the FPGA and parallel Flash PROM. The actual number of address line required depends on the density of the attached parallel Flash PROM.

Table 5-5: Number of Bits to Program an Extended Spartan-3A Family or Spartan-3E FPGA and Smallest Usable Parallel PROM

Family	FPGA	Uncompressed File Sizes (bits)	Smallest Usable Parallel Flash PROM	Minimum Required Address Lines
Spartan-3A/3AN	XC3S50A/AN	437,312	BPI Mode not available on XC3S50A FPGAs	
	XC3S200A/AN	1,196,128	2 Mbit	A[17:0]
	XC3S400A/AN	1,886,560	2 Mbit	A[17:0]
	XC3S700A/AN	2,732,640	4 Mbit	A[18:0]
	XC3S1400A/AN	4,755,296	8 Mbit	A[19:0]
Spartan-3A DSP	XC3SD1800A	8,197,280	8 Mbit	A[19:0]
	XC3SD3400A	11,718,304	16 Mbit	A[20:0]
Spartan-3E	XC3S100E	581,344	1 Mbit	A[16:0]
	XC3S250E	1,353,728	2 Mbit	A[17:0]
	XC3S500E	2,270,208	4 Mbit	A[18:0]
	XC3S1200E	3,841,184	4 Mbit	A[18:0]
	XC3S1600E	5,969,696	8 Mbit	A[19:0]

A multiple-FPGA daisy-chained application requires a parallel Flash PROM large enough to contain the sum of the FPGA file sizes. An application can also use a larger-density parallel Flash PROM to hold additional data beyond just FPGA configuration data. For example, the parallel Flash PROM might also contain the application code for a [MicroBlaze™](#) RISC processor core implemented within the Extended Spartan-3A family or Spartan-3E FPGA. After configuration, the MicroBlaze processor either executes directly from the external Flash memory or it copies the code to other, faster system memory before executing the code.

CCLK Frequency

In BPI mode, the FPGA's internal oscillator generates the configuration clock frequency that controls all the interface timing. The FPGA starts configuration at its lowest frequency and increases its frequency for the remainder of the configuration process if so specified in the configuration bitstream. The maximum frequency is specified using the *ConfigRate* bitstream generator option.

Table 5-6: Maximum ConfigRate Settings for Parallel Flash PROMs (Commercial Temperature Range)

<i>ConfigRate</i> Bitstream Setting	Parallel NOR Flash Read Access Time (T_{ACC} (t_{AVQV}))		Units
	Spartan-3E FPGAs	Spartan-3A/3AN, Spartan-3A DSP FPGAs	
1	≤ 263 ns	≤ 609 ns	ns
3	≤ 120 ns	≤ 189 ns	
6	≤ 49 ns	≤ 85 ns	
7	N/A	≤ 71 ns	
8	N/A	≤ 60 ns	
10	N/A	≤ 43 ns	
12	≤ 14 ns	≤ 33 ns	
13	N/A	≤ 28 ns	
17	N/A	≤ 18 ns	

Notes:

1. PCB signal propagation time assumed to be 1 ns.

Table 5-6 shows the maximum *ConfigRate* settings for various PROM read access times over the Commercial temperature operating range. See “Byte Peripheral Interface (BPI) Timing,” page 169 for more detailed timing information. Extended Spartan-3A family FPGAs have more *ConfigRate* options and therefore offer finer matching to specific memory interface speeds. See Table 5-8, page 160 for *ConfigRate* settings when using parallel Platform Flash PROMs.

Despite using slower *ConfigRate* settings, BPI mode is equally fast as the other configuration modes. In BPI mode, data is accessed at the *ConfigRate* frequency and internally serialized with an 8X clock frequency.

Using the BPI Interface after Configuration

After the FPGA successfully completes configuration, all pins connected to the parallel Flash PROM are available as user I/Os.

If not using the parallel Flash PROM after configuration, drive **LDC0** High to disable the PROM's chip-select input. The remainder of the BPI pins then become available to the FPGA application, including all **A[25:0]** or **A[23:0]** address lines, the eight **D[7:0]** data lines, and the **LDC2**, **LDC1**, and **HDC** control pins.

Because all the interface pins are user I/Os after configuration, the FPGA application can continue to use the interface pins to communicate with the parallel Flash PROM. Parallel Flash PROMs are available in densities ranging from 1 Mbit up to 128 Mbits and beyond. However, a single Spartan-3E/3A/-3AN FPGA requires typically less than 6 Mbits for configuration. If desired, use a larger parallel Flash PROM to contain additional nonvolatile application data, such as MicroBlaze processor code, or other user data, such as serial numbers and Ethernet MAC IDs. In such an example, the FPGA configures from parallel Flash PROM. Then using FPGA logic after configuration, a MicroBlaze processor embedded within the FPGA can either execute code directly from parallel Flash PROM or copy the code to external DDR SDRAM and execute from DDR SDRAM. Similarly, the FPGA application can store nonvolatile application data within the parallel Flash PROM.

For Spartan-3E FPGAs, the configuration data is stored starting at either at location 0 (**BPI Up**) or starting at the highest address location (**BPI Down**) or at both locations for when performing MultiBoot configuration (see "[Spartan-3E MultiBoot](#)," page 263). For Extended Spartan-3A family FPGAs, there is always a configuration image starting at location 0 (**BPI Up**) and possibly at other higher address locations when performing Extended Spartan-3A family MultiBoot configuration (see "[Extended Spartan-3A Family MultiBoot](#)," page 271). Store any additional data beginning in other available parallel Flash PROM sectors.

Caution! Do not mix FPGA configuration data and user data in the same sector. Mixing both configuration and user data in the same sector should only be done with extreme caution.

Similarly, the parallel Flash PROM interface can be expanded to additional parallel peripherals. The address, data, **LDC1** (OE#) and **HDC** (WE#) control signals are common to all parallel peripherals. Connect the chip-select input on each additional peripheral to one of the FPGA user I/O pins. If **HSWAP** or **PUDC_B** = 0 during configuration, the FPGA holds the chip-select line High via an internal pull-up resistor. If **HSWAP** or **PUDC_B** = 1, connect the select line to +3.3V via an external 4.7 kΩ pull-up resistor to avoid spurious read or write operations. After configuration, drive the select line Low to select the desired peripheral. Refer to the individual peripheral data sheet for specific interface and communication protocol requirements.

The FPGA optionally supports a 16-bit peripheral interface by driving the **LDC2** (BYTE#) control pin High after configuration. See "[Precautions Using x8/x16 Flash PROMs](#)" for additional information.

A Spartan-3E FPGA provides up to 24 address lines during configuration, addressing up to 128 Mbits (16 Mbytes). An Extended Spartan-3A family provides up to 26 address lines, addressing up to 512 Mbits (64 Mbytes). If using a larger parallel PROM, connect the upper PROM address lines to FPGA user I/O. During configuration, the upper address lines will be pulled High if **HSWAP** or **PUDC_B** = 0. Otherwise, use external pull-up or pull-down resistors on these address lines to define their values during configuration.

Precautions Using x8/x16 Flash PROMs

Ⓓ Most low- to mid-density PROMs, typically 8 Mbits and below, are only available as byte-wide (x8) memories. Many higher-density Flash PROMs, usually 16 Mbits and above, support both byte-wide (x8) and word-wide (x16) data paths and include a mode input pin called **BYTE#** that switches between the x8 or x16 modes. During configuration, Spartan-3E and Extended Spartan-3A family FPGAs only support byte-wide data, as shown in Figure 5-3a. However, after configuration as shown in Figure 5-3b, the FPGA supports either x8 or x16 modes because the FPGA's **LDC2** pin, which controls the PROM's **BYTE#** mode input, is controlled by the FPGA application. In x16 mode, up to eight additional user I/O pins are required for the upper data bits, D[15:8].

Caution! Different Flash memory vendors use different nomenclature when naming address pins. Make sure that the FPGA connects correctly to the selected memory.

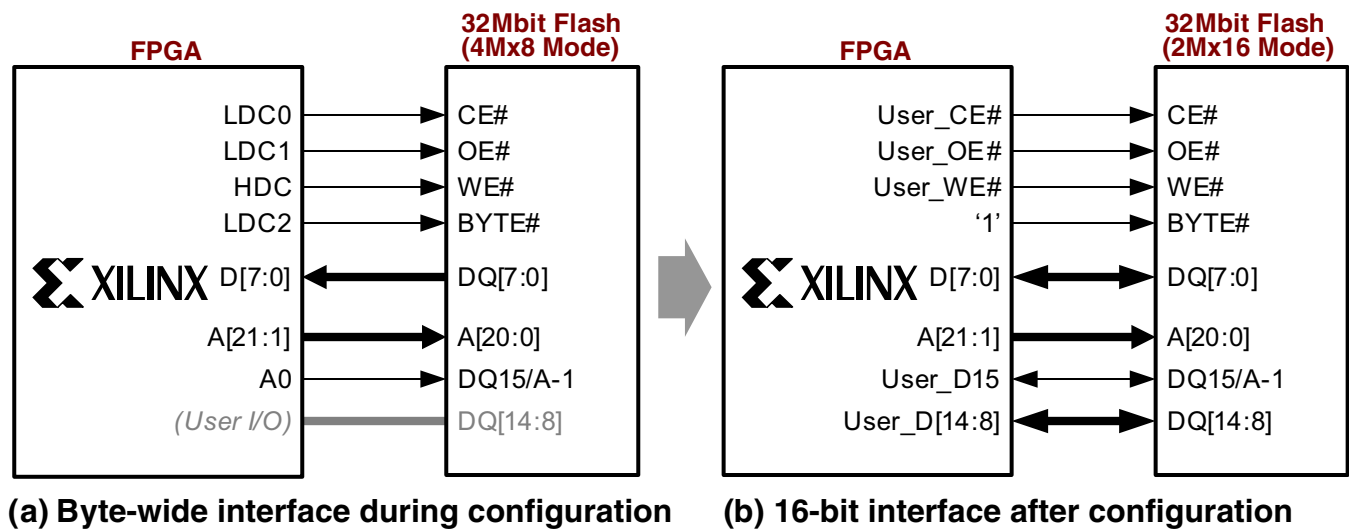


Figure 5-3: FPGA Supports x8 Interface before Configuration and Optional x16 Interface after Configuration

Connecting a Spartan-3E or Extended Spartan-3A family FPGA to a Flash PROM that supports both x8/x16 modes is simple, but does require a precaution. Various Flash PROM vendors use slightly different interfaces to support both x8 and x16 modes. Some vendors (Intel/Numonyx, Micron, some STMicroelectronics/Numonyx devices) use a straightforward interface with pin naming that matches the FPGA connections. However, the PROM's A0 pin is wasted in x16 applications and a separate FPGA user-I/O pin is required for the D15 data line. Fortunately, the FPGA A0 pin is still available as a user I/O after configuration, even though it connects to the Flash PROM.

Other vendors (AMD, Atmel, Silicon Storage Technology, Spansion, and some STMicroelectronics/Numonyx devices) use a pin-efficient interface but change the function of one pin, called **IO15/A-1**, depending if the PROM is in x8 or x16 mode. Figure 5-3 illustrates this interface. In x8 mode, **BYTE#** = 0 controlled by the FPGA's **LDC2** pin, the Flash's **IO15/A-1** pin becomes the least-significant address line into the Flash memory. The **IO15/A-1** line selects a byte location. The A0 address line, which one might assume to be the least-significant address line, is actually the select line for word (x16) locations.

After the FPGA configures successfully, the FPGA application can optionally access the Flash memory using a 16-bit data interface. The FPGA application drives **BYTE#** = 1,

which switches the definition of the **IO15/A-1** pin. This pin then becomes the most-significant data bit, D15 because byte addressing is not required in x16 mode. Check to see if the Flash PROM has a pin named **IO15/A-1** or **DQ15/A-1**. If so, be careful to connect x8/x16 Flash PROMs correctly, as shown in [Figure 5-3](#) and [Table 5-7](#). Also, remember that the D[14:8] data connections require FPGA user I/O pins but that the D15 data is already connected for the FPGA's A0 pin.

Table 5-7: **FPGA Connections to Flash PROM with IO15/A-1 Pin**

FPGA Pin	Connection to Flash PROM with IO15/A-1 Pin	x8 Flash PROM Interface After FPGA Configuration	x16 Flash PROM Interface After FPGA Configuration
LDC2	BYTE#	Drive LDC2 Low or leave unconnected and tie PROM BYTE# input to GND	Drive LDC2 High
LDC1	OE#	Active-Low Flash PROM output-enable control	Active-Low Flash PROM output-enable control
LDC0	CS#	Active-Low Flash PROM chip-select control	Active-Low Flash PROM chip-select control
HDC	WE#	Flash PROM write-enable control	Flash PROM write-enable control
A[23:1]	A[n:0]	A[n:0]	A[n:0]
A0	IO15/A-1	IO15/A-1 is the least-significant address input	IO15/A-1 is the most-significant data line, IO15
D[7:0]	IO[7:0]	IO[7:0]	IO[7:0]
User I/O	Upper data lines IO[14:8] not required unless used as x16 Flash interface after configuration	Upper data lines IO[14:8] not required	IO[14:8]

Some x8/x16 Flash PROMs have a long setup time requirement on the **BYTE#** signal. For the FPGA to configure correctly, the PROM must be in x8 mode with **BYTE# = 0** at power-on or when the FPGA's **PROG_B** pin is pulsed Low. If required, extend the **BYTE#** setup time for a 3.3V PROM using an external 680 Ω pull-down resistor on the FPGA's **LDC2** pin or by delaying assertion of the **CSL_B** select input to the FPGA.

Daisy Chaining

If the application requires multiple FPGAs with different configurations, then configure the FPGAs using a daisy chain, as shown in [Figure 5-4, page 156](#) or [Figure 5-5, page 158](#).

- Parallel daisy chains from a BPI mode master FPGA are supported by both Spartan-3E and Extended Spartan-3A family FPGAs.
- Serial daisy chains from a BPI mode master FPGA are only supported by Extended Spartan-3A family FPGAs.

To successfully configure a daisy chain, the **GTS_cycle** bitstream option must be set to a Startup phase after the **DONE_cycle** setting for all FPGAs in the chain. This is the software default setting. Optionally, set **GTS_cycle:Done**.

Parallel Daisy Chaining

Both Spartan-3E and Extended Spartan-3A family FPGA families support parallel configuration daisy chains when the first device in the chain uses BPI mode.

As shown in Figure 5-4, all downstream FPGAs in the daisy chain use Slave Parallel mode (M[2:0] = <1:1:0>). However, if there are more than two FPGAs in the daisy chain, the last FPGA in the chain can be from any recent Xilinx FPGA family that supports the SelectMAP interface, such as the Virtex®-II, Virtex-II Pro, and Spartan-3 FPGAs. However, all intermediate FPGAs located in the middle of the chain between the first and last FPGAs must from either the Spartan-3E, Extended Spartan-3A family, or Virtex-5 FPGA families. These three FPGA families are the only ones that provide a CSO_B output while in Slave Parallel (SelectMAP) mode.

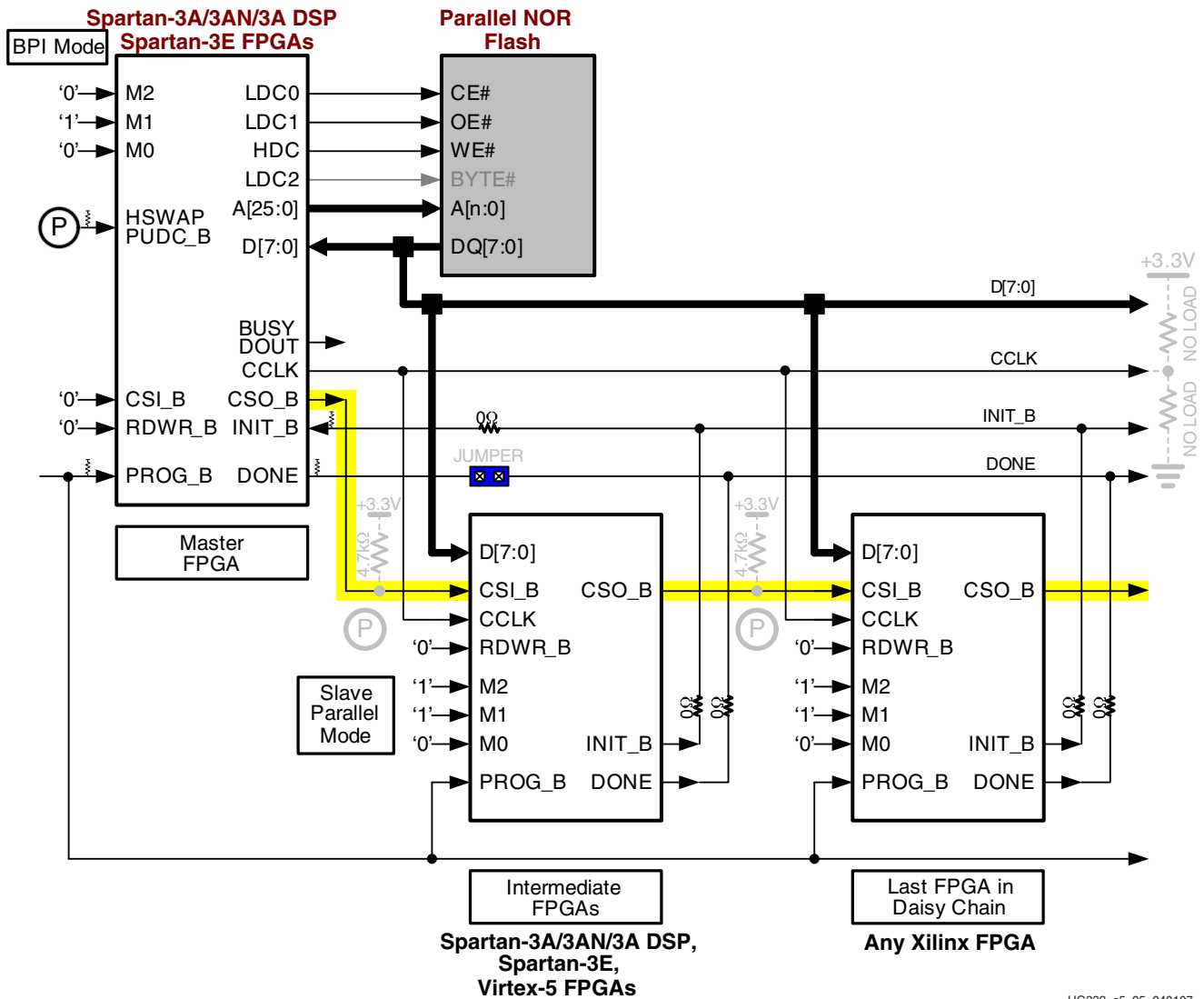


Figure 5-4: Parallel Daisy Chain using BPI Mode

UG332_c5_05_040107

After the master FPGA—the FPGA on the top left in Figure 5-4—finishes loading its configuration data from the parallel Flash PROM, the master device continues generating addresses to the Flash PROM and asserts its CSO_B output Low, enabling the next FPGA

in the daisy chain. The next FPGA then receives parallel configuration data from the Flash PROM. The master FPGA's **CCLK** output synchronizes data capture.

If the FPGA's **HSWAP** or **PUDC_B** pin is High, then pull-up resistors are disabled during configuration and an external 4.7kΩ pull-up resistor must be added on the **CSO_B** pin, which guarantees a logic High to the **CSI_B** input of the next device in the chain. If FPGA's **HSWAP** or **PUDC_B** pin is Low, no external pull-up is necessary.

Serial Daisy Chaining (Extended Spartan-3A Family FPGAs Only)

The Extended Spartan-3A family FPGA family supports serial daisy chains, where the first device in the chain uses BPI mode. The first or master device effectively provides a parallel-to-serial conversion of the bitstream data for the downstream slave devices. Serial daisy chains from BPI mode are not supported for Spartan-3E FPGAs.

As shown in [Figure 5-5, page 158](#), all downstream FPGAs in the serial daisy chain use Slave Serial mode ($M[2:0] = <1:1:1>$) and can be from any Xilinx FPGA family.

The **CCLK** output from the master device operates at 8 times the frequency of the Flash read interface and **CCLK** synchronizes all FPGAs in the daisy chain. The master FPGA accesses the byte-wide Flash once every 8 **CCLK** cycles but provides serial data on its **DOUT** output to downstream FPGAs every **CCLK** cycle. iMPACT programming software automatically adjusts the **CCLK** frequency when serial daisy chains are selected in Step 14, [Figure 5-10, page 164](#). In standalone BPI mode, the **ConfigRate** option determines the byte-wide interface frequency. When a BPI daisy chain is selected, the **ConfigRate** option determines the serial interface frequency, and the parallel Flash interface will run at 1/8 of that rate.

After the master FPGA—the FPGA on the top left in [Figure 5-5](#)—finishes loading its configuration data from the parallel Flash PROM, the master device continues generating addresses to the Flash PROM. The master FPGA reads byte-wide data from the PROM, internally serializes the data, and provides the data to downstream devices via its **DOUT** output pin. The next FPGA in the daisy chain then receives serial configuration data from the preceding FPGA in the chain. The master FPGA's **CCLK** output synchronizes data capture.

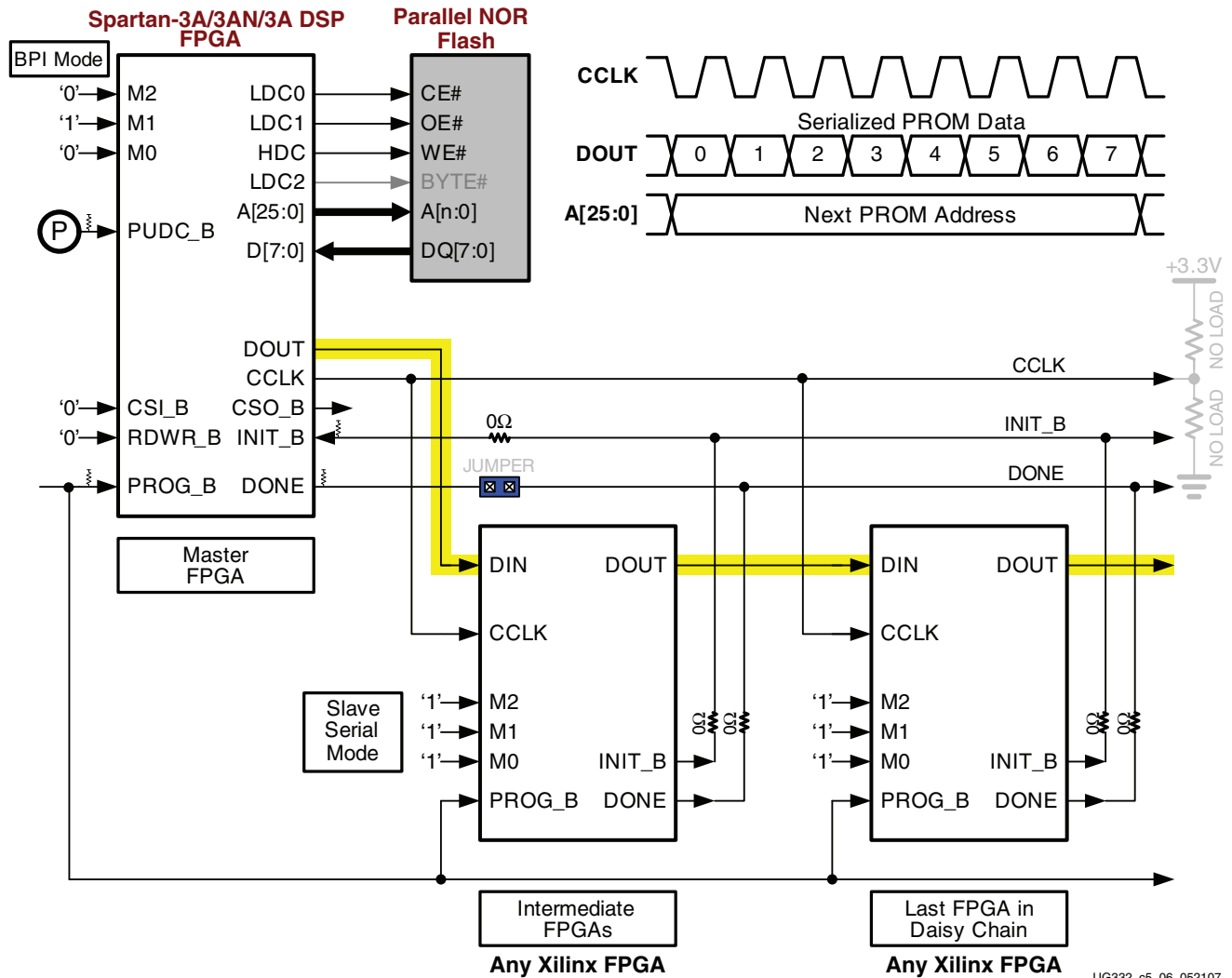


Figure 5-5: Serial Daisy Chains are Only Available for Extended Spartan-3A Family BPI Mode

Using Xilinx Platform Flash PROMs with Master BPI Mode

The Master BPI mode also supports the Xilinx [Parallel Platform Flash PROM](#) (XCFxxP) family, as shown in [Figure 5-6](#).

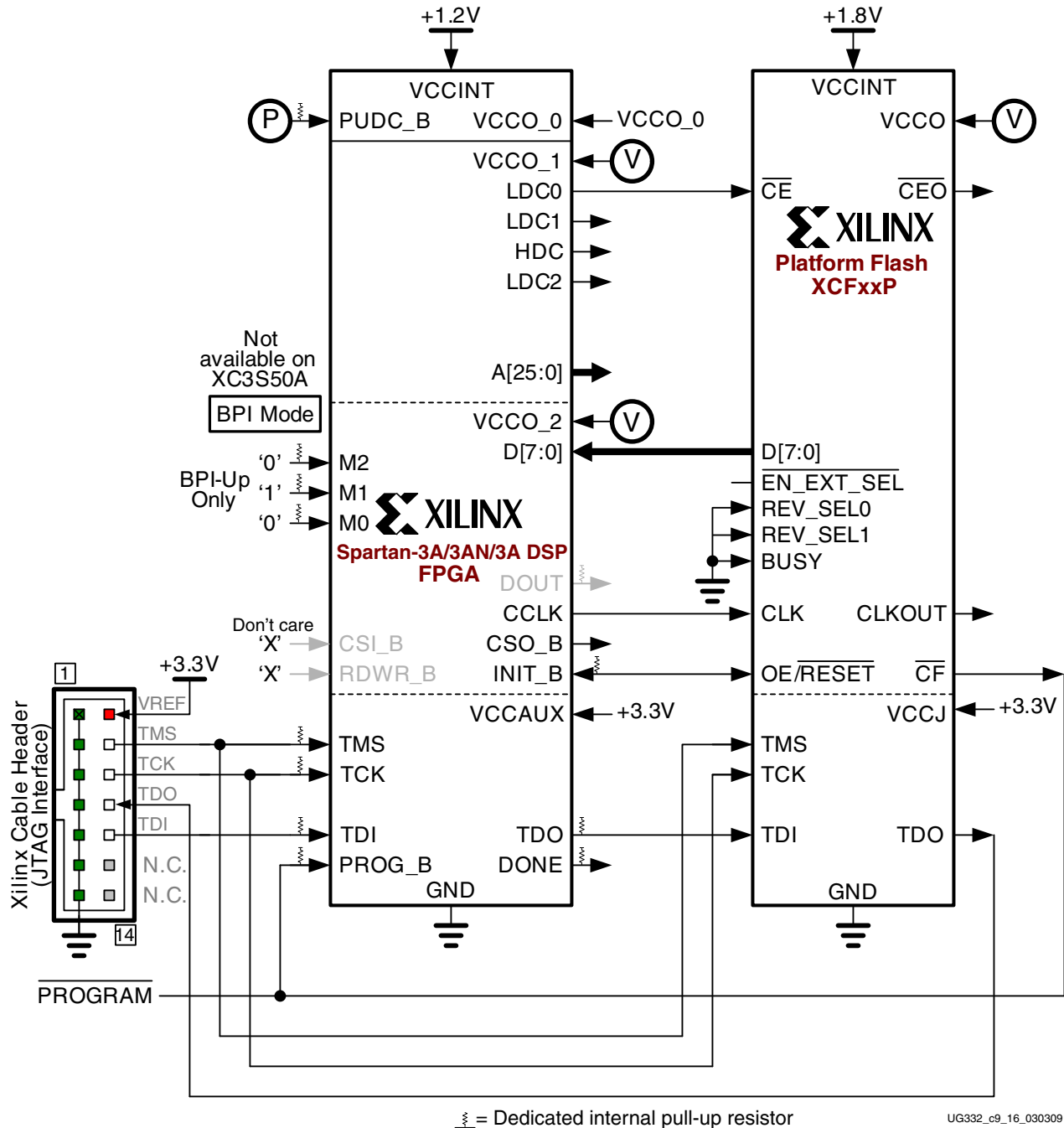


Figure 5-6: Master BPI Mode Using Xilinx Parallel Platform Flash PROMs (XCFxxP)

- The diagram in Figure 5-6 shows an Extended Spartan-3A family FPGA, but the same approach also works with Spartan-3E FPGAs.
- The Xilinx Parallel Platform Flash PROM family is in-system programmable using JTAG, similar to the FPGA.
- See [XAPP483](#), *Multiple-Boot with Platform Flash PROMs*
- The FPGA's LDC2, LDC1, LDC0, and HDC outputs actively drive during configuration. Use the LDC0 output to enable the Platform Flash PROM during

configuration. After configuration, the FPGA application drives LDC0, now an I/O pin to enable or disable the PROM.

- After configuration, the FPGA application can control the I/O pins that connect to the PROM, the application can read additional non-configuration data from the PROM.

A similar approach using Slave Parallel mode is possible, minus the MultiBoot capability. The solution requires either an external configuration clock source or the Platform Flash PROM's internal clock option. The advantage of the alternate solution is that the FPGA's address pins are not active during configuration. Furthermore, if using an external clock source, the clock frequency has little variation and likely operates at a higher average frequency, which shortens configuration time.

ConfigRate Settings Using Platform Flash

As shown in Table 5-8, parallel Platform Flash PROMs support a high *ConfigRate* setting. The performance is even more dramatic considering that the PROM loads eight bits per clock. The resulting bandwidth on an Extended Spartan-3A family FPGAs is between 110 to 190 Mbits per second!

Table 5-8: Maximum *ConfigRate* Settings Using Parallel Platform Flash

Platform Flash Part Number	I/O Voltage (VCCO_2, VCCO)	Spartan-3E <i>ConfigRate</i> Setting	Spartan-3A/3AN Spartan-3A DSP <i>ConfigRate</i> Setting
XCF08P	3.3V or 2.5V	25	33
XCF16P	1.8V		N/A
XCF32P			

Generating the Bitstream for a Master BPI Configuration

To create the FPGA bitstream for a Master BPI configuration, follow the steps outlined in "Setting Bitstream Options, Generating an FPGA Bitstream," page 42. For an FPGA configured in Master BPI mode, set the following bitstream generator options.

ConfigRate: CCLK Frequency

Set the *ConfigRate* option as described in "CCLK Frequency," page 152. Using the ISE® software Project Navigator, the Configuration Rate frequency is set in Step 7 in Figure 1-7, page 44.

```
-g ConfigRate:12
```

StartupClk: CCLK

By default, the configuration Startup clock source is the internally generated CCLK. Keep the *StartupClk* bitstream generation option, shown as Step 13 in Figure 1-8, page 45.

```
-g StartupClk:Cclk
```

DriveDone: Actively Drive DONE Pin

In a single FPGA design or for the Master FPGA in a multi-FPGA daisy chain, set the FPGA to actively drive the DONE pin after successfully completing the configuration process.

Using ISE Project Navigator, check the **Drive Done Pin High** option, shown as Step 16 in Figure 1-8, page 45.

```
-g DriveDone:Yes
```

GTS_cycle: Global Three-State Release Timing for Daisy Chains

If creating a multi-FPGA daisy chain, set the *GTS_cycle* option to be later than the *DONE_cycle* setting, which is the default setting for both. Alternatively, set *GTS_cycle:Done*. From ISE Project Navigator, the *GTS_cycle* setting is the **Enable Outputs (Output Events)** option, shown as Step 14 in Figure 1-8, page 45.

Preparing a Parallel NOR Flash PROM File

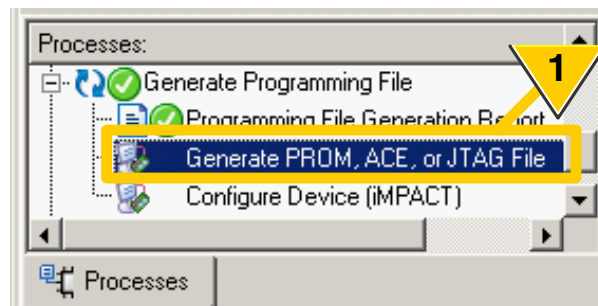
This section provides guidelines to create PROM files for parallel NOR Flash memories.

The Xilinx software tools, “iMPACT” or PROMGen, generate formatted PROM files from the FPGA bitstream or bitstreams.

iMPACT

The following steps graphically describe how to create a PROM file for parallel NOR Flash using iMPACT from within the ISE Project Navigator. iMPACT supports indirect programming for the Spartan-3A and Spartan-3A DSP families. iMPACT supports programming the internal Flash of the Spartan-3AN family. If creating a Spartan-3A/3A DSP MultiBoot image for a parallel Flash memory, see “[Generating an Extended Spartan-3A Family MultiBoot PROM Image using iMPACT](#),” page 278.

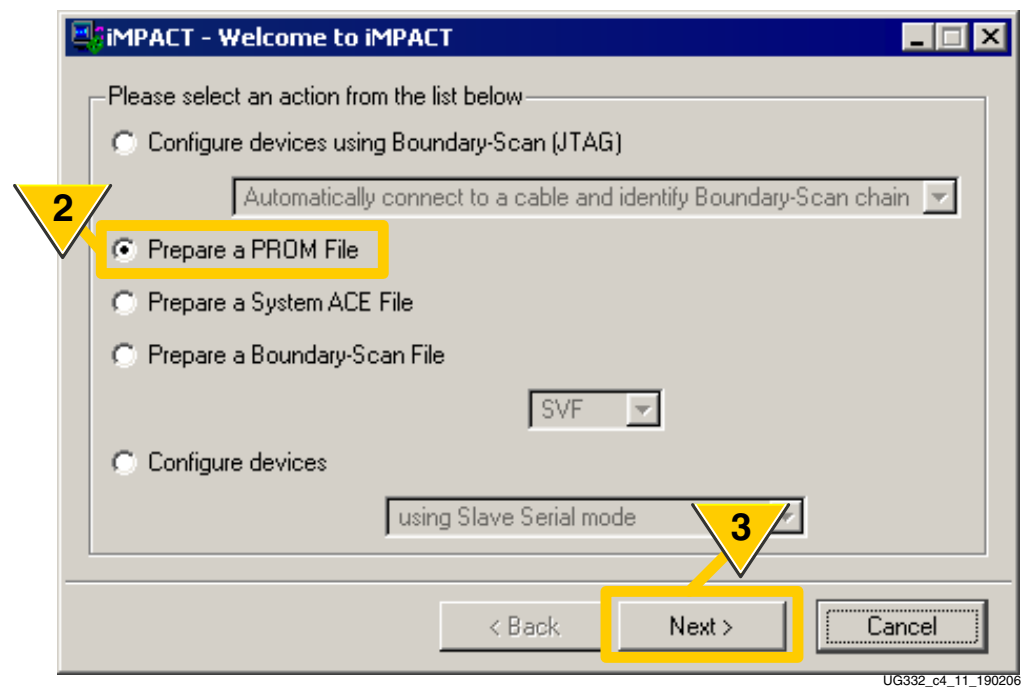
1. From within the ISE Project Navigator, double-click **Generate PROM, ACE, or JTAG File** from within the Process pane, as shown in Figure 5-7.



UG332_c4_10_110206

Figure 5-7: Double-click **Generate PROM, ACE or JTAG File**

- As shown in [Figure 5-8](#), select **Prepare a PROM File**.



UG332_c4_11_190206

Figure 5-8: Prepare a PROM File

- Click Next.

- As shown in [Figure 5-9](#), target a **Generic Parallel PROM**.

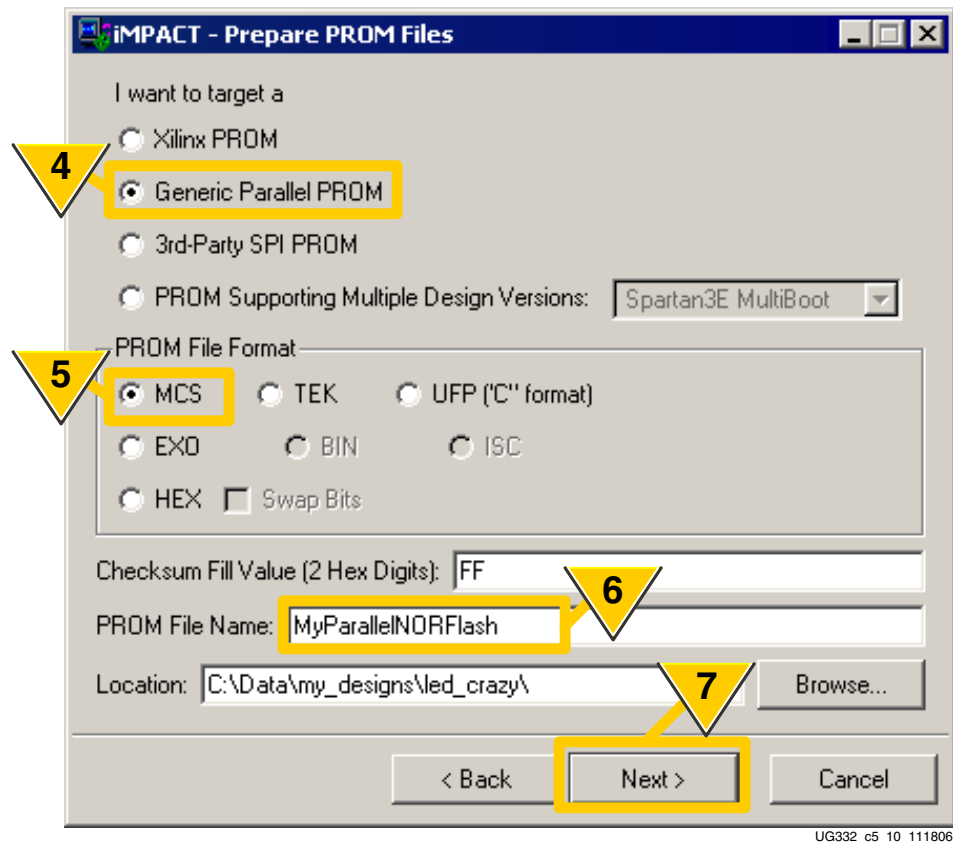


Figure 5-9: Set Options for a Generic Parallel PROM

- Select a **PROM File Format**.
- Name the output **PROM File Name**.
- Click **Next**.

- As shown in [Figure 5-10](#), select the Parallel PROM Density, measured in bytes. This example uses a 32 Mbit Flash PROM, equivalent to 4 Mbytes.

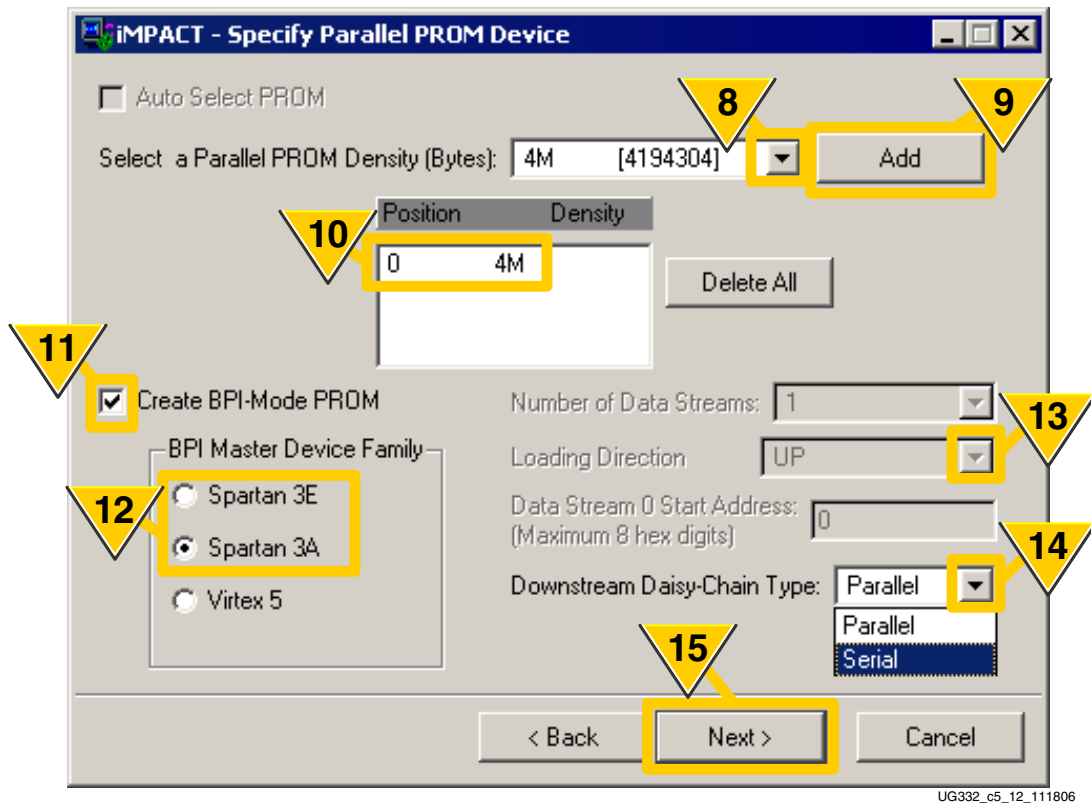
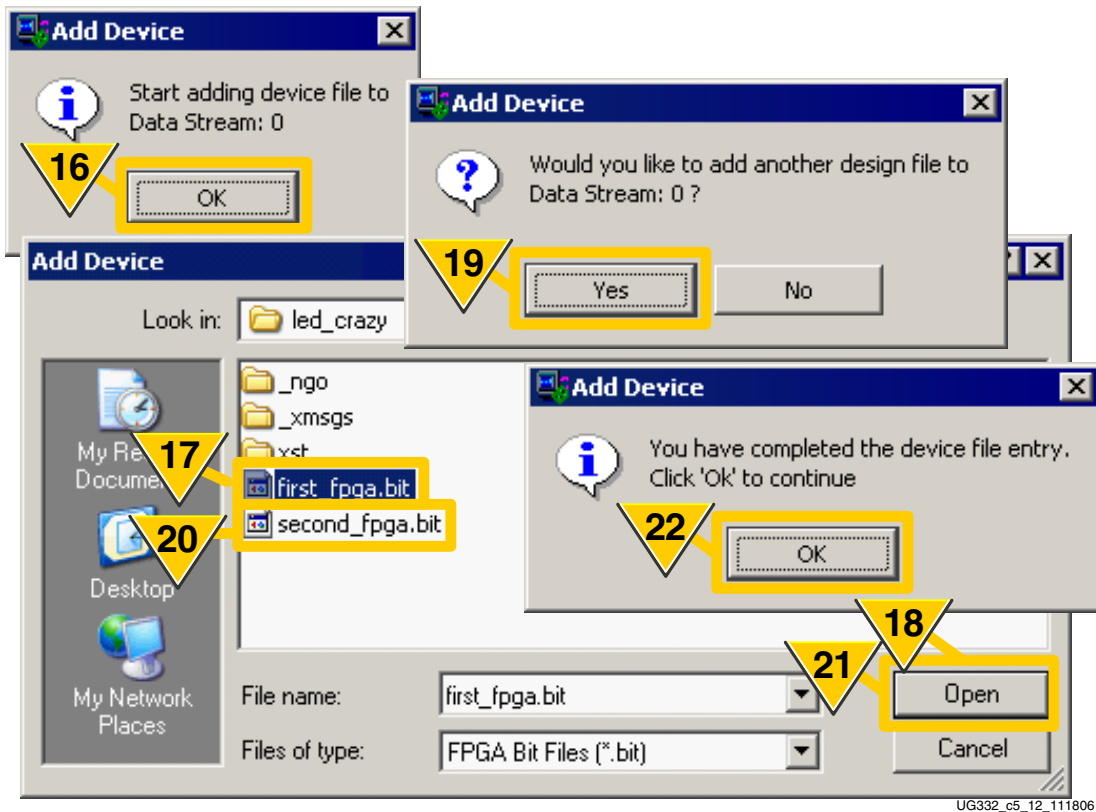


Figure 5-10: Select Parallel PROM Size and Configuration Style

- Click **Add**.
- The selected PROM size appears in the 0 position. The Master BPI mode uses a single PROM.
- Check **Create BPI-mode PROM**.
- Choose whether the **BPI Master Device** is either a **Spartan-3E** or **Spartan-3A** FPGA.
- If the **Spartan-3E** option is selected, then choose whether the PROM file is loaded at address 0 using incrementing addresses (**BPI Up**) or at the highest address location using decrementing addresses (**BPI Down**). This option is not available if the **Spartan-3A** option is the selected BPI Master Device.
- If the **Spartan-3A** option is selected, then choose whether to create a **Parallel** or **Serial** configuration daisy chain. This option is not available if the **Spartan-3E** option is the selected BPI Master Device, although Spartan-3E FPGAs support parallel daisy chains.
- Click **Next**.

16. As shown in Figure 5-11, start selecting the FPGA bitstreams to store in the PROM.



UG332_c5_12_111806

Figure 5-11: Select FPGA Bitstream Files

17. This example create a PROM file for a Spartan-3A serial daisy chain. Select the first FPGA bitstream.
18. Click **Open**.
19. When asked to add another design file, click **Yes**.
20. Select the second FPGA bitstream.
21. Click **Open**. Continue with Steps 19-21 until all FPGA bitstream files are selected. After entering the last bitstream, click **No** from Step 19 when asked to add another design file.
22. Click **OK**.

23. As shown in [Figure 5-12](#), the iMPACT software graphically displays the selected configuration topography. In this example, a single parallel PROM provides the bitstreams to two XC3S700A FPGAs using a serial daisy-chain configuration.

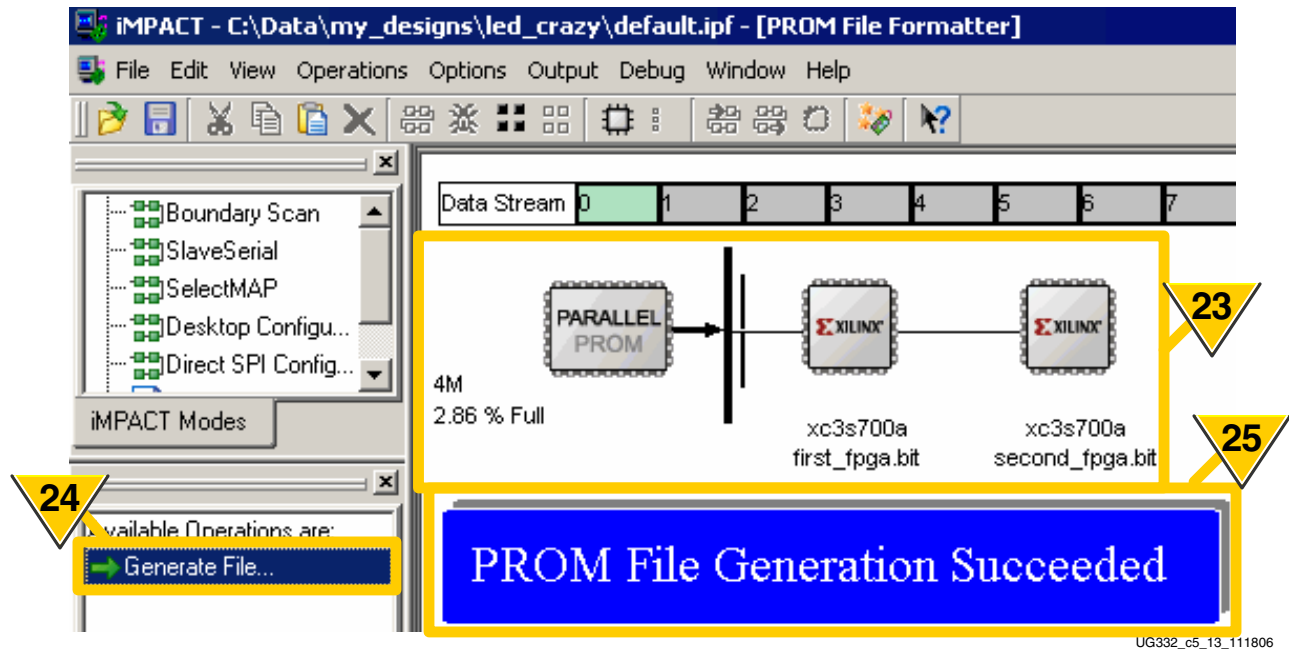


Figure 5-12: Generate Parallel PROM File

24. Click **Generate File**.
25. The iMPACT software indicates when the **PROM File Generation Succeeded**.

Indirect Parallel Flash Programming Using iMPACT

Indirect parallel Flash PROM programming support is available starting with Xilinx ISE 9.2.02i and later releases. In Indirect mode, the iMPACT software programs the memory attached to the FPGA through the FPGA's JTAG port. During the programming process, the FPGA is configured with a special programming application. Consequently, the FPGA's DONE pin will go High during the programming process.

The iMPACT software supports indirect programming of the Intel (Numonyx) [28F P30](#) and [J3 v D](#) StrataFlash embedded flash family through the Spartan-3A (XC3S400A, XC3S700A, and XC3S1400A) and Spartan-3A DSP FPGAs.

In-System Programming Support

In production applications, the parallel Flash PROM is typically preprogrammed before it is mounted on the printed circuit board. In-system programming support is available from third-party boundary-scan tool vendors and from some third-party PROM programmers using a socket adapter with attached wires. To gain direct access to the parallel Flash signals, hold the FPGA's **PROG_B** input Low throughout the programming process. This action places all FPGA I/O pins, including those attached to the parallel Flash, in high-impedance (Hi-Z). If the **HSWAP** or **PUDC_B** input is Low, the I/Os have pull-up resistors to the V_{CCO} input on their respective I/O bank. The external programming hardware then has direct access to the parallel Flash pins.

The FPGA itself can also be used as a parallel Flash PROM programmer during development and test phases. Because parallel NOR Flash is most commonly used with the [MicroBlaze](#) processor core, the Xilinx Platform Studio (XPS) includes Flash programming support. Essentially, XPS downloads a Flash programmer into the FPGA via the FPGA's JTAG port. The FPGA then performs necessary the Flash PROM programming algorithms and receives programming data from the host via the FPGA's JTAG interface.

- Chapter 9, "Flash Memory Programming" in **Embedded System Tools Reference Manual (EDK 10.1)**
http://www.xilinx.com/support/documentation/sw_manuals/edk10_est_rm.pdf

Similarly, the FPGA application can leverage an existing communication channel in the system to program or update the Flash memory. The [Spartan-3E FPGA Starter Kit](#) board provides a design example that programs the on-board Intel (Numonyx) StrataFlash PROM using the board's RS-232 serial port. Similarly, the [Spartan-3A FPGA Starter Kit](#) board provides a similar example, but for the STMicro (Numonyx) M29DW323DT parallel Flash PROM.

- **PicoBlaze™ Processor RS-232 StrataFlash™ Programmer**
www.xilinx.com/products/boards/s3estarter/reference_designs.htm#picoblaze_nor_flash_programmer
- **Programmer for the M29DW323DT Parallel NOR Flash**
www.xilinx.com/products/boards/s3astarter/reference_designs.htm#parallel_flash_programmer

Power-On Precautions if 3.3V Supply is Last in Sequence

Extended Spartan-3A family and Spartan-3E FPGAs have a built-in power-on reset (POR) circuit, as shown in [Figure 12-3, page 240](#). The FPGA waits for its three power supplies — V_{CCINT} , V_{CCAUX} , and V_{CCO} to I/O Bank 2 (V_{CCO_2}) — to reach their respective power-on thresholds before beginning the configuration process.

The parallel NOR Flash PROM is powered by the same voltage supply feeding the FPGA's V_{CCO_2} voltage input, typically 3.3V. Parallel NOR Flash PROMs specify that they cannot be accessed until their V_{CC} supply reaches its minimum data sheet voltage, followed by an additional delay, often called a V_{CC} setup time. [Table 5-9](#) shows some representative values.

Table 5-9: Example Minimum Power-On to Setup Times for Various Parallel NOR Flash PROMs

Vendor	Flash PROM Part Number	Data Sheet Minimum Time from V_{CC} min to Select = Low		
		Symbol	Value	Units
Intel Corp. (Numonyx)	J3 v. D	t_{VCCPH}	60	μ s
Spansion	S29AL016M	t_{VCS}	50	μ s
Macronix	MX29LV004C	t_{VCS}	50	μ s

In many systems, the 3.3V supply feeding the FPGA's V_{CCO_2} input is valid before the FPGA's other V_{CCINT} and V_{CCAUX} supplies, and consequently, there is no issue. However, if the 3.3V supply feeding the FPGA's V_{CCO_2} supply is last in the sequence, a potential race occurs between the FPGA and the NOR Flash PROM, as shown in [Figure 5-13](#).

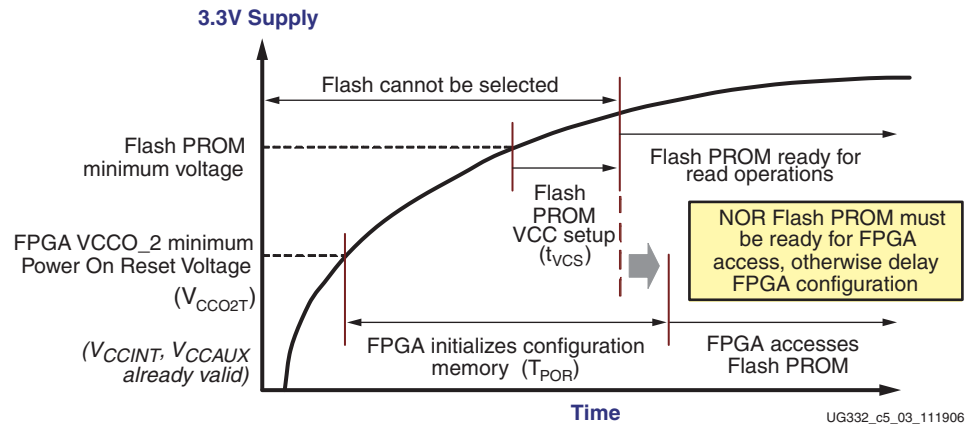


Figure 5-13: Parallel NOR Flash PROM/FPGA Power-On Timing if 3.3V Supply is Last in Power-On Sequence

If the FPGA's V_{CCINT} and V_{CCAUX} supplies are already valid, then the FPGA waits for V_{CCO_2} to reach its minimum threshold voltage before starting configuration. This threshold voltage is labeled as V_{CCO2T} in Table 12-1, page 241 and ranges from approximately 0.4V to 2.0V, substantially lower than the NOR Flash PROM's minimum voltage. Once all three FPGA supplies reach their respective Power On Reset (POR) thresholds, the FPGA starts the configuration process and begins initializing its internal configuration memory. The initialization varies by family and arrays size, listed in Table 12-2, page 242. After initialization, the FPGA deasserts $INIT_B$, selects the NOR Flash PROM, and starts accessing data. The parallel NOR Flash PROM must be ready for read operations at this time.

If the 3.3V supply is last in the sequence and does not ramp fast enough, or if the parallel NOR Flash PROM cannot be ready when required by the FPGA, delay the FPGA configuration process by holding either the FPGA's $PROG_B$ input or $INIT_B$ input Low, described in "Delaying Configuration," page 243. Release the FPGA when the parallel NOR Flash PROM is ready. For example, a simple R-C delay circuit attached to the $INIT_B$ pin forces the FPGA to wait for a preselected amount of time. Alternately, a Power Good signal from the 3.3V supply or a system reset signal accomplishes the same purpose. If using a multi-FPGA daisy-chain configuration, use an open-drain or open-collector output when driving $PROG_B$ or $INIT_B$ as multiple FPGAs are connected to the same node. Similarly, if the Power Good signal is a 3.3V signal, remember that $PROG_B$ is powered by V_{CCAUX} , which must be 2.5V on Spartan-3 and Spartan-3E FPGAs and may be 2.5V or 3.3V on Spartan-3A/3A DSP FPGAs. Add a 68 Ω or larger series resistor if there is a voltage mismatch.

Extended Spartan-3A Family and Configuration Watchdog Timer

Extended Spartan-3A family FPGAs include a configuration watchdog timer (CWDT) which makes parallel Flash configuration more robust, even when the 3.3V supply is applied last.

In Master BPI mode, the CWDT ensures that the FPGA reads a valid synchronization word from the parallel NOR Flash PROM within the first $2^{16}-1$ cycles of $CCLK$. The synchronization word is part of the FPGA configuration bitstream. If the FPGA does not find the synchronization word, the CWDT forces the FPGA to automatically restart the BPI the configuration process. The CWDT retries to successfully configure from parallel NOR

Flash three times before failing. If the FPGA fails to configure, it then drives the `INIT_B` pin Low, indicating a failure.

Byte Peripheral Interface (BPI) Timing

Figure 5-14 provides a detailed timing diagram for the BPI configuration mode. The specific diagram is for the Spartan-3E FPGA family, using the BPI Down mode. However, the timing is also similar for the Extended Spartan-3A family FPGA families and for the BPI Up mode.

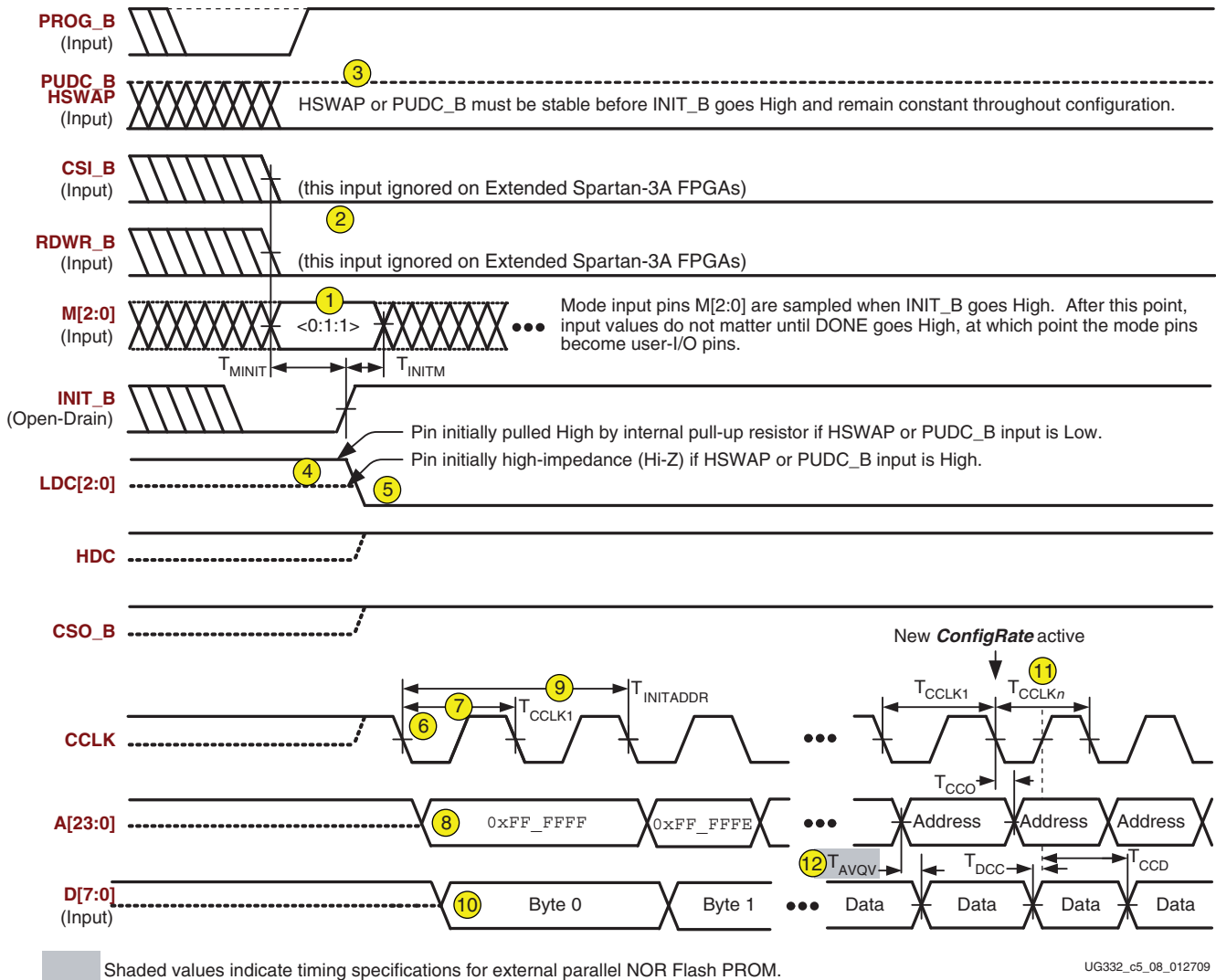


Figure 5-14: BPI Configuration Timing Waveform (Spartan-3E BPI Down mode shown)

The following numbered items correspond to the markers provided in Figure 5-14.

1. The M[2:0] mode pins must be set for BPI mode. Only the Spartan-3E FPGA supports the BPI Down mode. Both Spartan-3E and Extended Spartan-3A family FPGAs support BPI Up mode. See Table 5-2. The mode pin must be setup with sufficient time before the rising edge of `INIT_B`.

2. On Spartan-3E FPGAs, the **CSI_B** select input and the **RDWR_B** read/write control input must be Low before the rising edge of **INIT_B**. It is possible to delay the start of BPI mode configuration by controlling when **CSI_B** is asserted Low. The **CSI_B** and **RDWR_B** pins are not used for Extended Spartan-3A family FPGAs.
3. The **HSWAP** or **PUDC_B** pull-up resistor control input must be setup and valid before the rising edge of **INIT_B**. Similarly, the example in [Figure 5-14](#) shows the pull-up resistors enabled.
4. The **HSWAP** or **PUDC_B** control input defines the initial condition for the FPGA pins that control the Flash, including **LDC2**, **LDC1**, **LDC0**, **HDC**, and **CSO_B**. If **HSWAP** or **PUDC_B** = 1, then these pins are floating (Hi-Z). If **HSWAP** or **PUDC_B** = 0, then these pins have an internal pull-up resistor.
5. After the FPGA completes its internal housecleaning and allows **INIT_B** to go High, the FPGA actively drives the Flash control outputs.
6. The FPGA begins driving the **CCLK** clock output, which controls all the timing for BPI interface.
7. The **CCLK** output begins operating at its lowest frequency option. The ultimate frequency is controlled by a bitstream option called **ConfigRate**.
8. The FPGA-generated address outputs are clocked by the falling edge of **CCLK**.
9. The initial address is held for five **CCLK** cycles in **BPI Up** mode and two **CCLK** cycles in **BPI Down** mode. **BPI Down** mode is only available on Spartan-3E FPGAs.
10. In response to the address inputs provided by the FPGA, the attached PROM asynchronously presents output data.
11. During the first 320 bits in the bitstream, the FPGA loads the **ConfigRate** bitstream setting that potentially increases the **CCLK** output frequency of in order to reduce configuration time.
12. Two directly-related factors control the interface timing. One factor is the PROM data access time, typically called T_{ACC} (t_{AVQV}) or T_{AVQV} in memory data sheets. The other is the maximum **CCLK** frequency, controlled by the **ConfigRate** bitstream generator setting. A faster PROM access time allows a higher **ConfigRate** setting, resulting in a faster **CCLK** frequency and a correspondingly faster configuration time. See [Table 5-6](#), page 152.

[Table 5-10](#) shows the timing requirements of the attached parallel Flash PROM, based on FPGA data sheet timing values.

Table 5-10: Configuration Timing Requirements for Attached Parallel NOR Flash

Symbol	Description	Requirement	Units
T_{CE} (t_{ELQV})	Parallel NOR Flash PROM chip-select time	$T_{CE} \leq T_{INITADDR}$	ns
T_{OE} (t_{GLQV})	Parallel NOR Flash PROM output-enable time	$T_{OE} \leq T_{INITADDR}$	ns
T_{ACC} (t_{AVQV})	Parallel NOR Flash PROM read access time	$T_{ACC} \leq 50\%T_{CCLKn(min)} - T_{CCO} - T_{DCC} - PCB$	ns

Table 5-10: Configuration Timing Requirements for Attached Parallel NOR Flash (Cont'd)

Symbol	Description	Requirement	Units
T_{BYTE} (t_{FLQV} , t_{FHQV})	For x8/x16 PROMs only: BYTE# to output valid time ⁽³⁾	$T_{\text{BYTE}} \leq T_{\text{INITADDR}}$	ns

Notes:

1. These requirements are for successful FPGA configuration in BPI mode, where the FPGA generates the CCLK clock signal. The post-configuration requirements might be different, depending on the application loaded into the FPGA and the resulting clock source.
2. Subtract additional printed circuit board routing delay as required by the application.
3. The initial **BYTE#** timing can be extended using an external, appropriately sized pull-down resistor on the FPGA's **LDC2** pin. The resistor value also depends on whether the FPGA's **HSWAP** or **PUDC_B** pin is High or Low.

Limitations when Reprogramming via JTAG if FPGA Set for BPI Configuration

The FPGA can always be reprogrammed via the JTAG port, regardless of the mode pin (M[2:0]) settings. However, there is a minor limitation if using BPI mode and versions of the ISE software prior to ISE 9.1i, Service Pack 1 (ISE 9.1.01i). The issue with prior software releases exists for all Spartan-3A/3AN FPGA and all Spartan-3E FPGA FPGAs. The issue is resolved using ISE 9.1i, Service Pack 1 or later. The issue does not exist for Spartan-3A DSP FPGAs because support started in later software versions.

Using versions prior to ISE 9.1i, Service Pack 1, if the FPGA is set to configure in BPI mode and the FPGA is attached to a parallel memory containing a valid FPGA configuration file, then subsequent reconfigurations using the JTAG port will fail. Potential workarounds include setting the mode pins for JTAG configuration (M[2:0] = <1:0:1>) or offsetting the bitstream start address in Flash by 0x2000.

Spartan-3E BPI Mode Interaction with Right and Bottom Edge Global Clock Inputs

Some of the Spartan-3E BPI mode configuration pins are shared with global clock inputs along the right and bottom edges of the device (Bank 1 and Bank 2, respectively). These pins are not easily reclaimable for clock inputs after configuration, especially if the FPGA application access the parallel NOR Flash after configuration. Table 5-11 summarizes the shared pins on Spartan-3E FPGAs. These pins are not shared connections on Extended Spartan-3A family FPGAs.

Table 5-11: Spartan-3E: Shared BPI Configuration Pins and Global Buffer Input Pins

Device Edge	Global Buffer Input Pin	BPI Mode Configuration Pin
Bottom	GCLK0	RDWR_B
	GCLK2	D2
	GCLK3	D1
	GCLK12	D7
	GCLK13	D6
	GCLK14	D4
	GCLK15	D3
Right	RHCLK0	A10
	RHCLK1	A9
	RHCLK2	A8
	RHCLK3	A7
	RHCLK4	A6
	RHCLK5	A5
	RHCLK6	A4
	RHCLK7	A3

BPI Data Ordering

On Xilinx FPGAs, data bit D0 is the most-significant bit (MSB) and bit D7 is the least-significant bit (LSB). Consequently, it is crucial to understand how the data ordering in the configuration data file corresponds to the data ordering expected by the FPGA. The Xilinx PROM file generation software provides the option to generate bit-swapped PROM files. The .mcs, .exo, and .tek PROM file formats are byte-swapped unless the BitGen -spi option is used. The .hex file format can be byte-swapped or not byte-swapped, depending on user options. The bitstream files (.bit, .rbit, .bin) are never byte-swapped. Although this convention differs from many other devices, it is consistent across all Xilinx FPGAs. The BPI data ordering is the same as the SelectMAP data ordering. See [“SelectMAP Data Ordering” in Chapter 7](#) for more details.

Master Parallel Mode

Master Parallel Mode is only available on the Spartan®-3 FPGA family. See the [DS099: Spartan-3 FPGA Family Data Sheet](#) for details.

The Extended Spartan-3A family and Spartan-3E FPGA families do not support Master Parallel Mode, but do support a variation described in [Chapter 5, “Master BPI Mode.”](#)

Slave Parallel (SelectMAP) Mode

When using Slave Parallel mode configuration ($M[2:0] = <1:1:0>$), an external host, such as a microprocessor or microcontroller, writes byte-wide configuration data into the FPGA, using a typical peripheral interface. The interface for Spartan®-3E and Extended Spartan-3A family FPGAs appears in [Figure 7-1, page 176](#). The interface for Spartan-3 FPGAs is similar but there are a few minor differences, as shown in [Figure 7-2, page 177](#). The figures show optional components in gray and designated “NO LOAD”. A list of Slave Parallel (SelectMAP) interface pins appears in [Table 7-2, page 179](#).

An overview of Slave Parallel functions is provided in [Table 7-1, page 178](#). The external download host starts the configuration process by pulsing the FPGA’s **PROG_B** pin Low and monitoring that the **INIT_B** pin returns High, indicating that the FPGA is ready to receive its first data. The host asserts the active-Low chip-select signal (**CSI_B** or **CS_B** in the Spartan-3 family) and the active-Low Write signal (**RDWR_B**). The host then continues supplying data and clock signals until either the FPGA’s **DONE** pin goes High, indicating a successful configuration, or until the FPGA’s **INIT_B** pin goes Low, indicating a configuration error.

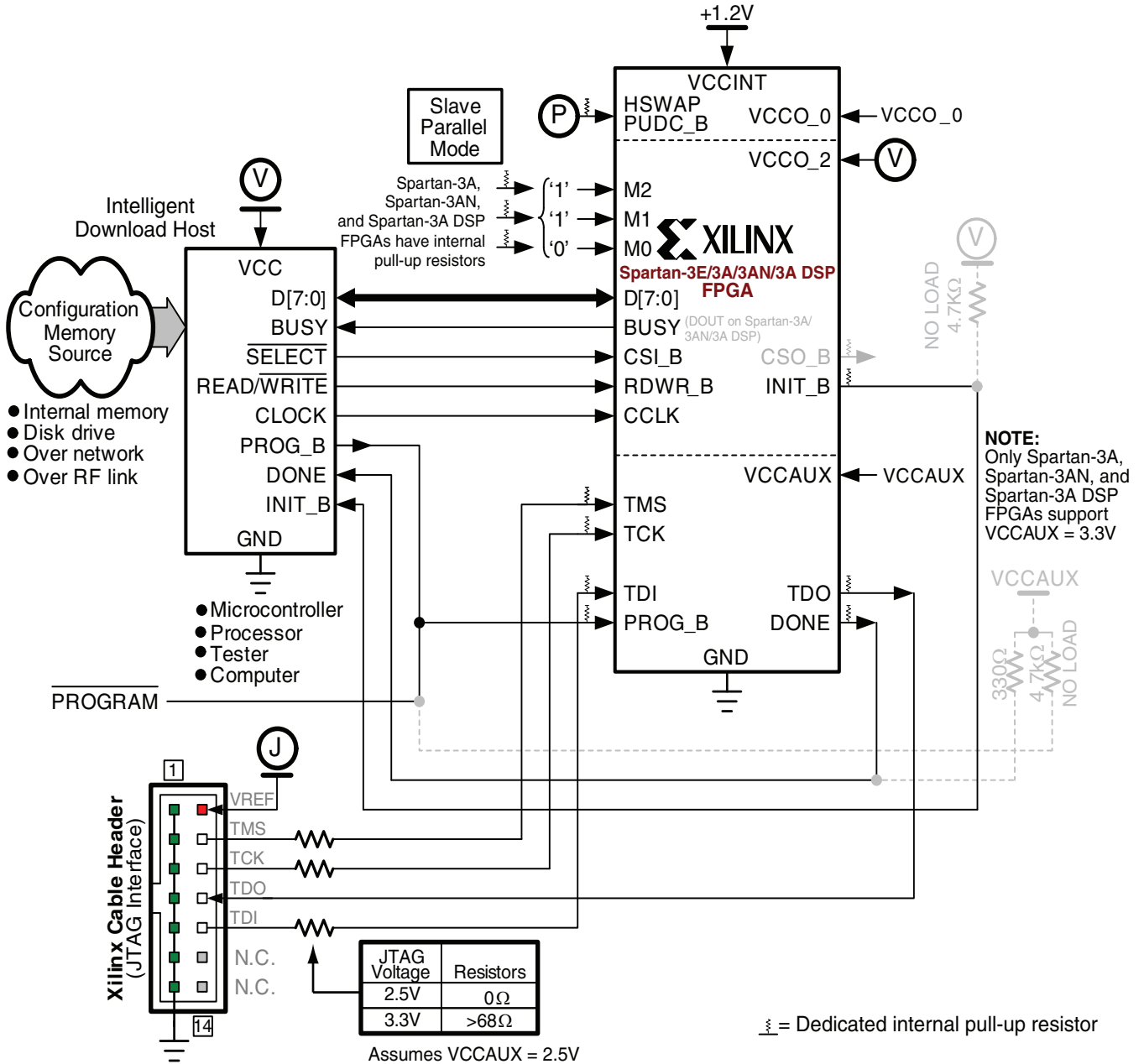
The FPGA captures data on the rising **CCLK** edge. On Spartan-3 and Spartan-3E FPGAs, if the **CCLK** frequency exceeds 50 MHz, then the host must also monitor the FPGA’s **BUSY** output. Extended Spartan-3A family FPGAs do not have a **BUSY** pin. If the FPGA asserts **BUSY** High, the host must hold the data for an additional clock cycle, until **BUSY** returns Low. If the **CCLK** frequency is 50 MHz or below, the **BUSY** pin may be ignored but actively drives during configuration.

The configuration process requires more clock cycles than indicated from the configuration bitstream size alone. Additional clocks are required during the FPGA’s start-up sequence, especially if the FPGA is programmed to wait for selected Digital Clock Managers (DCMs) to lock to their respective clock inputs (**LCK_cycle**). See [“Startup,” page 248](#) for additional information.

If the Slave Parallel interface is only used to configure the FPGA, never to read data back from the FPGA, then the **RDWR_B** signal can also be removed from the interface, but must remain Low during configuration.

After configuration, all of the interface pins except **DONE** and **PROG_B** are available as user I/Os. Alternatively, the bidirectional SelectMAP configuration interface is available after configuration. To continue using SelectMAP mode, set the **Persist:Yes** bitstream generator option. The external host can then read and verify configuration data.

The Slave Parallel mode is also used with BPI mode to create multi-FPGA daisy chains. The lead FPGA is set for BPI mode configuration; all the downstream daisy-chain FPGAs are set for Slave Parallel configuration, as highlighted in [Figure 5-4, page 156](#).



UG332_c7_01_062708

Figure 7-1: Slave Parallel Mode (Spartan-3E and Extended Spartan-3A Family FPGAs)

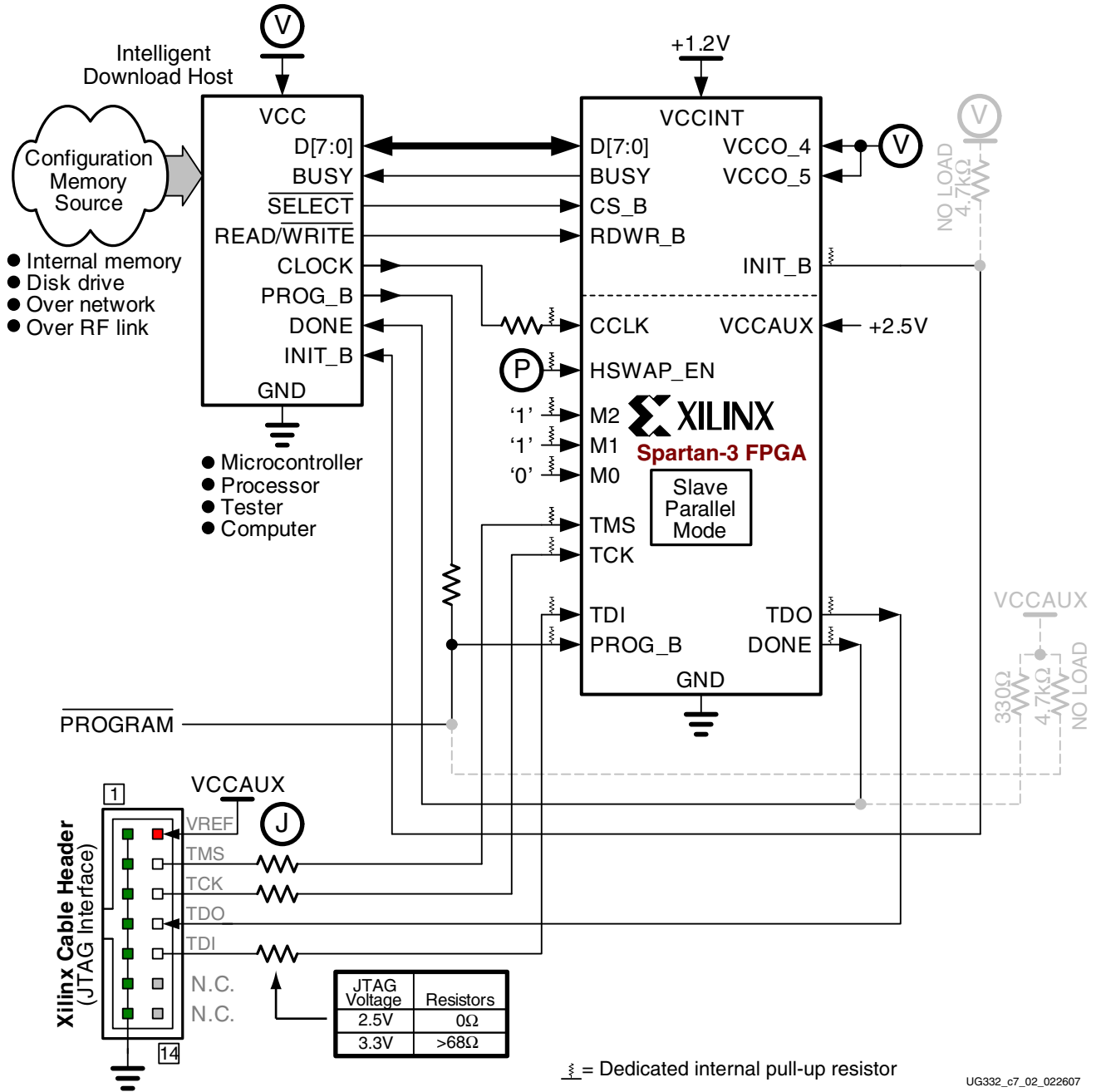


Figure 7-2: Slave Parallel Mode (Spartan-3 FPGAs)

Table 7-1: Slave Parallel (SelectMAP) Function Overview

Inputs to FPGA (D[7:0] is bidirectional)						FPGA Outputs		Function
PROG_B	CSI_B	RDWR_B	D[7:0]	BUSY	CCLK	INIT_B	DONE	
0	X	X	X		X	X	X	Drive PROG_B Low to reset FPGA.
1	X	X	X		X	0	0	FPGA initializing when INIT_B is Low after a PROG_B pulse or initially at power-on.
1	X	X	X		X	1	0	FPGA ready for configuration when INIT_B returns High.
1	1	X	X		X	1	0	No operation when CSI_B is High.
1	0	0	D[7:0] to FPGA	0	↑	1	0	To write configuration data to FPGA, drive RDWR_B Low before or coincident with driving CSI_B Low. Each D[7:0] byte captured on each rising CCLK edge.
1	0	0	D[7:0] to FPGA	1	↑	1	0	BUSY is High, indicating that the FPGA not ready to receive data. Hold current D[7:0] byte until the next CCLK cycle when BUSY returns Low. BUSY not used on Extended Spartan-3A family FPGAs.
1	0	0 to 1	X		X	X	X	ABORT condition if RDWR_B changes state while CSI_B is Low.
1	0	1 to 0s	X		X	X	X	
1	0	1	D[7:0] from FPGA	0	↑	1	1	After configuration, if the Persist:Yes bitstream option is set, the Slave Parallel (SelectMAP) interface can be used to Readback data from the FPGA, assuming the security bits were not set in the FPGA bitstream.
1	X	X	X	0	8x ↑	X	1	FPGA successfully configured eight CCLK cycles after DONE goes High.
1	X	X	X		X	0	0	At the end of configuration, if INIT_B is again Low, then a configuration CRC error occurred.

Notes:

X = don't care

↑ = rising edge

CSI_B is CS_B in the Spartan-3 family

Table 7-2: Slave Parallel Mode Connections

Pin Name	FPGA Direction	Description	During Configuration	After Configuration
Spartan-3E: HSWAP Spartan-3A: Spartan-3AN Spartan-3A DSP: PUDC_B Spartan-3: HSWAP_EN	Input	User I/O Pull-Up Control. When Low during configuration, enables pull-up resistors in all I/O pins to respective I/O bank V _{CCO} input. 0: Pull-ups during configuration 1: No pull-ups	Drive at valid logic level throughout configuration.	Spartan-3: Dedicated pin (don't care after configuration) Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP: User I/O
M[2:0]	Input	Mode Select. Selects the FPGA configuration mode. See "Design Considerations for the HSWAP, M[2:0], and VS[2:0] Pins," page 75.	M2 = 1, M1 = 1, M0 = 0 Sampled when INIT_B goes High.	User I/O
D[7:0]	Input	Data Input.	Byte-wide data provided by host. FPGA captures data on rising CCLK edge. D0 is the MSB (see "SelectMAP Data Ordering")	User I/O. If bitstream option Persist:Yes , becomes part of SelectMap parallel peripheral interface.
Spartan-3: Spartan-3E: BUSY	Output	Busy Indicator. Not required for Extended Spartan-3A family FPGAs but function exists on DOUT pin. Do not connect DOUT to active logic during configuration.	If CCLK frequency is less than 50 MHz, this pin may safely be ignored. When High, indicates that the FPGA is not ready to receive additional configuration data. Host must hold data an additional clock cycle.	User I/O. If bitstream option Persist:Yes , becomes part of SelectMap parallel peripheral interface.
Spartan-3E: Spartan-3A Spartan-3AN Spartan-3A DSP: CSL_B Spartan-3: CS_B	Input	Chip Select Input. Active Low.	Must be Low during valid data cycles.	User I/O. If bitstream option Persist:Yes , becomes part of SelectMap parallel peripheral interface.
RDWR_B	Input	Read/Write Control. Active Low write enable.	Must be Low throughout configuration. Do not change the state of RDWR_B while CSL_B or CS_B is asserted; otherwise an ABORT is issued.	User I/O. If bitstream option Persist:Yes , becomes part of SelectMap parallel peripheral interface.
CCLK	Input	Configuration Clock. If CCLK PCB trace is long or has multiple connections, terminate this output to maintain signal integrity. See "CCLK Design Considerations," page 58.	External clock.	User I/O. If bitstream option Persist:Yes , becomes part of SelectMap parallel peripheral interface.

Table 7-2: Slave Parallel Mode Connections (Cont'd)

Pin Name	FPGA Direction	Description	During Configuration	After Configuration
Spartan-3E: Spartan-3A Spartan-3AN Spartan-3A DSP: CSO_B	Output	Chip Select Output. Active Low. Not provided on Spartan-3 FPGAs.	Not used in single-FPGA designs; CSO_B is pulled up, not actively driving. In a Spartan-3E or Extended Spartan-3A family parallel daisy-chain configuration, this pin connects to CSI_B or CS_B input of the next FPGA in the chain.	User I/O
INIT_B	Open-drain bidirectional I/O	Initialization Indicator. Active Low. Goes Low at the start of configuration during the Initialization memory clearing process. Released at the end of memory clearing, when mode select pins are sampled.	Active during configuration. If CRC error detected during configuration, FPGA drives INIT_B Low.	User I/O. If unused in the application, drive INIT_B High to avoid a floating value. See INIT_B "After Configuration".
DONE	Open-drain bidirectional I/O	FPGA Configuration Done. Low during configuration. Goes High when FPGA successfully completes configuration.	Low indicates that the FPGA is not yet configured.	When High, indicates that the FPGA successfully configured.
PROG_B	Input	Program FPGA. Active Low. When asserted Low for 500 ns or longer, forces the FPGA to restart its configuration process by clearing configuration memory and resetting the DONE and INIT_B pins once PROG_B returns High.	Must be High to allow configuration to start.	Drive PROG_B Low and release to reprogram FPGA.

Voltage Compatibility

Ⓟ Most Slave Parallel interface signals are within the FPGA's I/O Bank 2, supplied by the VCCO_2 supply input. The VCCO_2 voltage can be 1.8V, 2.5V, or 3.3V to match the requirements of the external host, ideally 2.5V. Using 1.8V or 3.3V requires additional design considerations as the DONE and PROG_B pins are powered by the FPGA's 2.5V VCCAUX supply. Extended Spartan-3A family FPGAs do not support 1.8V PROMs because of the Spartan-3A FPGA's Power-On Reset (POR) voltage threshold, V_{CCO2T} , shown in the appropriate Extended Spartan-3A family data sheet and summarized in [Table 12-1, page 241](#).

See [XAPP453: The 3.3V Configuration of Spartan-3 FPGAs](#) for additional information.

Also see "JTAG Cable Voltage Compatibility," [page 198](#).

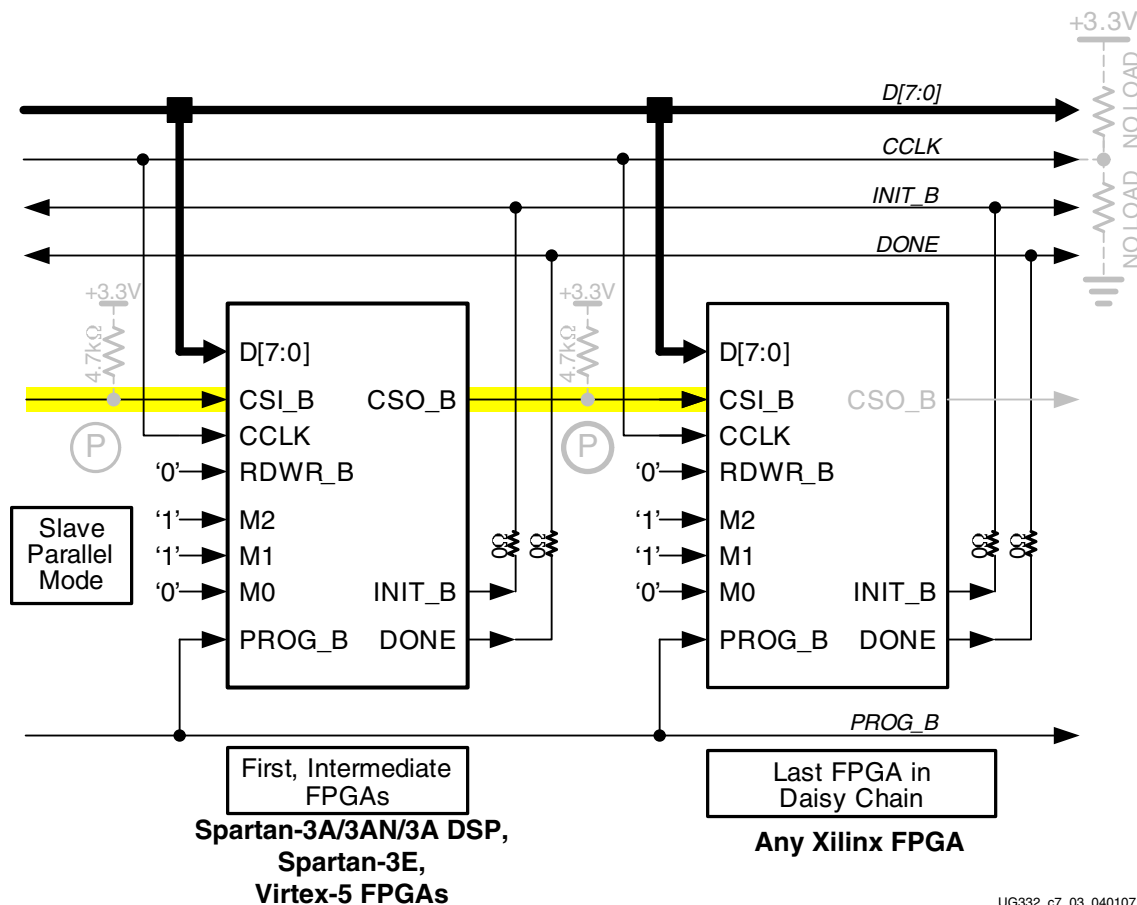
Daisy Chaining

If the application requires multiple FPGAs with different configurations, then configure the FPGAs using a daisy chain. Use Slave Parallel mode ($M[2:0] = <1:1:0>$) for all FPGAs in the daisy chain. There are two possible topologies available, one that supports only Spartan-3E and Extended Spartan-3A family FPGAs and another that works with any modern Xilinx® FPGA, Virtex® or Spartan-II FPGA and later.

Spartan-3E/Extended Spartan-3A Family Slave Parallel Daisy Chains

Figure 7-3, page 181 shows a daisy-chain topology that primarily supports Spartan-3E and Extended Spartan-3A family FPGAs, although the last FPGA in the chain can be from any modern Xilinx FPGA family. It essentially leverages the BPI mode daisy-chain technique. The upstream FPGA in the chain drives its CSO_B Low, enabling the downstream FPGA's CSI_B or CS_B input. Only Spartan-3E, Spartan-3A, Spartan-3AN, Spartan-3A DSP, and Virtex-5 FPGAs have a CSO_B output. Consequently, one of these FPGAs must be the first and intermediate FPGAs in the daisy chain.

Pull-up resistors on the CSO_B to CSI_B connect are required if the FPGAs HSWAP, PUDC_B, HSWAP_EN or input is High, meaning that the FPGA's internal pull-up resistors are disabled during configuration.



UG332_c7_03_040107

Figure 7-3: Slave Parallel Daisy Chain for Spartan-3E/Extended Spartan-3A Family FPGAs

Slave Parallel Daisy Chains Using Any Modern Xilinx FPGA Family

Figure 7-4, page 182 describes an alternate Slave Parallel daisy-chain scheme that supports any modern Xilinx FPGA family, including all Spartan-3 generation FPGAs. The topology is similar to that shown in Figure 7-3, page 181 except that each FPGA has a separate **CSI_B** or **CS_B** chip-select input.

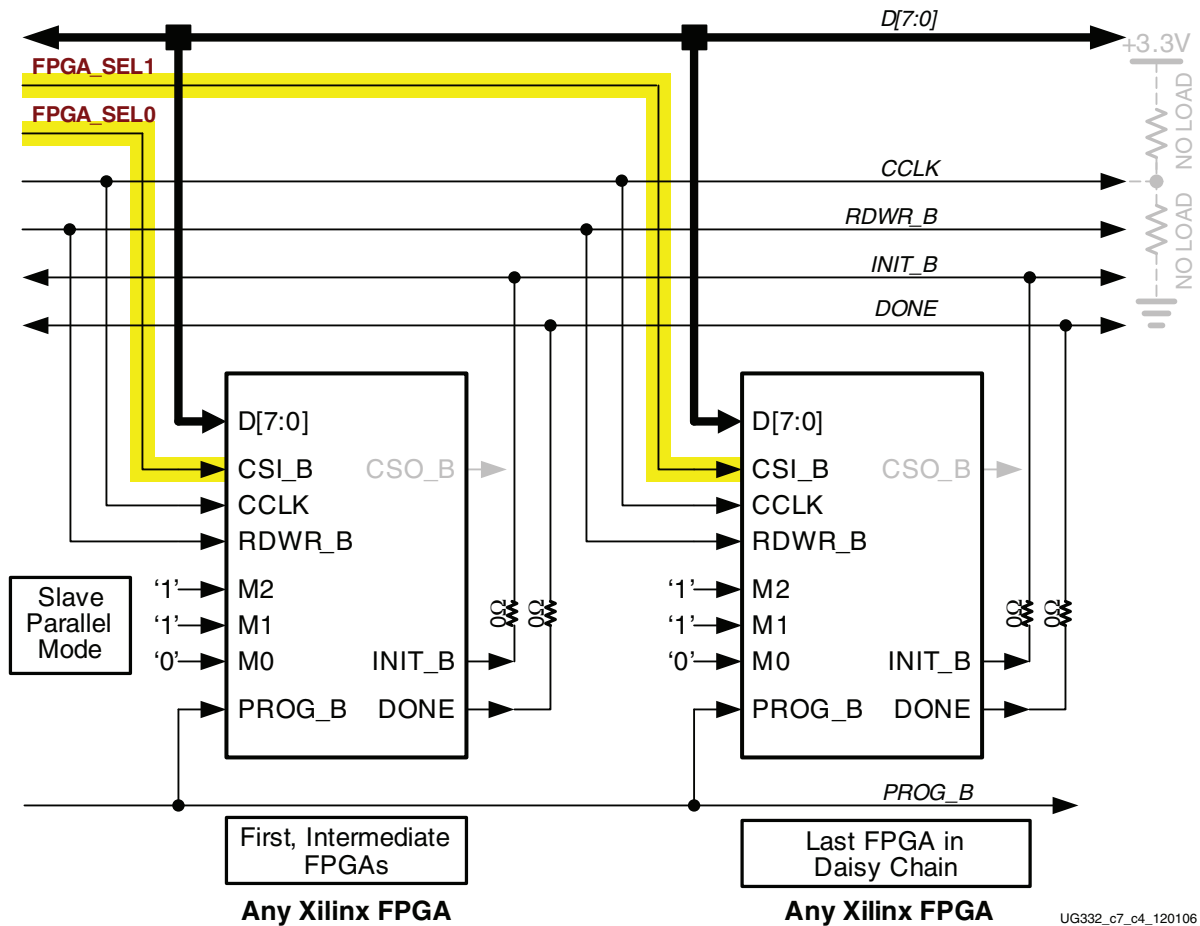


Figure 7-4: Slave Parallel Daisy Chain Using Any Modern Xilinx FPGA

SelectMAP Data Loading

The SelectMAP interface provides for either continuous or non-continuous data loading. Data loading is controlled by the FPGA's **CSI_B**, **RDWR_B**, **CCLK**, and **BUSY** signals. Extended Spartan-3A family FPGAs do not have a **BUSY** signal.

CSI_B

The active-Low chip-select input (**CSI_B**) enables the SelectMAP interface. When **CSI_B** is High, the FPGA ignores the SelectMAP interface. The data port and **BUSY** output pin are high-impedance (Hi-Z). **CSI_B** in the Spartan-3E and Extended Spartan-3A families is equivalent to **CS_B** in the Spartan-3 family.

If only one device is being configured through the SelectMAP and readback is not required, or if ganged SelectMAP configuration is used, connect the **CSI_B** signal to GND.

RDWR_B

The **RDWR_B** input controls whether the SelectMAP data pins are inputs or outputs.

- When **RDWR_B** = 0, the **D[7:0]** data pins are inputs (writing to the FPGA).
- When **RDWR_B** = 1, the **D[7:0]** data pins are outputs (reading from the FPGA).

When writing configuration data to the FPGA, the **RDWR_B** pin must be Low. When reading back configuration information from the FPGA, the **RDWR_B** pin must be High, while **CSI_B** is deasserted.

Changing the value of **RDWR_B** while **CSI_B** is Low triggers an ABORT if the FPGA receives a rising edge on **CCLK** (see “SelectMAP ABORT,” page 186). If Readback is not used, **RDWR_B** can be tied to ground or used for debugging with SelectMAP ABORT.

The **RDWR_B** signal is ignored while **CSI_B** is High. Read/write control (three-state control) of the **D[7:0]** data pins is asynchronous. The FPGA actively drives SelectMAP data.

CCLK

All activity on the SelectMAP data bus is synchronous to **CCLK**. When writing configuration data to the FPGA, **RDWR_B** is Low and the FPGA samples the data on rising **CCLK** edges. When **RDWR_B** is set for read control (**RDWR_B** = 1, Readback), the FPGA updates the SelectMAP data pins on rising **CCLK** edges.

Configuration can be paused by pausing **CCLK** as outlined in “Non-Continuous SelectMAP Data Loading,” page 185.

BUSY

If the system writes data to or reads data from the FPGA at less than 50 MHz, then the **BUSY** pin can be left unconnected. Extended Spartan-3A family FPGAs do not require a **BUSY** pin but have the same functionality on the DOUT pin.

BUSY is an output indicating when the device is ready to receive configuration data or drive Readback data.

- When **BUSY** = 0, the FPGA is ready to receive or send data, depending on the operation.
- When **BUSY** = 1, the FPGA is not ready to receive or send data. If writing to the FPGA, hold the current data value until **BUSY** returns Low.

When **CSI_B** is deasserted (**CSI_B** = 1), the **BUSY** pin is in a high-impedance (Hi-Z) state.

BUSY remains in a Hi-Z state until **CSI_B** is asserted. If **CSI_B** is asserted before power-up — for example, if the pin is tied to GND — **BUSY** initially is in a Hi-Z state, then drives Low after the Power-On Reset is released.

Continuous SelectMAP Data Loading

Continuous data loading occurs when the external processor or controller provides an uninterrupted stream of configuration data to the FPGA. After power-up, the controller asserts **RDWR_B** = 0 to write data to the FPGA and asserts **CSI_B** = 0 to select the FPGA. This action causes the FPGA to drive **BUSY** Low, which is an asynchronous transition. Drive the FPGA's **RDWR_B** pin Low before or coincident with asserting **CSI_B** Low, otherwise an ABORT occurs, described in “SelectMAP ABORT,” page 186.

On the next rising **CCLK** edge, the FPGA begins sampling the **D[7:0]** data pins. Actual FPGA configuration begins after the FPGA recognizes the synchronization word, as described in “Synchronization,” page 244.

After the configuration bitstream is loaded, the device enters the Startup sequence. The FPGA asserts its **DONE** signal High in the Startup phase specified by the **DONE_cycle** bitstream option. See “Startup,” page 248. The processor or controller must continue sending **CCLK** pulses until after the Startup sequence successfully completes, which requires several **CCLK** pulses after **DONE** goes High.

After configuration, the **CSI_B** and **RDWR_B** signals can be deasserted, or they can remain asserted. Because the SelectMAP port is inactive, toggling **RDWR_B** at this time does not cause an ABORT event. Figure 7-5 summarizes the timing of SelectMAP configuration with continuous data loading.

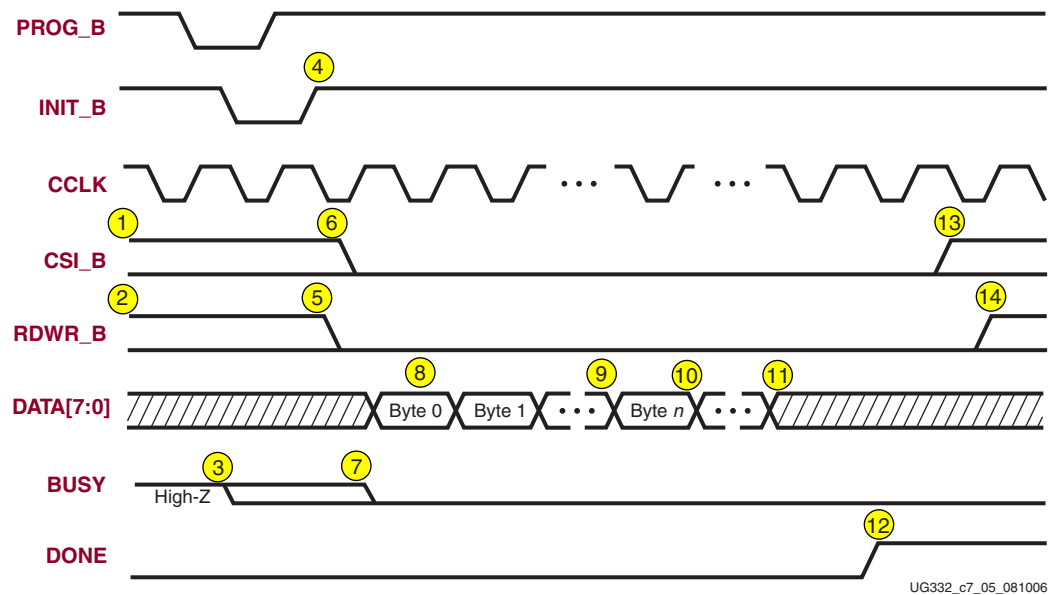


Figure 7-5: SelectMAP Continuous Data Loading

The following numbered items correspond to the markers provided in Figure 7-5.

1. **CSI_B** signal can be tied Low if there is only one device on the SelectMAP bus. If **CSI_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 **CSI_B** has been asserted because this triggers an ABORT. See “SelectMAP ABORT,” page 186.
3. If **CSI_B** is tied Low, **BUSY** drives Low before **INIT_B** returns High.
4. The FPGA samples the **M[2:0]** mode-select pins when **INIT_B** goes High.
5. Assert **RDWR_B** before **CSI_B** to avoid causing an abort.
6. **CSI_B** is asserted, enabling the SelectMAP interface.
7. **BUSY** (Spartan-3/3E only) remains in High-Z state until **CSI_B** is asserted.
8. The first **D[7:0]** byte is loaded on the first rising **CCLK** edge after **CSI_B** is asserted.
9. The configuration bitstream is loaded one byte per rising **CCLK** edge.
10. After the last byte is loaded, the FPGA 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 **CCLK** cycles can be required to complete the startup sequence. See “Startup,” page 248.
13. After configuration has finished, the **CSI_B** signal can be deasserted.
14. After the **CSI_B** signal is deasserted, **RDWR_B** can be deasserted.

Non-Continuous SelectMAP Data Loading

Non-continuous data loading is used in applications where the processor or controller cannot provide an uninterrupted stream of configuration data. This may occur, for example, if the controller pauses configuration while it fetches additional data, switches to another task, or services an interrupt.

There are two methods to throttle or pause the configuration data throughput in the Spartan-3 and Spartan-3E FPGAs. Only the second method is supported in the Spartan-3A, Spartan-3AN, and Spartan-3A DSP FPGAs.

1. Deassert the **CSI_B** signal with a free-running **CCLK**, shown in Figure 7-6 and described in “Deasserting **CSI_B**,” page 185.
2. Pause **CCLK**, shown in Figure 7-7 and described in “Pausing **CCLK**,” page 186.

Deasserting **CSI_B**

Note: This method is only supported in the Spartan-3 and Spartan-3E FPGAs. It is not supported in the Extended Spartan-3A FPGAs, which should instead use the “Pausing **CCLK**” method. **CSI_B** is labeled **CS_B** in the Spartan-3 family.

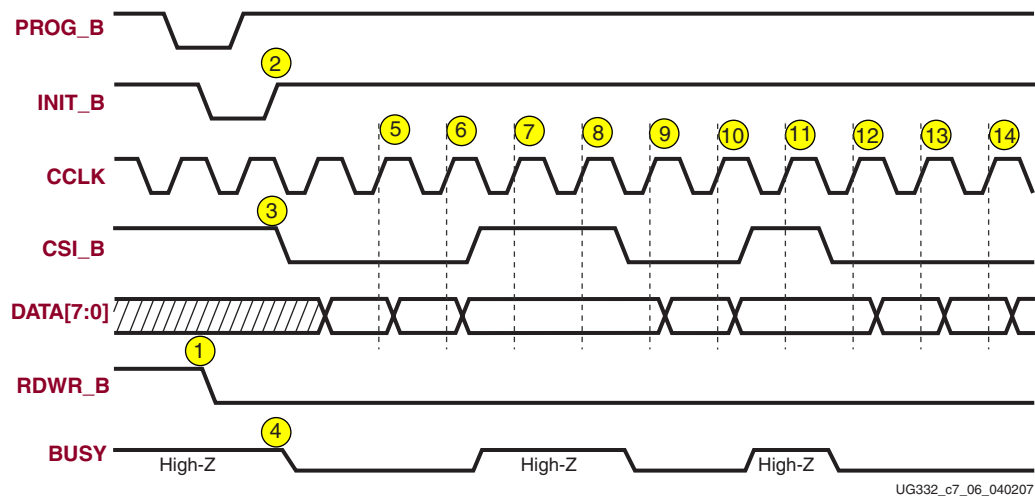


Figure 7-6: SelectMAP Non-Continuous Data Loading with Controlled **CSI_B**

The following numbered items correspond to the markers provided in Figure 7-6.

1. The external processor drives **RDWR_B** Low, setting the FPGA’s **D[7:0]** pins as inputs for configuration. The **RDWR_B** input can be tied Low if Readback is not used in the application. **RDWR_B** should not be toggled after **CSI_B** has been asserted because this triggers an ABORT, described in “SelectMAP ABORT,” page 186.
2. The FPGA is ready for configuration after **INIT_B** returns High.

3. The processor asserts **CSI_B** Low, enabling the SelectMAP interface. The **CSI_B** input can be tied Low if there is only one device on the SelectMAP bus. If **CSI_B** is not tied Low, it can be asserted at any time.
4. **BUSY** goes Low shortly after **CSI_B** is asserted. If **CSI_B** is tied Low, **BUSY** is driven Low before **INIT_B** returns High.
5. A **D[7:0]** data byte is loaded on the rising **CCLK** edge.
6. A **D[7:0]** data byte is loaded on the rising **CCLK** edge.
7. The processor deasserts **CSI_B**, and the data on **D[7:0]** is ignored.
8. The processor deasserts **CSI_B**, and the data on **D[7:0]** is ignored.
9. A **D[7:0]** data byte is loaded on the rising **CCLK** edge.
10. A **D[7:0]** data byte is loaded on the rising **CCLK** edge.
11. The processor deasserts **CSI_B**, and the data on **D[7:0]** is ignored
12. A **D[7:0]** data byte is loaded on the rising **CCLK** edge.
13. A **D[7:0]** data byte is loaded on the rising **CCLK** edge.
14. A **D[7:0]** data byte is loaded on the rising **CCLK** edge.

Pausing CCLK

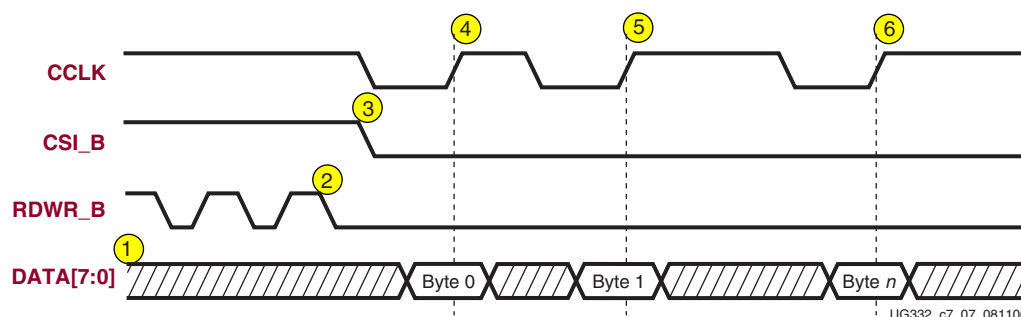


Figure 7-7: Non-Continuous SelectMAP Data Loading with Controlled CCLK

The following numbered items correspond to the markers provided in Figure 7-7.

1. The **D[7:0]** data pins are high-impedance (Hi-Z) while **CSI_B** is deasserted.
2. **RDWR_B** has no effect on the device while **CSI_B** is deasserted.
3. **CSI_B** is asserted by the processor. The FPGA captures configuration data on rising **CCLK** edges.
4. A **D[7:0]** data byte is loaded on the rising **CCLK** edge.
5. A **D[7:0]** data byte is loaded on the rising **CCLK** edge.
6. A **D[7:0]** data byte is loaded on the rising **CCLK** edge.

SelectMAP ABORT

An ABORT is an interruption in the SelectMAP configuration process or in the Readback sequence that occurs if the **RDWR_B** pin changes state while **CSI_B** is asserted Low. During a configuration ABORT, the FPGA drives internal status information onto the **D[7:4]** pins over the next four **CCLK** cycles. The other data pins, **D[3:0]** remain High. After the ABORT sequence finishes, the processor that is downloading the FPGA must

resynchronize the configuration logic before resuming configuration. For applications that must deassert `RDWR_B` between bytes use the method described in “Pausing CCLK,” page 186.

Configuration Abort Sequence Description

An ABORT is signaled during configuration as shown in Figure 7-8.

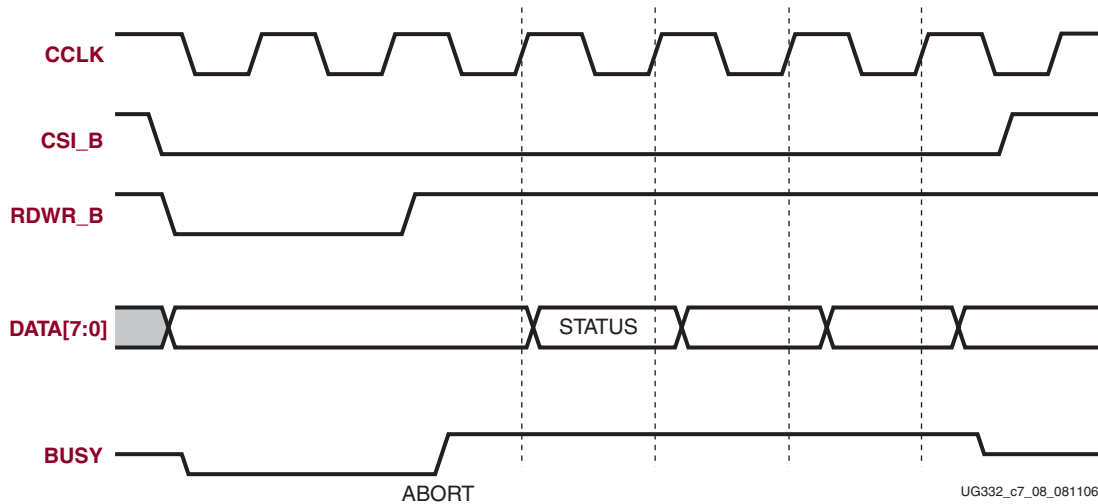


Figure 7-8: Configuration Abort Sequence

1. The configuration sequence begins normally.
2. The processor changes the value on the `RDWR_B` pin while the FPGA is still selected; `CSI_B` is Low.
3. `BUSY` goes High if `CSI_B` remains asserted Low. The FPGA drives the status word onto the data pins if `RDWR_B` is High, reading data from the FPGA. The Status value is not presented by the FPGA if `RDWR_B` is Low.
4. The ABORT lasts for four clock cycles, and Status is updated.

Readback Abort Sequence Description

An ABORT is signaled during readback as shown in Figure 7-9.

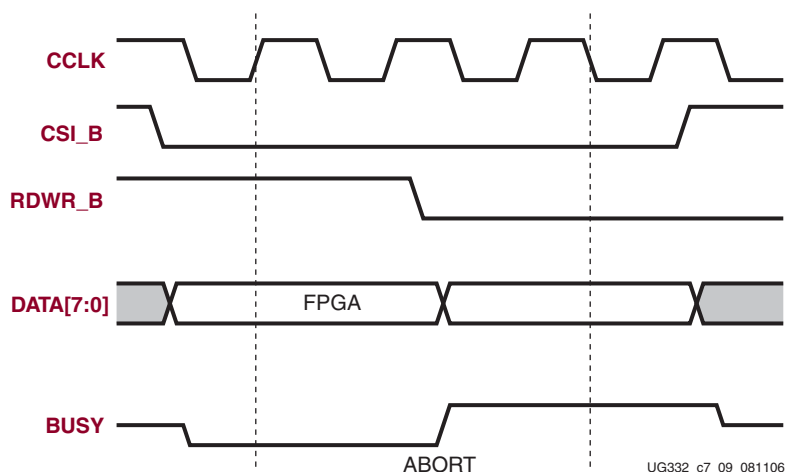


Figure 7-9: Readback Abort Sequence

1. The readback sequence begins normally.
2. The processor changes the `RDWR_B` pin while the FPGA is still selected; `CSI_B` is Low.
3. `BUSY` (Spartan-3/3E only) goes High if `CSI_B` remains asserted Low. The FPGA drives the status word onto the data pins if `RDWR_B` is High, reading data from the FPGA. The Status value is not presented by the FPGA if `RDWR_B` is Low.

ABORT operations during Readback typically are not followed by a status word because the `RDWR_B` signal will be Low, causing the ABORT. When `RDWR_B` is Low, the processor is writing to the FPGA and the FPGA's `D[7:0]` pins are inputs. The FPGA cannot present the Status value.

ABORT Status Word

During the configuration ABORT sequence, the FPGA presents a status word onto the `D[7:4]` pins. The other data pins, `D[3:0]`, are all High. The key for that status word is given in [Table 7-3](#).

Table 7-3: 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	Synchronization Word Received 0 = No synchronization word received. 1 = Synchronization word received.
D5	RIP	Readback In Progress 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.
D[3:0]	N/A	1111 (all High)

The ABORT sequence lasts four CCLK cycles. During those cycles, the status word changes to reflect data alignment and ABORT status. An example ABORT sequence appears in [Table 7-4](#).

Table 7-4: Example ABORT Sequence

D[7:0] from FPGA	D7 CFGERR_B	D6 DALIGN	D5 RIP	D4 IN_ABORT_B	D[3:0] N/A
11011111	1	1	0	1	1111
11001111	1	1	0	0	1111
10001111	1	0	0	0	1111
10011111	1	0	0	1	1111

After the last cycle, the synchronization word can be reloaded to establish data alignment.

Resuming Configuration or Readback After an Abort

There are two ways to resume configuration or readback after an ABORT:

1. The FPGA can be resynchronized after the ABORT completes by resending the configuration synchronization word. See [Table 12-3, page 244](#).
2. Reset the FPGA by pulsing **PROG_B** Low at any time.

To resynchronize the device, **CSI_B** must first be deasserted then reasserted. To resume configuration or readback, resend the last configuration or readback packet that was in progress when the ABORT occurred. Alternatively, restart configuration or readback from the beginning.

Persist

Generally, the FPGA's dual-purpose configuration pins become user-I/O pins after configuration. The SelectMAP configuration port can be maintained after configuration by setting the bitstream generation option **Persist:Yes** or by selecting "Allow SelectMAP Pins to Persist" in the Project Navigator. Allowing the configuration port to persist enables readback or reconfiguration through the external configuration pins.

The pins that retain their configuration function when **Persist:Yes** is selected appear in [Table 7-5](#). These pins become disconnected from the user design when Persist is used and therefore cannot be used by the design.

Table 7-5: Pins Affected by Persist

Pin Name	FPGA Families	Description
M[2:0]	Spartan-3, Spartan-3E, Spartan-3A, Spartan-3AN, Spartan-3A DSP	Mode Select
CCLK	Spartan-3E, Spartan-3A, Spartan-3AN, Spartan-3A DSP	Configuration Clock (Dedicated in Spartan-3)
INIT_B	Spartan-3, Spartan-3E, Spartan-3A, Spartan-3AN, Spartan-3A DSP	Initialization
CSI_B	Spartan-3E, Spartan-3A, Spartan-3AN, Spartan-3A DSP	Chip Select, Active-Low
CS_B	Spartan-3	
RDWR_B	Spartan-3, Spartan-3E, Spartan-3A, Spartan-3AN, Spartan-3A DSP	Read/Write
BUSY	Spartan-3, Spartan-3E	FPGA Busy Indicator
D[7:0]	Spartan-3, Spartan-3E, Spartan-3A, Spartan-3AN, Spartan-3A DSP	Data
A[23:20]	Spartan-3E	Highest-order Address Lines

When **Persist:Yes** is selected, the post-configuration CRC checker in the Spartan-3A, Spartan-3AN, and Spartan-3A DSP FPGAs is clocked by CCLK.

SelectMAP Reconfiguration

The term *reconfiguration* refers to reprogramming an FPGA after its **DONE** pin has gone High, which is distinctly different than programming the FPGA immediately after power is applied. To reconfigure the FPGA, pulse the **PROG_B** pin Low, which is identical to configuration, or reconfigure by resynchronizing the FPGA and sending configuration data.

Generally, the FPGA's SelectMAP pins become user-I/O pins after configuration, because the **Persist:No** bitstream option is set by default. To reconfigure a device in SelectMAP mode without pulsing **PROG_B**, set the bitstream option **Persist:Yes**, which reserves the Slave Parallel (SelectMAP) interface pins after configuration, preventing them from becoming user-I/O pins.

Reconfigure the FPGA by clocking the appropriate synchronization word, shown in [Table 12-3, page 244](#), into the SelectMAP port. The remainder of the operation is identical

to configuration as described above. The devices support full reconfiguration through the SelectMAP port.

SelectMAP Data Ordering

On Spartan-3 generation FPGAs, by Xilinx convention, data bit D0 is the most-significant bit (MSB) and bit D7 is the least-significant bit (LSB). However, this convention varies between vendors and can be especially confusing when the FPGA uses one convention and the attached processor downloading configuration data to the FPGA uses the opposite convention! Consequently, it is crucial to understand how the data ordering in the configuration data file corresponds to the data ordering expected by the FPGA.

In SelectMAP, the byte-wide configuration data is loaded one byte per CCLK, with the most-significant bit of each byte presented to the FPGA's D0 data pin. The same data ordering applies to the BPI mode.

Table 7-6 provides an example of how the FPGA would like to see the hexadecimal value 0xABCD presented on the SelectMAP data bus. Note how the bits within each byte need to be reversed.

Table 7-6: Bit Ordering for SelectMAP 8-Bit Mode

CCLK Cycle	Hex Equivalent	D7	D6	D5	D4	D3	D2	D1	D0
1	0xAB	1	1	0	1	0	1	0	1
2	0xCD	1	0	1	1	0	0	1	1

Notes:

1. D[0:7] represent the SelectMAP DATA pins.

Some applications can accommodate the non-conventional data ordering without much difficulty. For other applications, it may be more convenient to store the source configuration data file with the data bits already bit-swapped, meaning that the bits in each byte of the data stream are reversed. The Xilinx PROM file generation software provides the option to generate bit-swapped PROM files.

Byte Swapping

The .mcs, .exo, and .tek PROM file formats are byte-swapped unless the `-spi` option is used. The .hex file format can be byte-swapped or not byte-swapped, depending on user options. The bitstream files (.bit, .rbit, .bin) are never byte-swapped.

The .hex file format contains only configuration data. The other PROM file formats include address and checksum information that should not be sent to the FPGA. The address and checksum information is used by some third-party device programmers, but is not programmed into the PROM.

Figure 7-10 shows how two bytes of data (0xABCD) are byte-swapped.

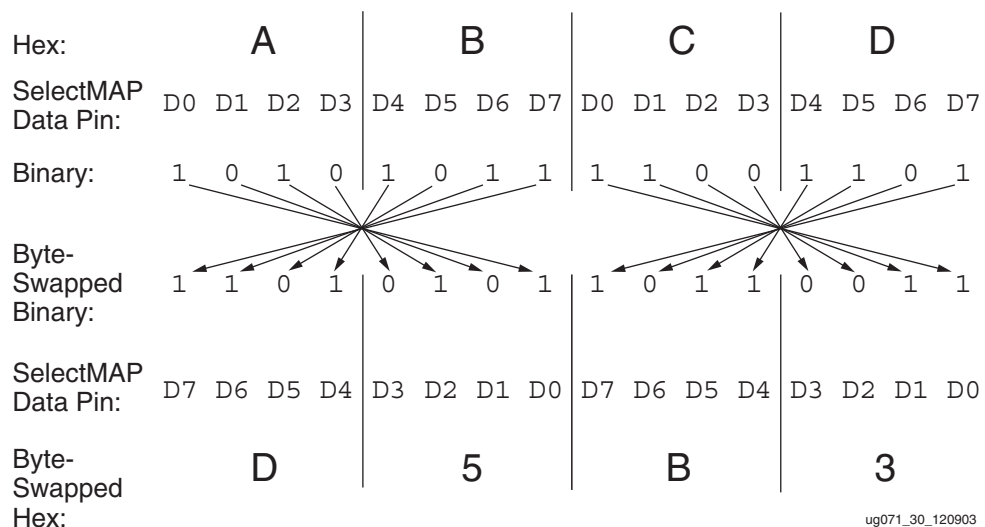


Figure 7-10: Byte Swapping Example

The MSB of each byte goes to the D0 pin regardless of the orientation of the data:

- In the byte-swapped version of the data, the bit that goes to D0 is the rightmost bit
- In the non-byte-swapped data, the bit that goes to D0 is the leftmost bit.

Whether or not data must be byte-swapped is entirely application-dependent, and is only applicable for SelectMAP configuration applications. Non-byte-swapped data should be used for SPI and Slave serial downloads.

Slave Serial Mode

In Slave Serial mode ($M[2:0] = \langle 1:1:1 \rangle$), an external host such as a microprocessor or microcontroller writes serial configuration data into the FPGA, using the synchronous serial interface shown in [Figure 8-1](#). The figure shows optional components in gray. The serial configuration data is presented on the FPGA's DIN input pin with sufficient setup time before each rising edge of the externally generated CCLK clock input.

The intelligent host starts the configuration process by pulsing PROG_B and monitoring that the INIT_B pin goes High, indicating that the FPGA is ready to receive its first data. The host then continues supplying data and clock signals until either the DONE pin goes High, indicating a successful configuration, or until the INIT_B pin goes Low, indicating a configuration error. The configuration process requires more clock cycles than indicated from the configuration file size. Additional clocks are required during the FPGA's start-up sequence, especially if the FPGA is programmed to wait for selected Digital Clock Managers (DCMs) to lock to their respective clock inputs (see ["Startup,"](#) page 248).

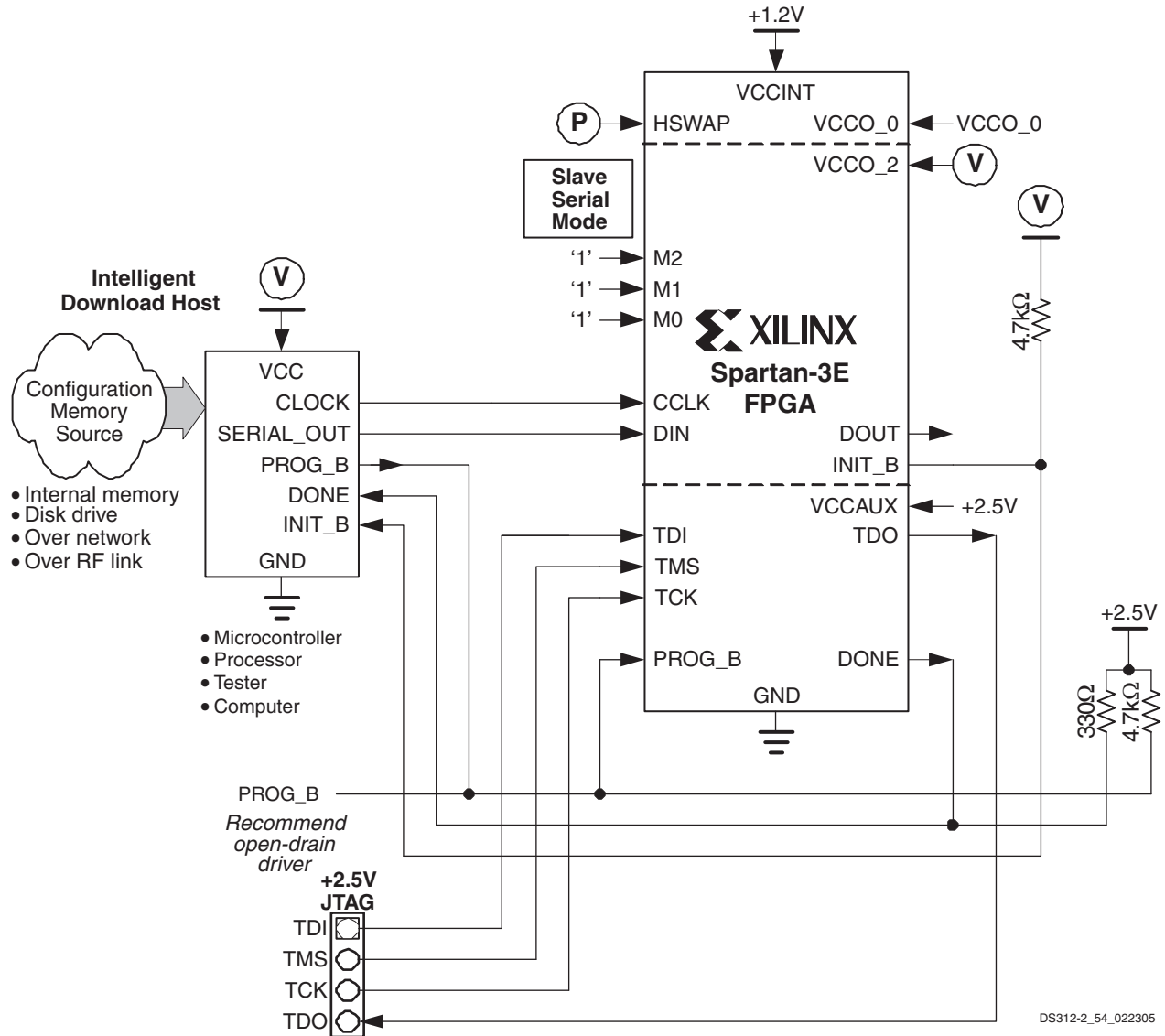


Figure 8-1: Slave Serial Configuration

The mode select pins, M[2:0], are sampled when the FPGA's INIT_B output goes High and must be at defined logic levels during this time. After configuration, when the FPGA's DONE output goes High, the mode pins are available as full-featured user-I/O pins.

(P) Similarly, the FPGA's HSWAP (PUDC_B) pin must be Low to enable pull-up resistors on all user-I/O pins or High to disable the pull-up resistors. The HSWAP (PUDC_B) control must remain at a constant logic level throughout FPGA configuration. After configuration, when the FPGA's DONE output goes High, the HSWAP (PUDC_B) pin is available as full-featured user-I/O pin and is powered by the VCCO_0 supply.

Voltage Compatibility

Ⓥ Most Slave Serial interface signals are within the FPGA's I/O Bank 2, supplied by the VCCO_2 supply input. The VCCO_2 voltage can be 3.3V, 2.5V, or 1.8V to match the requirements of the external host, ideally 2.5V. Using 3.3V or 1.8V requires additional design considerations as the DONE and PROG_B pins are powered by the FPGA's 2.5V VCCAUX supply. Extended Spartan-3A family FPGAs do not support 1.8V PROMs because of the Spartan-3A FPGA's Power-On Reset (POR) voltage threshold, V_{CCO2T} , shown in the appropriate Extended Spartan-3A family data sheet and summarized in [Table 12-1, page 241](#).

See [XAPP453: The 3.3V Configuration of Spartan®-3 FPGAs](#) for additional information.

Daisy Chaining

If the application requires multiple FPGAs with different configurations, then configure the FPGAs using a serial daisy chain, as shown in [Figure 1-3, page 33](#). Use Slave Serial mode (M[2:0] = <1:1:1>) for all FPGAs in the daisy chain. After the lead FPGA is filled with its configuration data, the lead FPGA passes configuration data via its DOUT output pin to the next FPGA on the falling CCLK edge.

Table 8-1: Slave Serial Mode Connections

Pin Name	FPGA Direction	Description	During Configuration	After Configuration
HSWAP_EN, HSWAP, or PUDC_B	Input	User I/O Pull-Up Control. When Low during configuration, enables pull-up resistors in all I/O pins to respective I/O bank V_{CCO} input. 0: Pull-up during configuration 1: No pull-ups	Drive at valid logic level throughout configuration.	Spartan-3: Dedicated pin (don't care after configuration) Spartan-3E Spartan-3A Spartan-3AN Spartan-3A DSP: User I/O
M[2:0]	Input	Mode Select. Selects the FPGA configuration mode. See “ Design Considerations for the HSWAP, M[2:0], and VS[2:0] Pins, ” page 75.	M2 = 1, M1 = 1, M0 = 1 Sampled when INIT_B goes High.	User I/O
DIN	Input	Data Input.	Serial data provided by host. FPGA captures data on rising CCLK edge.	User I/O
CCLK	Input	Configuration Clock. If CCLK PCB trace is long or has multiple connections, terminate this output to maintain signal integrity. See “ CCLK Design Considerations, ” page 58.	External clock.	User I/O
INIT_B	Open-drain bidirectional I/O	Initialization Indicator. Active Low. Goes Low at start of configuration during Initialization memory clearing process. Released at end of memory clearing, when mode select pins are sampled.	Active during configuration. If CRC error detected during configuration, FPGA drives INIT_B Low.	User I/O. If unused in the application, drive INIT_B High to avoid a floating value. See INIT_B “ After Configuration ”.
DONE	Open-drain bidirectional I/O	FPGA Configuration Done. Low during configuration. Goes High when FPGA successfully completes configuration	Low indicates that the FPGA is not yet configured.	Pulled High via external pull-up. When High, indicates that the FPGA successfully configured.
PROG_B	Input	Program FPGA. Active Low. When asserted Low for 500 ns or longer (300 ns in the Spartan-3 FPGAs), forces the FPGA to restart its configuration process by clearing configuration memory and resetting the DONE and INIT_B pins once PROG_B returns High.	Must be High to allow configuration to start.	Drive PROG_B Low and release to reprogram FPGA.

JTAG Configuration Mode and Boundary-Scan

Spartan®-3 generation FPGAs have a dedicated four-wire IEEE 1149.1/1532 JTAG port that is always available any time the FPGA is powered and regardless of the mode pin settings. However, when the FPGA mode pins are set for JTAG mode ($M[2:0] = \langle 1:0:1 \rangle$), the FPGA waits to be configured via the JTAG port after a power-on event or after PROG_B is pulsed Low. Selecting the JTAG mode simply disables the other configuration modes. No other pins are required as part of the configuration interface. See “[Mode Pin Considerations when Programming a Spartan-3AN FPGA via JTAG using iMPACT](#)” for special mode pin requirements.

[Figure 9-1](#) illustrates a JTAG-only configuration interface. The figure shows optional components in gray. The JTAG interface is easily cascaded to any number of FPGAs by connecting the TDO output of one device to the TDI input of the next device in the chain. The TDO output of the last device in the chain loops back to the port connector.

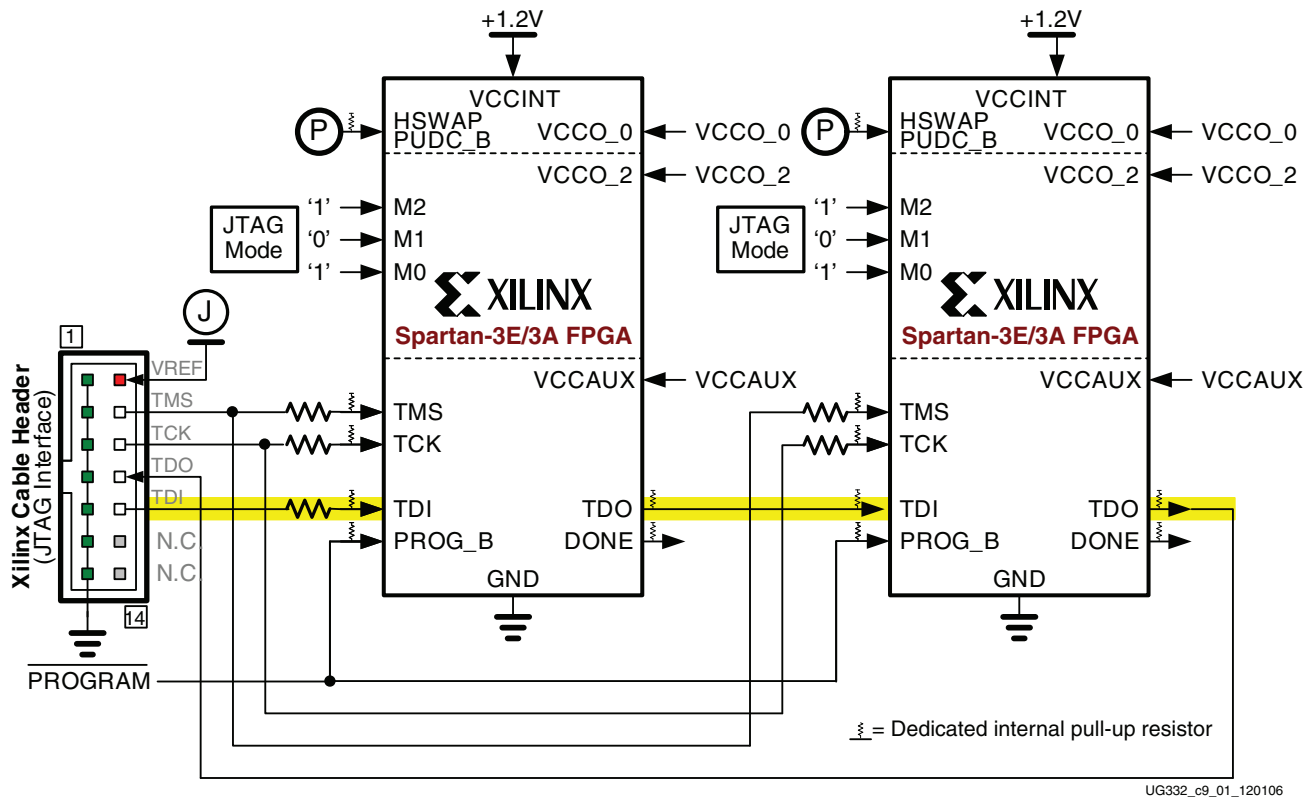


Figure 9-1: JTAG Configuration Interface

JTAG Cable Voltage Compatibility

The FPGA's JTAG interface is powered by the V_{CCAUX} supply. All of the user I/Os are separately powered by their respective $V_{CCO}_{\#}$ supplies.

The voltage supplied to the JTAG programming cable, shown as V_{REF} in Figure 9-1, may be different than the V_{CCAUX} supply. If the JTAG and V_{CCAUX} voltages are the same, simply connect the FPGA directly to the JTAG programming socket or use 0Ω resistors, as shown in Table 9-1. For Spartan-3AN FPGAs, V_{CCAUX} must be 3.3V. For Spartan-3A and Spartan-3A DSP FPGAs, V_{CCAUX} can optionally be set to 3.3V.

The interface becomes a bit more complex if the JTAG voltage is different than the FPGA's V_{CCAUX} voltage because current-limiting resistors are required. If the JTAG cable interface needs to be 3.3V to support devices in the JTAG chain, then place a series resistor between the 3.3V interface and the **TDI**, **TMS**, and **TCK** pins on the FPGA, as indicated in Table 9-1. The FPGA's **TDO** pin is a CMOS output powered by the V_{CCAUX} supply. Even when $V_{CCAUX} = 2.5V$, the **TDO** output can directly drive a 3.3V, input but with reduced noise immunity. See [XAPP453: The 3.3V Configuration of Spartan-3 FPGAs](#) for additional information.

Table 9-1: JTAG Cable Interface and Current-Limiting Resistor Requirements

JTAG Connector Supply Voltage	FPGA V _{CCAUX} Supply Voltage	Current-Limiting Resistors
2.5V	2.5V	None required or 0-ohm. Both voltages are identical
3.3V	2.5V	Use current-limiting resistors of 68Ω or larger.
3.3V	3.3V	None required or 0-ohm. Both voltages are identical

JTAG Device ID

Each Spartan-3 generation FPGA array type has a 32-bit device-specific JTAG device identifier as shown in [Table 12-4, page 246](#). The lower 28 bits represent the device vendor (Xilinx) and device identifier. The upper four bits, ignored by most tools, represent the revision level of the silicon mounted on the printed circuit board.

JTAG User ID

The Spartan-3 generation JTAG interface provides the option to store a 32-bit User ID, loaded during configuration. The User ID value is specified via the `UserID` configuration bitstream option, shown in [Table 11-2, page 232](#) or in Step 11, [Figure 1-7, page 44](#) from the ISE® Project Navigator software.

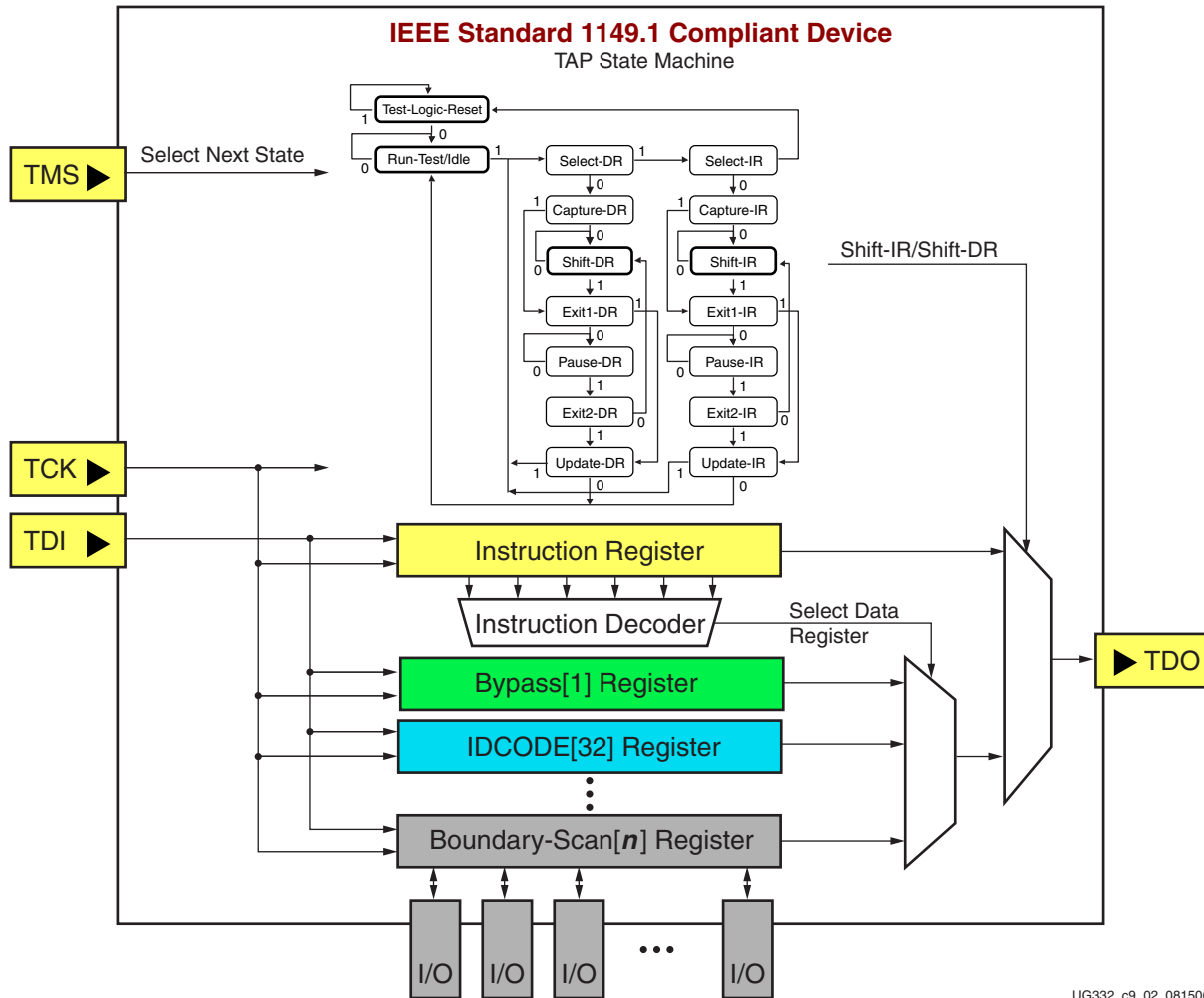
The user ID provides a convenient means to store an identifier or revision code for the FPGA bitstream loaded into the FPGA. This is different than the Device DNA identifier, which is unique to a specific Extended Spartan-3A family FPGA, not the bitstream, and permanently factory-programmed in the FPGA.

Using JTAG Interface to Communicate to a Configured FPGA Design

After the FPGA is configured, using any of the available modes, the JTAG interface offers a possible communications channel to internal FPGA logic. The [“Boundary-Scan \(BSCAN\),” page 253](#), design primitive provides two private JTAG instructions to create an internal boundary scan chain.

Boundary-Scan for Spartan-3 Generation FPGAs Using IEEE Standard 1149.1

Spartan-3 generation FPGAs are compliant with the IEEE Standard 1149.1 Test Access Port and Boundary-Scan Architecture. The architecture, outlined in [Figure 9-2](#), 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. Spartan-3 generation FPGAs also support a 32-bit Identification register in compliance with the standard. Outlined in the following sections are the details of the JTAG architecture for Spartan-3 generation FPGAs.



UG332_c9_02_081506

Figure 9-2: Typical JTAG (IEEE 1149.1) Architecture

Test Access Port (TAP)

The Spartan-3 generation TAP contains four mandatory dedicated pins as specified by the protocol given in Table 3-1 and illustrated in Figure 3-1, a typical JTAG architecture. Three input pins and one output pin control the 1149.1 Boundary-Scan TAP controller. Optional control pins, such as TRST (Test Reset) and enable pins 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 state machine (16 states) shown in Figure 9-3 and described in Table 9-2. The four mandatory TAP pins are outlined in Table 9-3.

A transition between the states only occurs on the rising edge of TCK, and each state has a different name. The two vertical columns with seven states each represent the Instruction Path and the Data Path. The data registers operate in the states whose names end with "DR" and the instruction register operates in the states whose names end in "IR". The states are otherwise identical.

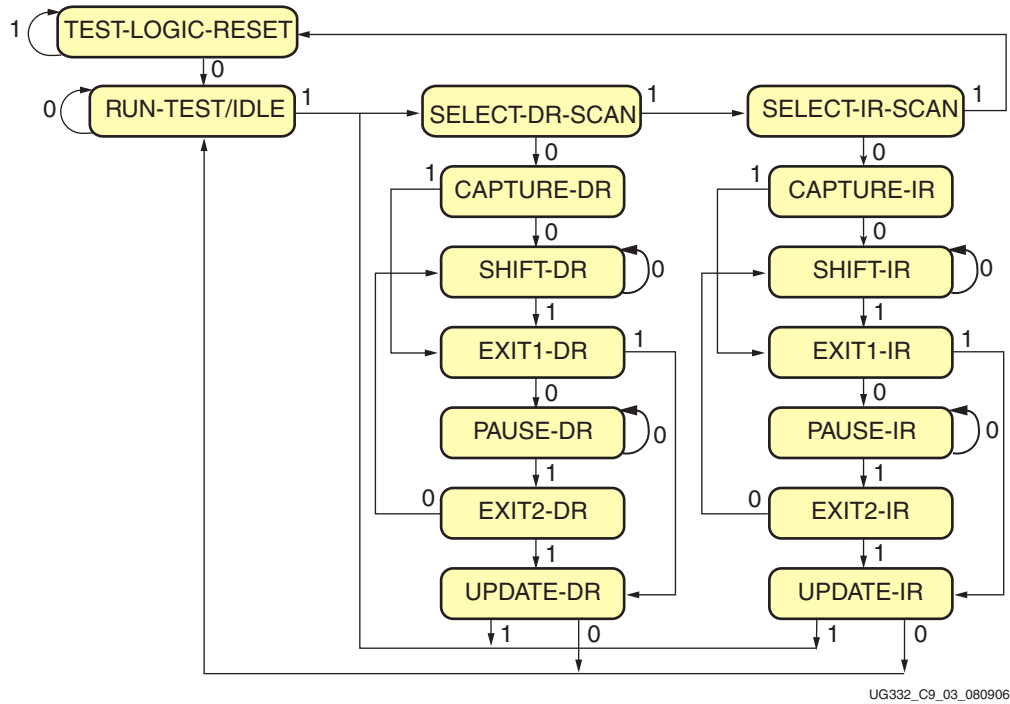


Figure 9-3: Test Access Port (TAP) State Machine

Table 9-2: TAP Controller States

State	Description
TEST-LOGIC-RESET	All JTAG logic is disabled, enabling the normal operation of the FPGA. No matter what the initial state of the controller is, the Test-Logic-Reset state can be entered by holding TMS High and pulsing TCK five times. This is why the Test Reset (TRST) pin is optional.
RUN-TEST/IDLE	The JTAG logic is active only if certain instructions are present. For example, if an instruction activates the self test, then it is executed when the controller enters this state. The JTAG logic is idle otherwise.
SELECT-DR-SCAN	Controls whether to enter the Data Path or the SELECT-IR-SCAN state.
SELECT-IR-SCAN	Controls whether or not to enter the Instruction Path. The Controller can return to the TEST-LOGIC-RESET state otherwise.
CAPTURE-IR	The shift register bank in the Instruction Register parallel loads a pattern of fixed values on the rising edge of TCK. The last two significant bits must always be "01".
SHIFT-IR	The Instruction Register gets connected between TDI and TDO, and the captured pattern gets shifted on each rising edge of TCK. The instruction available on the TDI pin is also shifted in to the Instruction Register.
EXIT1-IR	Controls whether to enter the PAUSE-IR state or UPDATE-IR state.
PAUSE-IR	Allows the shifting of the Instruction Register to be temporarily halted.
EXIT2-DR	Controls whether to enter either the SHIFT-IR state or UPDATE-IR state.
UPDATE-IR	The instruction in the Instruction Register is latched to the latch bank of the Instruction Register on every falling edge of TCK. This instruction becomes the current instruction once it is latched.

Table 9-2: TAP Controller States

State	Description
CAPTURE-DR	The data is parallel-loaded into the data registers selected by the current instruction on the rising edge of TCK.
SHIFT-DR	These controller states are similar to the SHIFT-IR, EXIT1-IR, PAUSE-IR, EXIT2-IR and UPDATE-IR states in the Instruction path.
EXIT1-DR	
PAUSE-DR	
EXIT2-DR	
UPDATE-DR	

Table 9-3: Spartan-3 Generation TAP Controller Pins

Pin	Description
TDI	Test Data In. This pin is the serial input to all JTAG instruction and data registers. The state of the TAP controller and the current instruction determine the register that 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	Test Data Out. This pin is the serial output for all JTAG instruction and data registers. The state of the TAP controller and the current instruction determine the register (instruction or data) that 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. TDO is an active driver output.
TMS	Test Mode Select. 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	Test Clock. TCK sequences the TAP controller and the JTAG registers.

Notes:

- As specified by the IEEE Standard, the TMS and TDI pins both have internal pull-up resistors. These internal pull-up resistors are active before configuration, regardless of the mode selected. See [Table 2-13, page 65](#) for resistor values. After configuration, these resistors are controlled by the *TmsPin* and *TdiPin* bitstream generator option settings, shown in [Table 11-2, page 232](#).

TAP Controller

[Figure 9-3](#) 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.

Spartan-3 generation FPGAs support the mandatory IEEE 1149.1 commands, as well as several Xilinx vendor-specific commands. The [EXTEST](#), [INTEST](#), [SAMPLE/PRELOAD](#), [BYPASS](#), [IDCODE](#), [USERCODE](#), and [HIGHZ](#) instructions are all included. The TAP also supports internal user-defined registers (USER1 and USER2) and configuration/readback of the device.

The Spartan-3 generation Boundary-Scan operations are independent of configuration mode selections. The Boundary-Scan mode overrides other mode selections. For this

reason, Boundary-Scan instructions using the Boundary-Scan register (SAMPLE/PRELOAD, INTEST, and EXTEST) must not be performed during the FPGA configuration process. All instructions, except the user-defined instructions, are available before a Spartan-3 generation FPGA device is configured. After configuration, all instructions are available.

JSTART and **JSHUTDOWN** are instructions specific to the Spartan-3 generation FPGA architecture and configuration flow. See [DS099: Spartan-3 FPGA Family Data Sheet](#) for details. *In Spartan-3 generation FPGAs, the TAP controller is not reset by the PROG_B pin and can only be reset by bringing the controller to the TLR state. The TAP controller is reset on power up.*

For details on the standard Boundary-Scan instructions EXTEST, INTEST, and BYPASS, refer to the IEEE Standard.

Caution! For JTAG compliance during pre-configuration boundary scan, the internal pull-ups should be made active by asserting HSWAP or PUDC_B Low.

Boundary-Scan Architecture

Spartan-3 generation FPGA 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, as described in [Table 9-4](#).

Table 9-4: Spartan-3 Generation JTAG Registers

Register Name	Register Length	Description
Boundary-Scan Register	3 bits per I/O	Controls and observes input, output, and output enable
Instruction Register	6 bits	Holds current instruction OPCODE and captures internal device status
BYPASS Register	1 bit	Bypasses the device
Identification Register	32 bits	Captures the Device ID
JTAG Configuration Register	32 bits	Allows access to the configuration bus when using the CFG_IN or CFG_OUT instructions
USERCODE Register	32 bits	Captures the user-programmable code
User-Defined Registers (USER1 and USER2)	Design-specific	Design-specific

Boundary-Scan Register

Each user I/O block (IOB), whether connected to a package pin or unbonded, contains additional logic that forms the boundary-scan data register, as shown in [Figure 9-4](#). Boundary-Scan operations are independent of how an individual I/O block is configured. By default, each I/O block starts as bidirectional with 3-state control. Later, it can be configured via JTAG operations to be an input, output, or 3-state pin.

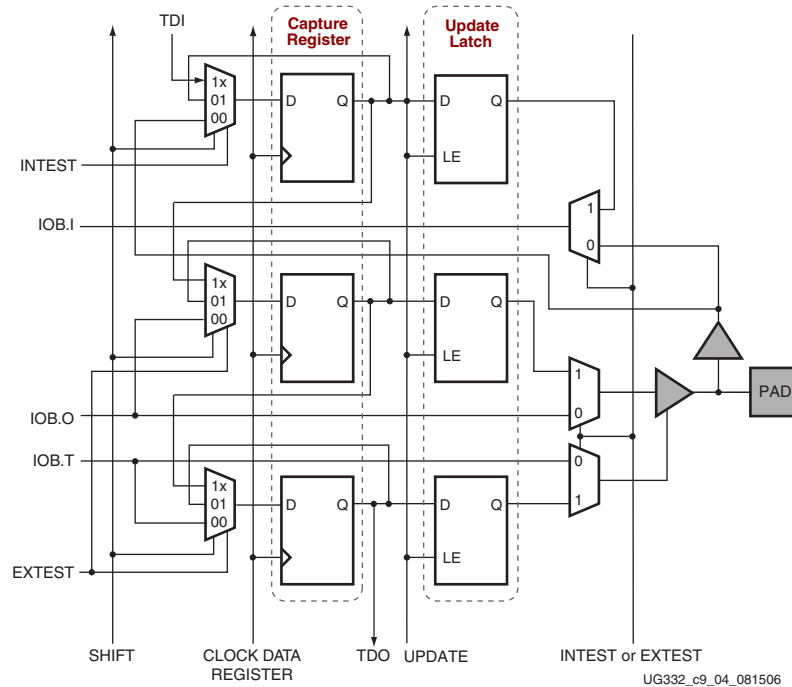


Figure 9-4: Boundary-Scan Logic per I/O Pin

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.

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.

Internal pull-up and pull-down resistors should be considered when test vectors are being developed for testing opens and shorts. The Boundary-Scan mode determines whether an I/O block has a pull-up resistor.

Bit Sequence Boundary-Scan Register

The order of 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 Spartan-3 generation FPGAs. The bit sequence always has the same bit order and the same number of bits and is independent of the design.

The BSDL files are provided with the Xilinx ISE Development Software or can be downloaded directly from the Xilinx web site. From the Xilinx web site, select **BSDL Models**, select the FPGA family, then click **Search**.

- **Xilinx Download Center**
<http://www.xilinx.com/support/download/index.htm>

Instruction Register

The Instruction Register (IR) for the Spartan-3 generation FPGA is connected between TDI and TDO during an instruction scan sequence. In preparation for an instruction scan sequence, the instruction register is parallel-loaded with a fixed instruction capture pattern. This pattern is shifted out onto TDO (LSB first), while an instruction is shifted into the instruction register from TDI.

To invoke an operation, load the desired OPCODE from [Table 9-5](#) into the Instruction Register (IR). The length of the instruction register varies by device type. However, the IR is six bits wide for all Spartan-3 generation FPGAs.

Note: In general, all JTAG OPCODEs are identical among Spartan-3 generation FPGA families. However, the EXTEST instruction is different between Spartan-3 FPGAs and FPGAs from the Spartan-3E or Extended Spartan-3A family families.

Table 9-5: Spartan-3 Generation Boundary-Scan Instructions

Boundary-Scan Command	Instruction	Description
EXTEST (Spartan-3E, Spartan-3A/3AN, Spartan-3A DSP FPGAs)	001111	Enables Boundary-Scan EXTEST operation.
EXTEST (Spartan-3 FPGA)	000000	
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-state output pins while enabling BYPASS Register.
JPROGRAM	001011	Equivalent to and has the same effect as PROGRAM.
JSTART	001100	Clocks the startup sequence when Startup clock source is TCK (<i>StartupClk:JtagClk</i>).
JSHUTDOWN	001101	Clocks the shutdown sequence.
ISC_ENABLE	010000	Marks the beginning of ISC configuration. Full shutdown is executed.
ISC_PROGRAM	010001	Enables in-system programming.
ISC_NOOP	010100	No operation.
ISC_READ	010101	Used to read back configuration data.

Table 9-5: Spartan-3 Generation Boundary-Scan Instructions (Cont'd)

Boundary-Scan Command	Instruction	Description
ISC_DISABLE	010110	Completes ISC configuration. Startup sequence is executed.
ISC_DNA	110001	Extended Spartan-3A family FPGAs: Read Device DNA. See “JTAG Access to Device Identifier,” page 297.
BYPASS	111111	Enables BYPASS.
RESERVED	All other codes	Xilinx reserved instructions.

Table 9-6 shows the instruction capture values loaded into the IR as part of an instruction scan sequence.

Table 9-6: Instruction Capture Values

TDI →	IR[5]	IR[4]	IR[3]	IR[2]	IR[1:0]	←TDO
	DONE	INIT(1)	ISC_ENABLED	ISC_DONE	0 1	

BYPASS Register

The BYPASS register, which consists of a single flip-flop between TDI and TDO, is required in all JTAG IEEE 1149.1-compliant devices. It passes data serially from the TDI pin to the TDO pin during a bypass instruction. The BYPASS register initializes to zero when the TAP controller is in the CAPTURE-DR state.

Identification (IDCODE) Register

Spartan-3 generation FPGAs have a 32-bit identification register called the IDCODE register. The IDCODE is based on the IEEE 1149.1 standard, and is a fixed, vendor-assigned value that is used to identify electrically the manufacturer and the type of device that is being addressed. This register allows easy identification of the part being tested or programmed by Boundary-Scan, and it can be shifted out for examination by using the IDCODE instruction.

The last bit of the IDCODE is always 1 (based on JTAG IEEE 1149.1). The last three hex digits appear as 0x093.

JTAG Configuration Register (Boundary-Scan)

The JTAG Configuration register is a 32-bit register. This register allows access to the configuration bus and readback operations.

The JTAG Configuration register does not pass TDI data through the register to TDO. When the CFG_IN JTAG instruction is active, the JTAG Configuration register is input-only. When the CFG_OUT JTAG instruction is active, the JTAG Configuration register is output-only.

Applications requiring standard FPGA configuration over JTAG should refer to [XAPP058](#), *Xilinx In-System Programming Using an Embedded Microcontroller*. Applications requiring custom FPGA configuration over JTAG should refer to [XAPP452](#), *Spartan-3 FPGA Family Advanced Configuration Architecture*, for information about the configuration bus.

Configuration bus commands must arrive at the FPGA TDI pin on a 32-bit shift boundary as explained in [XAPP188, Configuration and Readback of Spartan-II and Spartan-IIE FPGAs Using Boundary Scan](#).

USERCODE Register

The USERCODE instruction is supported in Spartan-3 generation FPGAs. This register allows a user to specify a design-specific identification code. The USERCODE can be programmed into the device and can be read back for verification later. The USERCODE is embedded into the bitstream during bitstream generation (BitGen -g UserID option) and is valid only after configuration. If the device is blank or the USERCODE was not programmed, the USERCODE register contains 0xFFFFFFFF.

USER1 and USER2 Registers

The USER1 and USER2 registers are only available after configuration. These two registers, if used in the application, must be implemented using FPGA logic. The “[Boundary-Scan \(BSCAN\)](#)” library primitive is required when creating these registers. This primitive is only required for driving internal scan chains (USER1 and USER2). These registers can be accessed after they are defined via the JTAG interface.

A common input pin (TDI) and shared output pins represent the state of the TAP controller (RESET, SHIFT, and UPDATE).

Using Boundary-Scan in Spartan-3 Generation FPGAs

[Figure 9-5](#) shows an example timing waveform for boundary-scan operations. Timing specifications are listed in the data sheet for each Spartan-3 generation FPGA family.

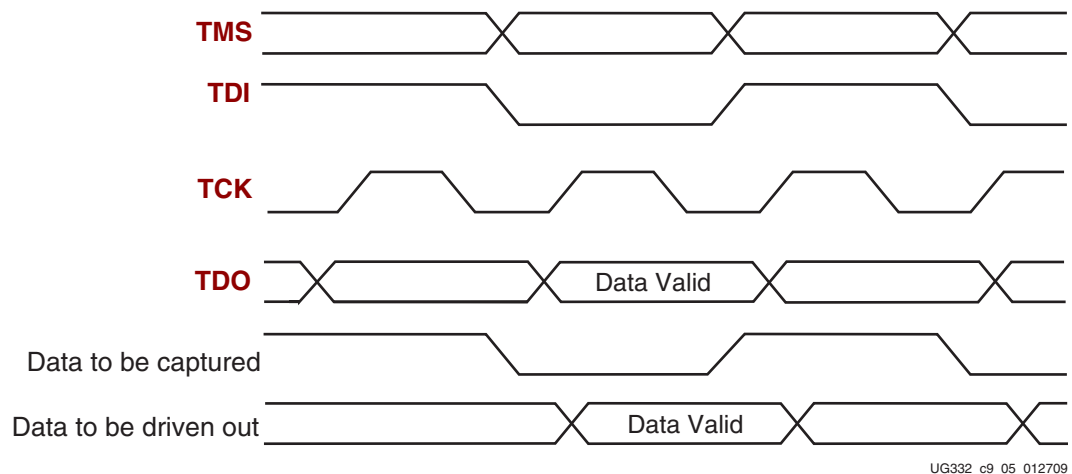


Figure 9-5: Spartan-3 Generation Boundary-Scan Timing Waveforms

Programming Cables and Headers

Xilinx provide various programming cables that support the design and development phase of a project.

- **Platform Cable USB II**
<http://www.xilinx.com/products/devkits/HW-USB-II-G.htm>

- **Parallel Cable IV**

<http://www.xilinx.com/products/devkits/HW-PC4.htm>

If possible, place a target interface connector on the FPGA board to facilitate easy programming. Xilinx recommends using the high-performance ribbon cable option, pictured in [Figure 9-6, page 208](#), for maximum performance and best signal integrity.

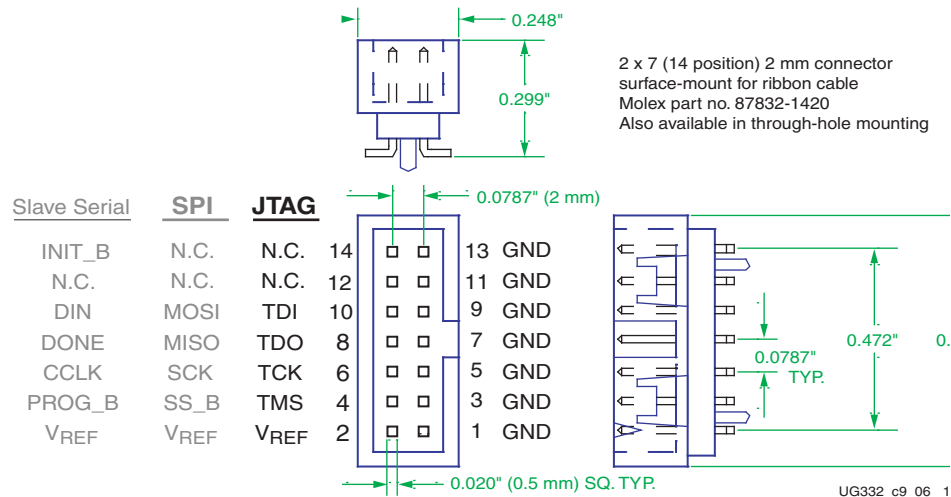


Figure 9-6: Target Interface Connector Dimensions and Pin Assignments

Such connectors are available in both through-hole and surface mount configurations, as shown in [Table 9-7](#). Use shrouded or keyed connectors to ensure guarantee proper orientation when inserting the cable. The specified connector requires only 0.162 square inches of board space.

Table 9-7: Mating Connectors for 2 mm pitch, 14 Conductor Ribbon Cable

Manufacturer ⁽¹⁾	Connector Style and Vendor Part Number			Vendor Web Site
	Surface Mount, Vertical	Through-Hole, Vertical	Through-Hole, Right Angle	
Molex	87832-1420	87831-1420	87833-1420	www.molex.com
FCI	98424-G52-14	98414-G06-14	98464-G61-14	www.fciconnect.com
Comm Con Connectors	2475-14G2	2422-14G2	2401R-G2-14	www.commcon.com

Notes:

1. Some manufacturer pin assignments may not conform to Xilinx pin assignments. Please refer to the manufacturer's data sheet for more information.
2. Additional ribbon cables can be purchased separately from the Xilinx Online Store (www.xilinx.com/store).

Pin 2 of the connector provides a reference voltage for the output buffers that drive the TDI, TCK, and TMS pins. Because these pins are powered by V_{CCAUX} on Spartan-3 generation FPGAs, connect the V_{CCAUX} supply to pin 2 of the connector.

Programming an FPGA Using JTAG

The JTAG interface is also a convenient means for downloading an FPGA design during development and debugging.

First, generate an FPGA bitstream as described in “Setting Bitstream Options, Generating an FPGA Bitstream,” page 42

The following steps graphically describe how to create a PROM file using iMPACT from within the ISE Project Navigator. This particular example shows how to configure the XC3S500E FPGA on the [Spartan-3E Starter Kit](#) board. Besides the FPGA, the JTAG chain on the board includes a Xilinx Platform Flash PROM and a Xilinx CPLD.

1. From within the ISE Project Navigator, double-click **Configure Device (iMPACT)** from the Processes pane, as shown in [Figure 9-7](#).

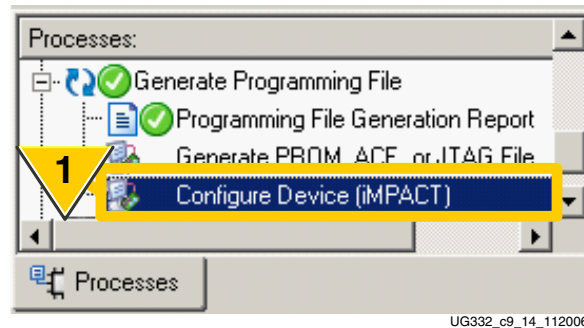


Figure 9-7: Double-click **Configure Device (iMPACT)**

2. As shown in [Figure 9-8](#), select **Configure devices using Boundary-Scan (JTAG)**.

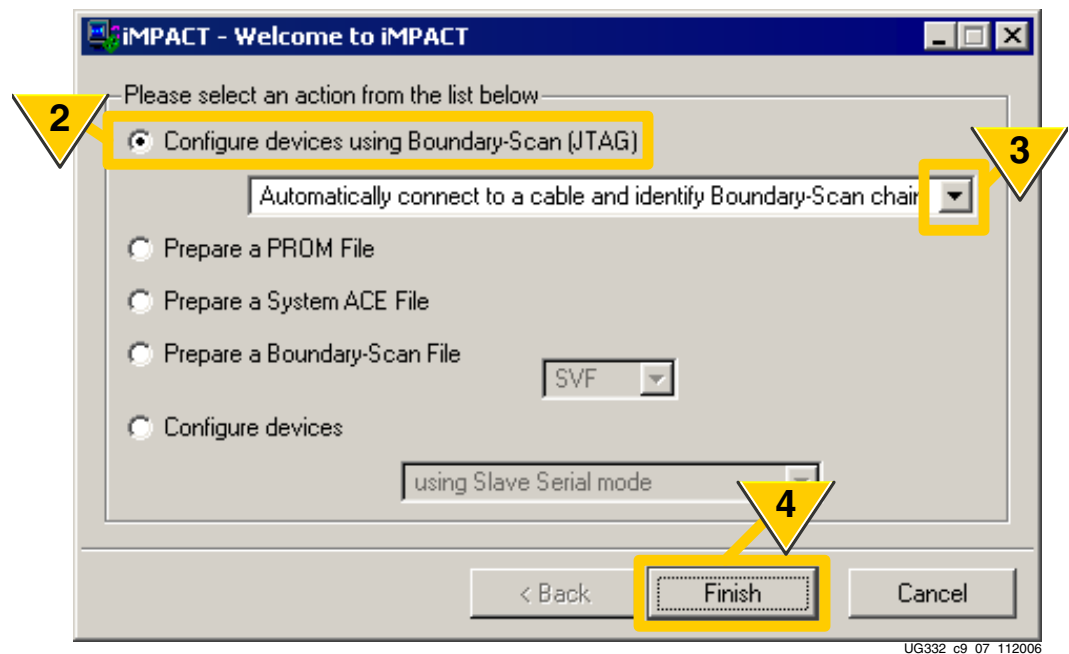


Figure 9-8: **Configure Devices Using JTAG**

3. If the board is powered and the Xilinx programming cable properly connected, the iMPACT software automatically initializes the JTAG chain and detects the various devices on the chain.
4. Click **Finish**.

- As shown in [Figure 9-9](#), the iMPACT software automatically detected the devices on the chain. In this example, a Xilinx XC3S500E Spartan-3E FPGA is first in the chain, followed by a Xilinx XCF04S Platform Flash PROM, followed by a Xilinx XC2C64A CPLD in the final position. The devices are yet unprogrammed.

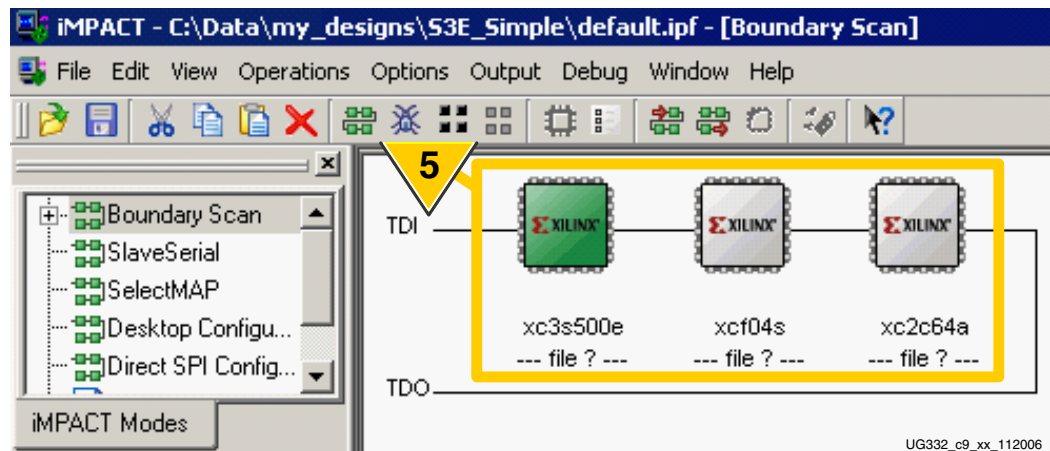


Figure 9-9: iMPACT Automatically Detects Devices on the JTAG Chain

- As shown in [Figure 9-10](#), the iMPACT software automatically prompts for the FPGA bitstream. Select the desired bitstream to download specifically to the FPGA.
- Click **Open**.

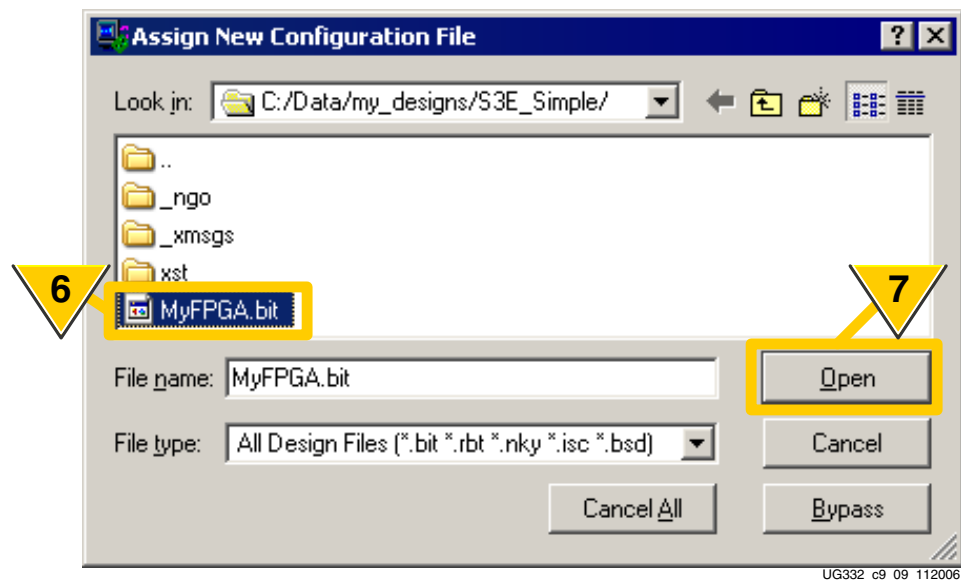


Figure 9-10: iMPACT Prompts for FPGA Bitstream

- As shown in [Figure 9-11](#), the iMPACT software automatically detects that the FPGA bitstream was generated for a non-JTAG configuration method. The iMPACT software automatically adjusts the Startup clock setting for successful JTAG configuration (*StartupClk:JtagClk*). The original bitstream file is unaffected.

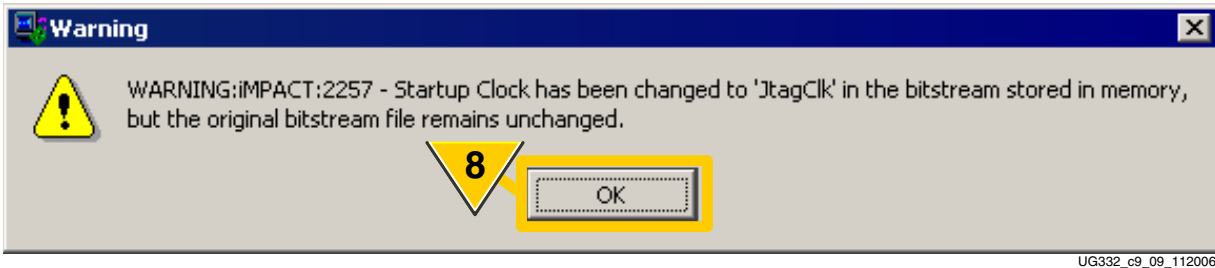


Figure 9-11: **iMPACT Automatically Adjusts FPGA Startup Clock for JTAG Configuration**

9. For faster downloading and a shorter FPGA debugging cycle, there is no need to program the Platform Flash PROM or CPLD unless actually desired. To skip programming the Platform Flash PROM, click Bypass, as shown in Figure 9-12.

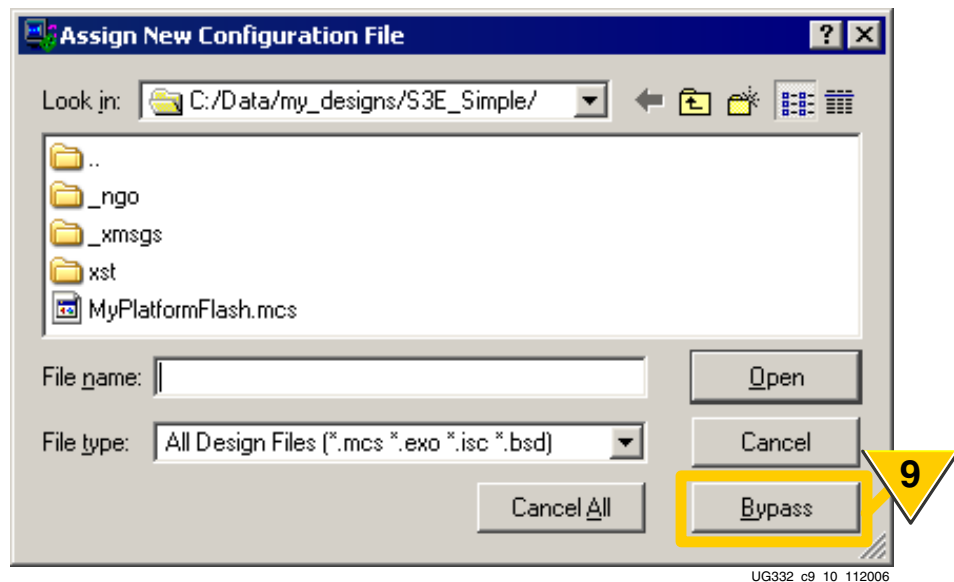


Figure 9-12: **Click Bypass to Skip Platform Flash Programming**

10. Similarly, click Bypass to skip programming of the CPLD, as shown in Figure 9-13.

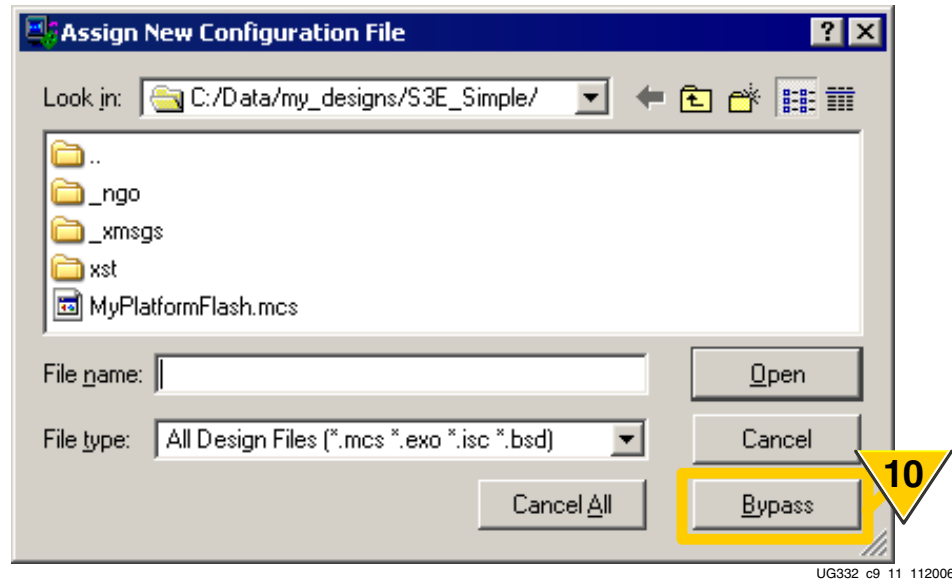


Figure 9-13: Click Bypass to Skip CPLD Programming

- As shown in Figure 9-14, the iMPACT software updates the display, showing the files assigned to each device in the JTAG chain. In this example, the XCF04S Platform Flash and XC2C64A CPLD are “bypassed” and are not programmed. Click the FPGA to highlight it on the display.

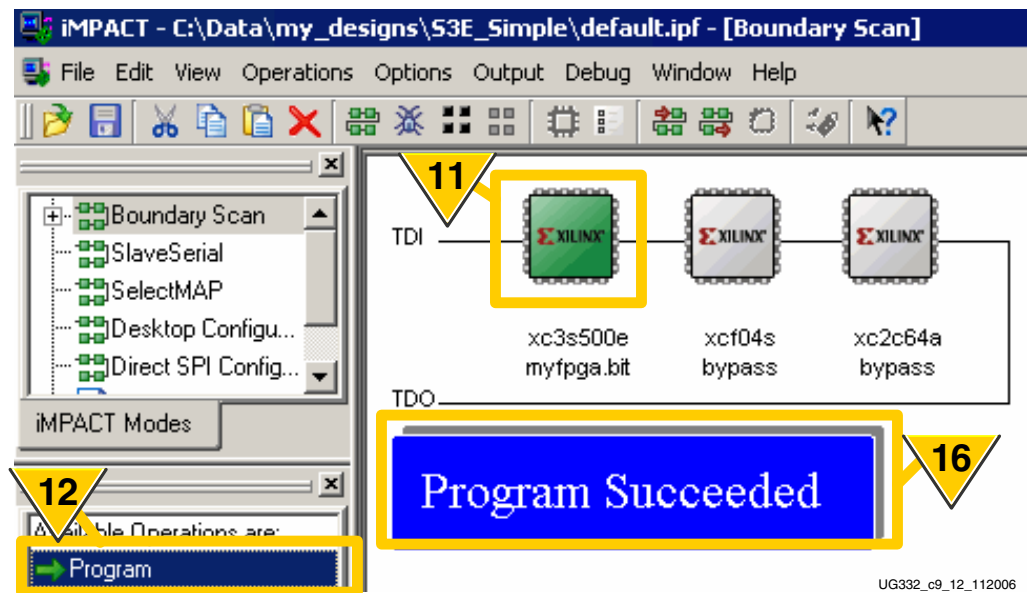


Figure 9-14: Double-Click Program to Configure FPGA via JTAG

- Once the FPGA is highlighted, the associated Available Operations are enabled on the display. Double-click Program.
- The Programming Properties dialog box appears, as shown in Figure 9-15.

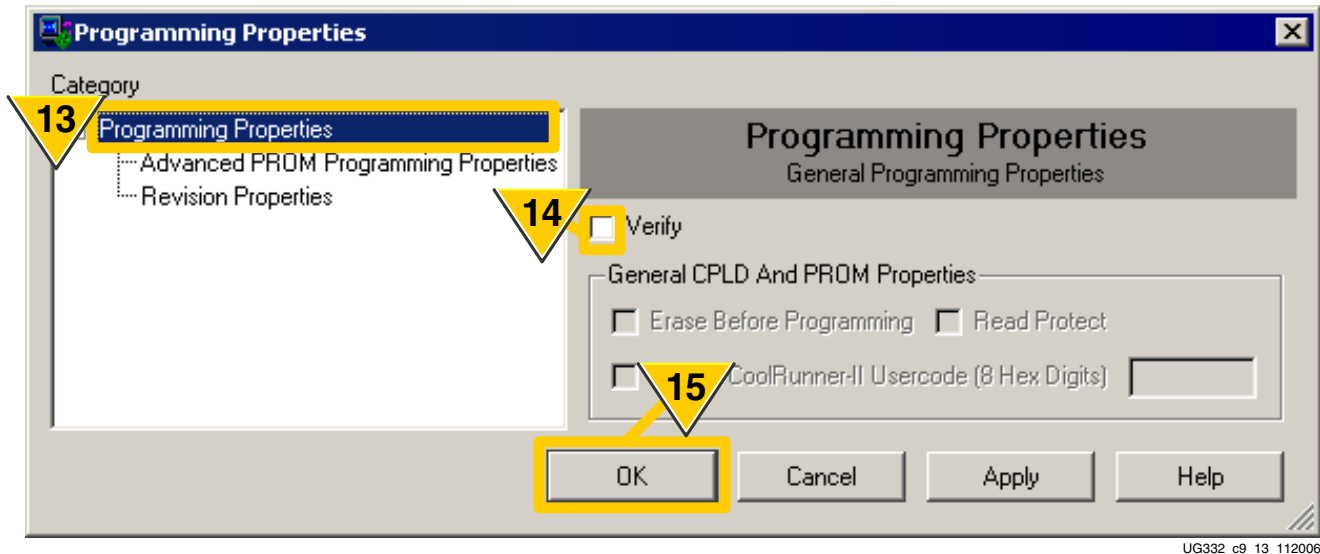


Figure 9-15: FPGA Programming Options

14. The iMPACT software provides a **Verify** feature, even for FPGA programming. Typically, the Verify function is not used when downloading the FPGA for debugging purposes.
15. Click **OK** to start the programming process.
16. The iMPACT software indicates when programming is complete, as shown in [Figure 9-14](#). The iMPACT software also forces the FPGA to reconfigure on the board. The FPGA is downloaded with the specified FPGA bitstream.

Mode Pin Considerations when Programming a Spartan-3AN FPGA via JTAG using iMPACT

When iMPACT 9.1i configures the Spartan-3AN FPGAs, it first programs the internal SPI Flash PROM. After this configuration is complete, a reboot is triggered and the FPGA configures itself from the internal SPI PROM. When the reboot is triggered, the mode pins M[2:0] are sampled. For the configuration to complete successfully, the FPGA mode select pins must be set to $M[2:0] = \langle 0:1:1 \rangle$, which is the Internal Master SPI mode.

If you are configuring from iMPACT and your mode pins are set to JTAG mode $M[2:0] = \langle 1:0:1 \rangle$, configuration of the FPGA will not complete. To finish configuration of the FPGA, you can simply change the mode pins to Internal Master SPI mode and pulse the PROG pin to trigger configuration, or reconfigure through iMPACT.

In iMPACT 9.2i and later, you have the option to either configure the FPGA directly through JTAG mode or to program the Internal SPI PROM and then configure through Internal Master SPI mode.

Configuration via JTAG using an Embedded Controller

By using an embedded controller to program Spartan-3 generation FPGAs from an on-board RAM or EPROM, designers can easily upgrade, modify, and test designs, even in the field. The design in XAPP058 is easily modified for remote downloading applications and the included C code can be compiled for any microcontroller.

- XAPP058 Xilinx In-System Programming Using an Embedded Microcontroller
http://www.xilinx.com/support/documentation/application_notes/xapp058.pdf

Internal Master SPI Mode

The Internal Master SPI Flash mode is only available on the Spartan[®]-3AN FPGA family. The Spartan-3AN FPGA family has integrated In-System Flash (ISF) memory, primarily for FPGA configuration. The ISF memory is sufficiently large to store two FPGA configuration bitstreams (MultiBoot) plus additional nonvolatile data storage for the FPGA application.

Spartan-3AN FPGAs also support all of the other Extended Spartan-3A family FPGA configuration modes shown in [Table 2-1, page 50](#).

Caution! This configuration mode is only supported by the Spartan-3AN FPGA family. The V_{CCAUX} supply **MUST** be 3.3V.

Figure 10-1 shows the logic levels and signals involved during configuration.

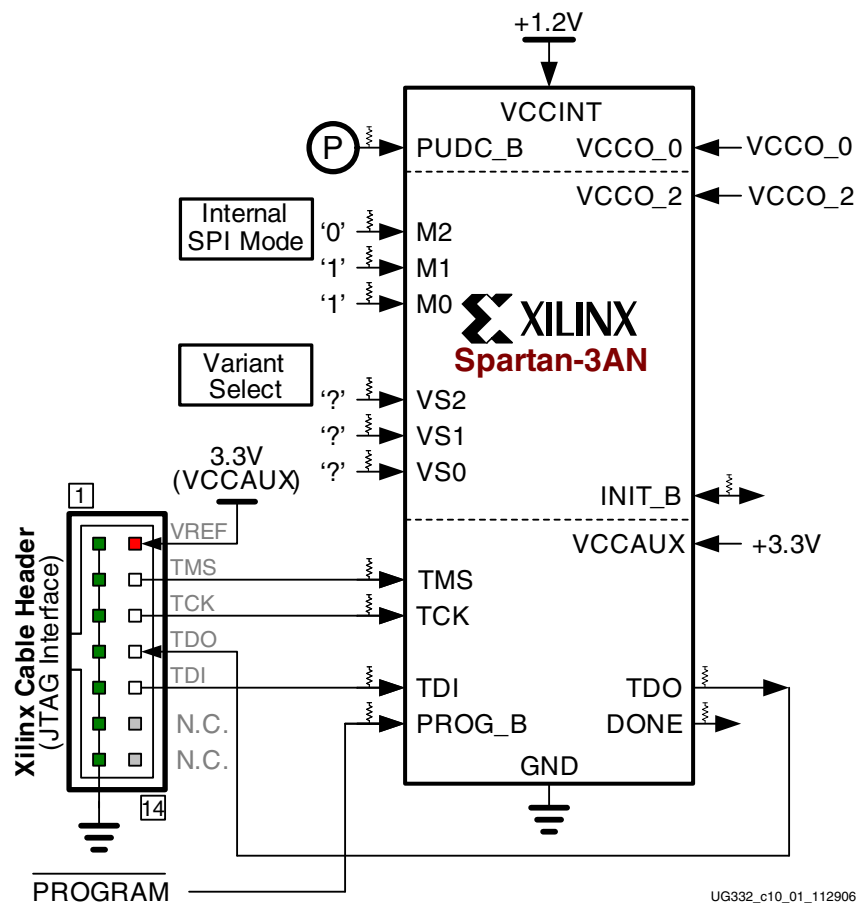


Figure 10-1: Spartan-3AN FPGA using Internal Master SPI Flash Mode

Internal Flash Memory

The amount of ISF memory varies by Spartan-3AN FPGA logic density as shown in [Table 10-1](#). The amount of Flash memory exceeds the amount required to configure the FPGA. There is sufficient additional memory for at least two uncompressed bitstream images to support MultiBoot or for additional nonvolatile storage for the FPGA application.

Table 10-1: Number of Bits to Program a Spartan-3AN FPGA and Internal SPI Flash Memory

FPGA	Number of Configuration Bits (Uncompressed)	In-System Flash Memory
XC3S50AN	437,312	1 Mbit
XC3S200AN	1,196,128	4 Mbit
XC3S400AN	1,886,560	4 Mbit
XC3S700AN	2,732,640	8 Mbit
XC3S1400AN	4,755,296	16 Mbit

Mode Select Pins, M[2:0]

The Spartan-3AN FPGA family is generally designed to be pin and function compatible with the Spartan-3A/3A DSP FPGA families. The Spartan-3AN FPGA family supports all the same configuration modes as the Spartan-3A/3A DSP FPGAs and adds the ability to configure from the internal In-System Flash memory.

To configure from Internal Master SPI Flash mode, the FPGA mode select pins must be set to M[2:0] = <0:1:1>. Furthermore, the V_{CCAUX} supply must be 3.3V.

Variant Select Pins, VS[2:0]

For backward compatibility, the Spartan-3AN FPGA monitors the variant-select pins, VS[2:0], to decide which read command to issue to the SPI Flash PROM. Spartan-3AN FPGAs and the integrated SPI serial Flash support the variant-select codes listed in [Table 10-2](#). The choice of a variant select code potentially affects configuration performance. For more details on the Spartan-3AN FPGA read commands, see [UG333, Spartan-3AN In-System Flash User Guide](#).

Furthermore, the VS[2:0] pins have dedicated pull-up resistors that are active, regardless of the PUDC_B pin, whenever the M[2:0] mode-select pins are set for Internal Master SPI mode.

Table 10-2: Spartan-3AN FPGA Supported Variant Select (VS[2:0]) Options

Variant Select Pins VS[2:0]	SPI Flash Read Command (Command Code)	Supported by Spartan-3AN FPGA Family?	Maximum CLK Frequency
<1:1:1>	FAST_READ (0x0B)	Yes	50 MHz
<1:0:1>	READ (0x03)	Yes	33 MHz
All Others	--	No	--

Supply Voltage Requirements

The Spartan-3AN FPGA family imposes some minor restrictions on FPGA supply voltages.

VCCAUX

The VCCAUX supply input must be 3.3V. The VCCAUX rail supplies power to the In-System Flash memory.

VCCO_2

The VCCO_2 supply rail, which must be the same voltage as the configuration memory in other configuration modes, has no such restriction on Spartan-3AN FPGAs. However, VCCO_2 must reach 2.0V to meet the power-on requirements; after configuration, it can drop down to a lower level.

Sequencing

When configuring from the In-System Flash, V_{CCAUX} must be in the recommended operating range; on power-up make sure V_{CCAUX} reaches at least 3.0V before INIT_B goes High to indicate the start of configuration. V_{CCINT} , V_{CCAUX} , and V_{CCO} supplies to the FPGA can be applied in any order if this requirement is met. However, an external configuration source might have specific requirements. Check the data sheet for the attached configuration source. Apply V_{CCINT} last for lowest overall power consumption (see the chapter called “Powering Spartan-3 Generation FPGAs” in UG331 for more information). The FPGA typically delays configuration long enough for the configuration source to be ready. If the configuration source is not ready when the FPGA begins configuration, the Configuration Watchdog Timer will allow the FPGA to automatically re-attempt configuration.

Accessing the Internal SPI Flash PROM After Configuration

The FPGA application has full access to the internal In-System Flash memory after configuration using the SPI_ACCESS design primitive, as shown in [Figure 10-2](#).

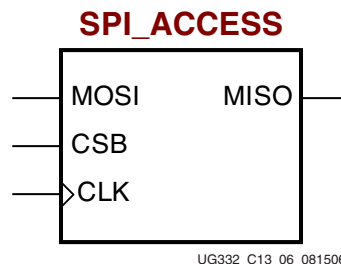


Figure 10-2: Spartan-3AN SPI_ACCESS Design Primitive

Details on accessing the In-System Flash memory after configuration, from inside the FPGA application, are found in **UG333: Spartan-3AN In-System Flash User Guide**.

- **UG333: Spartan-3AN In-System Flash User Guide**
www.xilinx.com/support/documentation/user_guides/ug333.pdf

No Configuration Daisy Chains in Internal Master SPI Mode

Spartan-3AN FPGAs do not support multi-FPGA daisy chains when configuring from Internal Master SPI mode. The FPGA does not supply the DOUT or CCLK outputs required for serial daisy chains when configuring in this mode.

However, the Spartan-3AN FPGA supports daisy chaining when configured using any of the other modes or when configured in a Slave configuration mode.

Generating the Bitstream for a Master SPI Configuration

To create the FPGA bitstream for a Internal Master SPI configuration, follow the steps outlined in “[Setting Bitstream Options, Generating an FPGA Bitstream](#),” page 42. For an FPGA configured in Internal Master SPI mode, set the following bitstream generator options.

ConfigRate: CCLK Frequency

Set the *ConfigRate* option for 33 MHz. Using the ISE® software Project Navigator, the Configuration Rate frequency is set in Step 7 in [Figure 1-7, page 44](#).

```
-g ConfigRate:33
```

StartupClk: CCLK

By default, the configuration Startup clock source is the internally generated CCLK. Keep the *StartupClk* bitstream generation option, shown as Step 13 in [Figure 1-8, page 45](#).

```
-g StartupClk:Cclk
```

DriveDone: Actively Drive DONE Pin

In a single FPGA design or for the Master FPGA in a multi-FPGA daisy chain, set the FPGA to actively drive the DONE pin after successfully completing the configuration process. Using ISE Project Navigator, check the **Drive Done Pin High** option, shown as Step 16 in [Figure 1-8, page 45](#).

-g DriveDone:Yes

Programming a Spartan-3AN FPGA Using JTAG

A Spartan-3AN FPGA is programmed using JTAG and iMPACT software in the same way described for other FPGA families in [“Programming an FPGA Using JTAG” in Chapter 9](#). The iMPACT software only requires associating a bitstream with the FPGA, and will automatically generate the PROM file for the In-System Flash, program the Flash in the Spartan-3AN FPGA, and then configure the Spartan-3AN FPGA from the In-System Flash. See [“Mode Pin Considerations when Programming a Spartan-3AN FPGA via JTAG using iMPACT” in Chapter 9](#).

Preparing an In-System Flash Programming File

This section provides guidelines to create a programming file for the Spartan-3AN In-System Flash (ISF) memory. These steps are not needed when programming a single bitstream into the ISF using iMPACT.

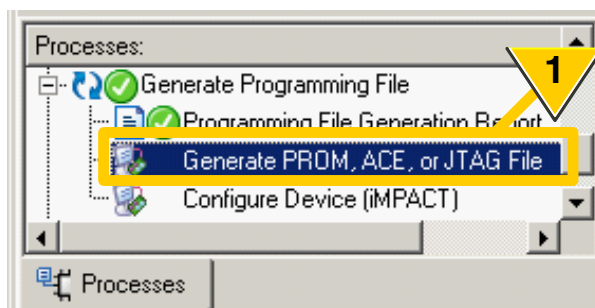
Caution! Requires ISE 9.1i, Service Pack 3 or later.

The Xilinx® software tools, [iMPACT](#) or [PROMGen](#), generate files from the Spartan-3AN FPGA bitstream or bitstreams. The Spartan-3AN ISF memory is a serial, SPI-based memory and data bytes are stored most-significant bit (MSB) first. When using PROMGen, the `-spi` option is **required** for proper formatting.

iMPACT

The following steps graphically describe how to create an SPI-formatted PROM file using iMPACT from within the ISE Project Navigator. To create a Spartan-3AN MultiBoot image for an SPI Flash memory, see [“Generating an Extended Spartan-3A Family MultiBoot PROM Image using iMPACT,”](#) page 278.

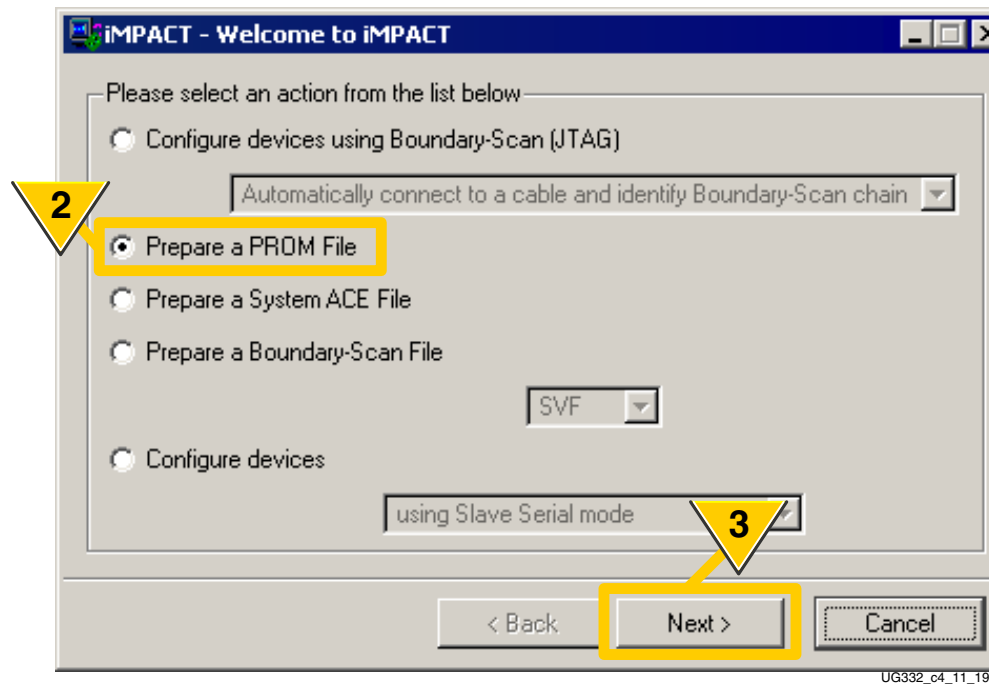
1. From within the ISE Project Navigator, double-click **Generate PROM, ACE, or JTAG File** from within the Process pane, as shown in [Figure 10-3](#).



UG332_c4_10_110206

Figure 10-3: Double-click **Generate PROM, ACE or JTAG File**

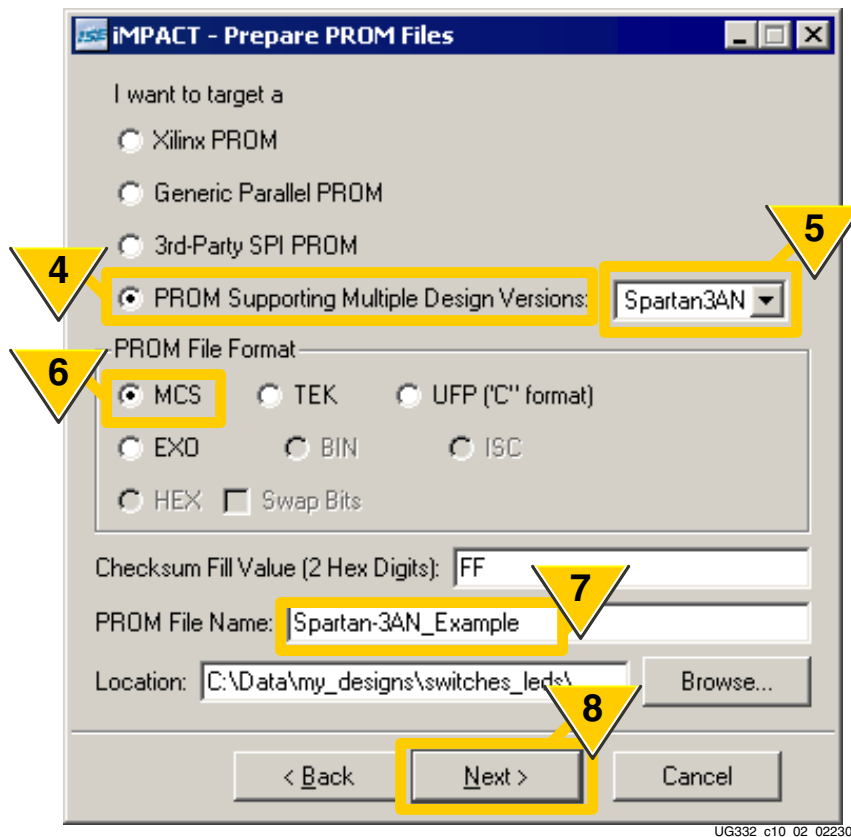
2. As shown in [Figure 10-4](#), select **Prepare a PROM File**.



UG332_c4_11_19

Figure 10-4: Prepare a PROM File

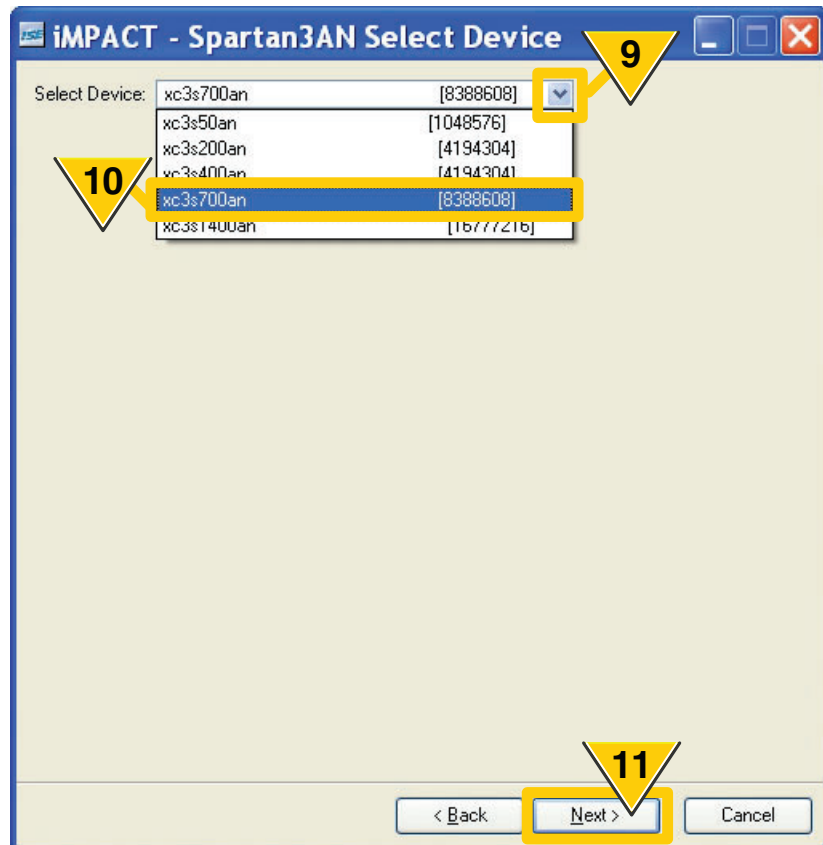
3. Click Next.
4. As shown in [Figure 10-5](#), format the FPGA bitstream or bitstreams for a **PROM Supporting Multiple Design Versions**.



UG332_c10_02_022307

Figure 10-5: Set Options for Spartan-3AN In-System Flash PROM

5. Select **Spartan3AN** from the drop list.
6. Select a **PROM File Format**.
7. Enter a **PROM File Name**.
8. Click **Next**.
9. Click the drop list to **Select Device**.



UG332_c10_03_052207

Figure 10-6: Select a Spartan-3AN FPGA

10. Choose a specific Spartan-3AN FPGA device. The bit size of the In-System Flash memory for the associated FPGA is also displayed.
11. Click Next.
12. The Default Spartan-3AN configuration bitstream (Bitstream 0) is always located at address 0. Bitstream 0 is the bitstream that the FPGA automatically loads when power is applied or whenever the PROG_B pin is pulsed Low.

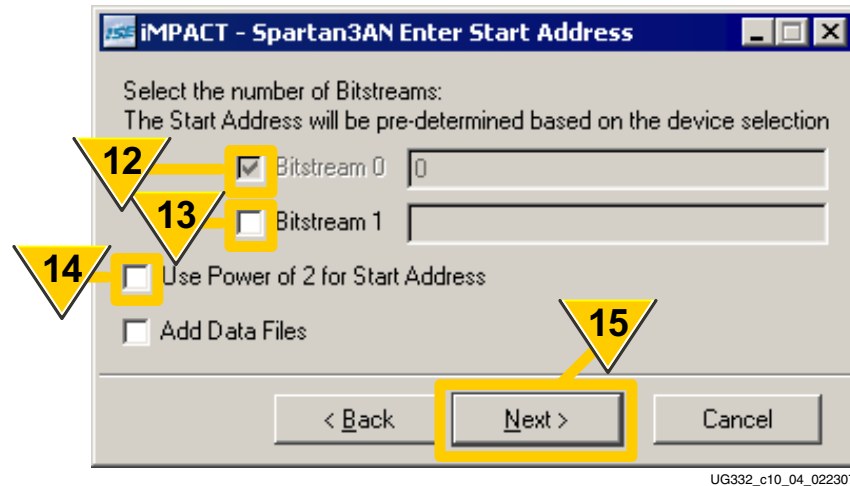


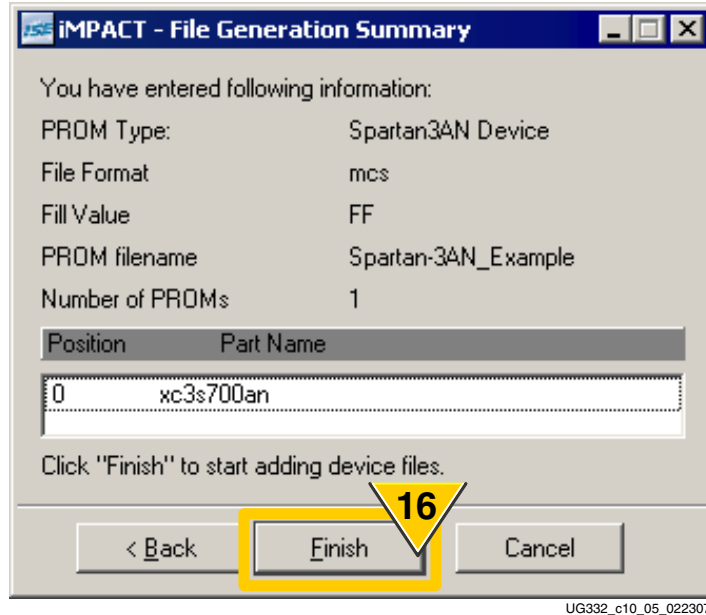
Figure 10-7: Specify the FPGA Configuration Bitstream(s)

13. Click the option box to include a second MultiBoot bitstream (Bitstream 1). Bitstream 1 is always aligned to the next ISF memory sector boundary following Bitstream 0. The iMPACT software displays the sector address in decimal, based on the current addressing mode, as shown in Table 10-3. This is the address used for MultiBoot operations to load the second bitstream.

Table 10-3: Locations of Default Bitstream and Second MultiBoot Bitstream

Bitstream	Spartan-3AN FPGA	ISF Memory Page	Bitstream Starting Address			
			Default		Optional Power-of-2	
			Hex	Byte Address (Decimal)	Hex	Byte Address (Decimal)
Bitstream 0	All	0	0x00_0000	0	0x00_0000	0
Bitstream 1	XC3S50AN	256	0x02_0000	67584	0x01_0000	65536
	XC3S200AN	768	0x06_0000	202752	0x03_0000	196608
	XC3S400AN	1,024	0x08_0000	270336	0x04_0000	262144
	XC3S700AN	1,536	0x0C_0000	405504	0x06_0000	393216
	XC3S1400AN	1,280	0x14_0000	675840	0x0A_0000	655360

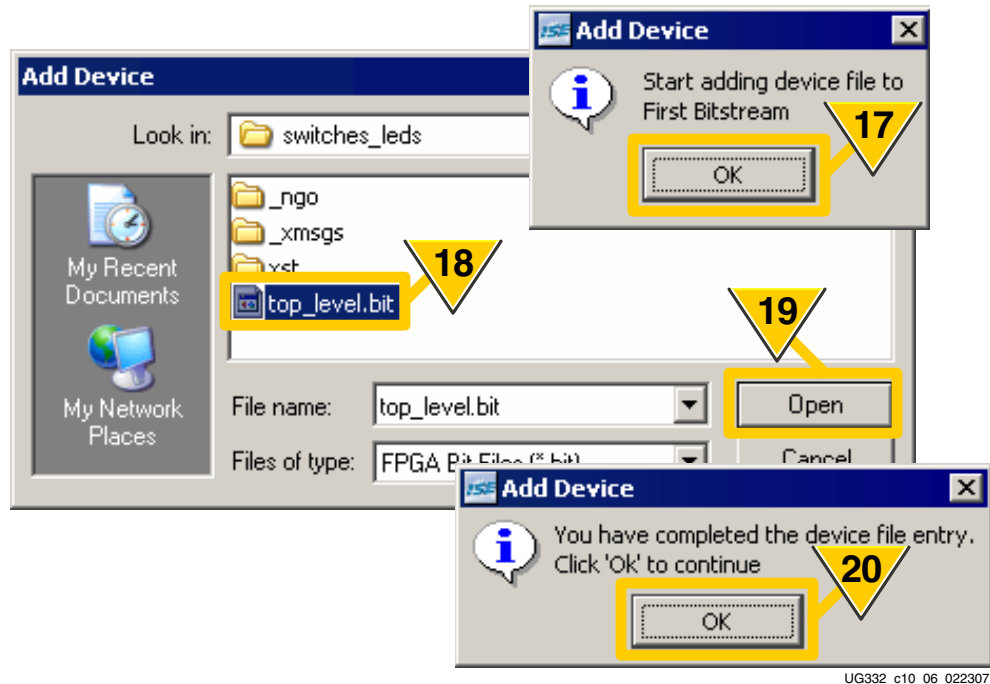
14. **By default, leave this option box unchecked!** Check this box *only* if the intended Spartan-3AN target was previously and specifically programmed to support the optional Power-of-2 addressing mode. See [UG333: Spartan-3AN In-System Flash User Guide](#) for more information.
15. Click Next.
16. As shown in Figure 10-8, review that the settings are correct to format the Spartan-3AN In-System Flash. Click **Finish** to confirm the settings or **Back** to change the settings.



UG332_c10_05_022307

Figure 10-8: Review Spartan-3AN In-System Memory Formatting Settings

- As shown in Figure 10-9, click OK to start adding FPGA configuration bitstreams to the In-System Flash image.



UG332_c10_06_022307

Figure 10-9: Add FPGA Configuration Bitstream File(s)

- Locate and select the desired Spartan-3AN FPGA bitstream.
- Click **Open**.
- If the Bitstream 1 option box was checked in Step 12, the iMPACT software will prompt for a second bitstream. After selecting the last FPGA bitstream, click **OK**.

21. As shown in [Figure 10-10](#), the iMPACT software graphically displays the selected Spartan-3AN FPGA and any associated FPGA bitstream(s).

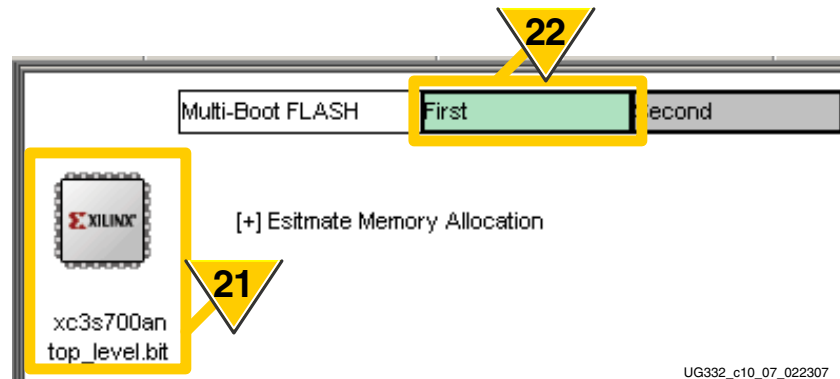


Figure 10-10: iMPACT View of the Spartan-3AN In-System Flash Memory

22. The location of the first and second bitstreams is also highlighted.
 23. As shown [Figure 10-11](#), click **Generate File**.

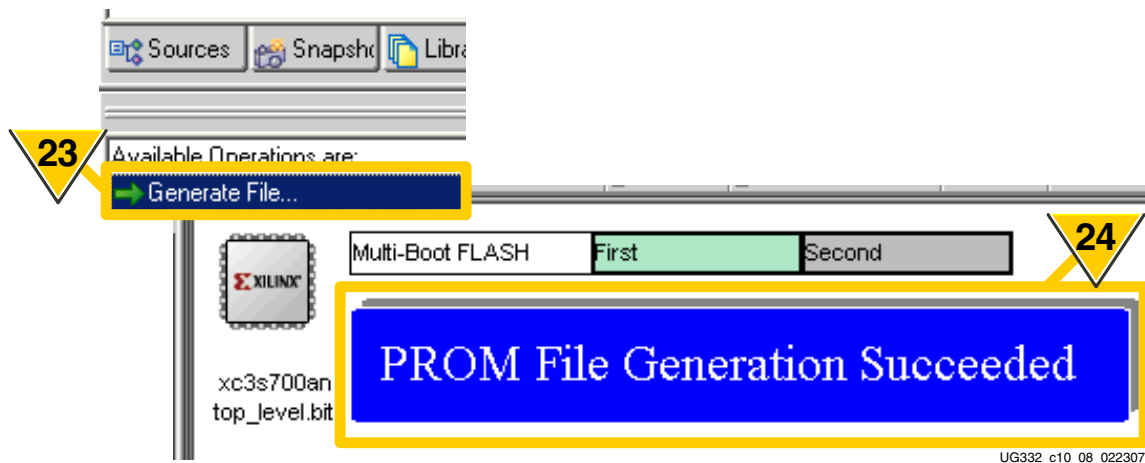


Figure 10-11: Generate the Spartan-3AN In-System Flash File

24. The iMPACT software indicates when the PROM file is successfully created.

PROMGen

PROMGen is a command-line utility that provides an alternate means to create a Spartan-3AN programming file. PROMGen can be invoked from within a command window or from within a script file.

[Table 10-4](#) shows the relevant options for formatting a Spartan-3AN programming file.

Table 10-4: PROM Generator Command Options

PROMGen Option	Description
-spi	REQUIRED! Specifies the correct bit ordering required to configure from the SPI-based In-System Flash memory.
-p <format>	PROM output file format. Specifies the file format required by the SPI programming software. Refer to the third party programmer documentation for details.
-s <size>	Specifies the PROM size in <u>kilobytes</u> . The PROM size must be a power of 2, and the default setting is 64 kilobytes. By default, the Spartan-3AN In-System Flash memory uses a non-binary addressing method, which uses an additional address bit. Use the size settings shown in Table 10-5 .
-u <address>	Loads the <code>.bit</code> file from the specified starting address in an upward direction. This option must be specified immediately before the input bitstream file. See Table 10-3, page 224 for starting addresses by Spartan-3AN FPGA part type.

Table 10-5: Spartan-3AN PROMGen Size Settings

Spartan-3AN FPGA	In-System Flash Size (bits)	-s <size> Setting by Address Mode	
		Default	Power-of-2
XC3S50AN	1M	256	128
XC3S200AN	4M	1,024	512
XC3S400AN	4M	1,024	512
XC3S700AN	8M	2,048	1,024
XC3S1400AN	16M	4,096	2,048

The example PROMGen command, provided below, generates a PROM file for an XC3S700AN FPGA with the following characteristics.

- Formatted for the SPI-based In-System Memory by specifying the `-spi` option.
- Formatted using the Intel MCS format by specifying the `-p mcs` option. The output filename is specified by the `-o <promdata>.mcs` option, where `<promdata>` is a user-specified file name.
- The XC3S700AN In-System Flash memory is only slightly larger than 8M bits or 1,024 bytes. However, set the size option to twice the size, or `-s 2048`, because the default addressing method uses an additional address line. If using the optional power-of-2 addressing mode, which requires an additional, separate special programming step, set the size option to `-s 1024`.
- The first FPGA bitstream (`bitstream0`) is loaded in the upward direction, starting at address 0 by specifying the `-u 0` option. A second MultiBoot bitstream (`bitstream1`) is loaded at the next sector boundary, shown in [Table 10-3, page 224](#), `0x0C_0000` for the XC3S700AN.
- The FPGA bitstreams to be added to the In-System Flash memory are specified as the last option, `<bitstream0>.bit` and `<bitstream1>.bit`, where `<inputfile>` is the user-specified file name used when generating the FPGA bitstream.

```
promgen -spi -p mcs -o <promdata>.mcs -s 2048 -u 0 <bitstream0>.bit ↵
-u c0000 <bitstream1>.bit
```

Programming Spartan-3AN FPGAs Using iMPACT

Beginning in ISE 9.1i, Service Pack 3, the iMPACT software provides programming support for prototyping and initial hardware development. Production programming support is described in “Third-Party Programmer Support,” page 228.

The iMPACT software programs the Spartan-3AN FPGA using the Xilinx programming cables, described in “Programming Cables and Headers,” page 207, using the connections shown in Figure 10-1, page 215.

Third-Party Programmer Support

The Xilinx iMPACT software, starting with ISE 9.1i, Service Pack 3, provides in-system programming support for prototyping and initial development. However, the iMPACT software is not efficient for high-volume production programming. The available Spartan-3AN production programming solutions are listed below by vendor.

BPM Microsystems

BPM Microsystems is a global supplier of engineering and production device programmers and is the leading supplier of automated programming systems to the semiconductor and electronics industries.

- **BPM Microsystems Web Site**
www.bpmicrosystems.com

Production Hardware Programming Solutions

Table 10-6 lists the BPM Microsystems programming solutions for Spartan-3AN FPGAs. Support is available both for new installations and for pre-existing programmers. Socket adapters are required.

Table 10-6: **BPM Microsystems Programmers Supporting Spartan-3AN FPGAs**

Status	Programmer Model Number	Programmer Type
Recommended for new installations, available for purchase	3610	Automated Production
	4610	
	4710	
	3710MK2	Multi-site Concurrent
	BP-2610	
	BP-2710	
	BP-1410	Single-site Engineering
	BP-1610	
	BP-1710	

Table 10-6: BPM Microsystems Programmers Supporting Spartan-3AN FPGAs

Status	Programmer Model Number	Programmer Type
Legacy model. May already be installed in many programming centers	4700	Automated Production
	3700MK2	
	BP-3500	
	BP-3510	
	BP-3600	
	BP-4500	
	BP-4510	
	BP-4600	
	BP-2500	Multi-site Concurrent
	BP-2510	
	BP-2600	
	BP-2700	
	BP-1600	Single-site Engineering
	BP-1700	

Programming Socket Modules and Software

Table 10-7 lists the socket adapters and software required to program Spartan-3AN FPGAs on the programming solutions shown in Table 10-6. Check the BPM Microsystems web site for the most up-to-date information.

Table 10-7: BPM Microsystems Socket Modules and Software for Spartan-3AN FPGAs

Spartan-3AN FPGA	BPM Microsystems Socket Module Model Number	Programming Software
XC3S50AN		
XC3S200AN	ASM256BGT SM256BGT	BPWin V4.66.0 and later
XC3S400AN		
XC3S700AN	ASM484BGD SM484BGD	BPWin V4.66.0 and later
XC3S1400AN	ASM676BG SM676BG	BPWin V4.66.0 and later

Configuration Bitstream Generator (BitGen) Settings

Various Spartan[®]-3 generation FPGA functions are controlled by individual settings in the configuration bitstream image. These values are specified when creating the bitstream image with the Bitstream Generator (BitGen) software.

Table 11-2, page 232 lists the more commonly-used bitstream generator options for Spartan-3 generation FPGAs. Each of these options can be specified on the command line with the following format:

```
bitgen -g <option>:<value> infile
```

The option name and value are separated by a colon with no spaces.

For more information and a complete listing of all options, see the “BitGen” chapter in the following document:

- ISE[®] Software 10.2i Development System Reference Guide
<http://toolbox.xilinx.com/docsan/xilinx10/books/docs/dev/dev.pdf>

For a quick summary of available options for particular FPGA family, type the command shown in Table 11-1 in a DOS box or command window.

Table 11-1: Command Line to Review Bitstream Generator Options per Family

FPGA Family	Command Line
Spartan-3	<code>bitgen -help spartan3</code>
Spartan-3E	<code>bitgen -help spartan3e</code>
Spartan-3A	<code>bitgen -help spartan3a</code>
Spartan-3AN	<code>bitgen -help spartan3an</code>
Spartan-3A DSP	<code>bitgen -help spartan3adsp</code>

Some of the bitstream options can be controlled from the ISE Project Navigator, as described in “ISE Software Project Navigator,” page 42. Any option not specifically listed in the graphic interface can be included as Step 5 shown in Figure 1-6, page 43.

Table 11-2: Spartan-3 Generation Bitstream Generator (BitGen) Options

Option Name	Pins/Function Affected	Values (<i>default</i>)	Description
ConfigRate	CCLK, Configuration, Master Modes only	Extended Spartan-3A FPGA: 1, 3, <u>6</u> , 7, 8, 10, 12, 13, 17, 22, 25, 27, 33, 44, 50, 100	Sets the frequency, approximately in MHz, of the internal oscillator used for Master configuration modes. Drives out on the FPGA's CCLK pin. The internal oscillator powers up at its lowest frequency, and the new setting is loaded as part of the configuration bitstream. See “Configuration Clock: CCLK,” page 56 for more information.
		Spartan-3E FPGA: <u>1</u> , 3, 6, 12, 25, 50	
		Spartan-3 FPGA: 3, <u>6</u> , 12, 25, 50	
StartupClk	Configuration, Startup	Cclk	Default. The CCLK signal (internally or externally generated) controls the Startup sequencer as the FPGA transitions from configuration mode to the application loaded into the FPGA. See “Startup Clock Source,” page 250.
		UserClk	A clock signal from within the FPGA application controls the Startup sequencer as the FPGA transitions from configuration mode to the application loaded into the FPGA. See “Startup Clock Source,” page 250. The FPGA application supplies the user clock on the CLK pin on the STARTUP primitive. See “Start-Up (STARTUP),” page 255.
		JtagClk	The JTAG TCK input controls the startup sequence when the FPGA transitions from the configuration mode to the user mode. See “Startup,” page 248.
ProgPin	PROG_B pin	Pullup	Default. Internally connects a pull-up resistor or between PROG_B pin and V _{CCAUX} . See “Program or Reset FPGA: PROG_B,” page 56.
		Pullnone	No internal pull-up resistor on PROG_B pin. An external 4.7 kΩ pull-up resistor to V _{CCAUX} is required.
UnusedPin	Unused I/O Pins	Pulldown	Default. All unused I/O pins and input-only pins have a pull-down resistor to GND.
		Pullup	All unused I/O pins and input-only pins have a pull-up resistor to the VCCO_# supply for its associated I/O bank.
		Pullnone	All unused I/O pins and input-only pins are left floating (Hi-Z, high-impedance, three-state). Use external pull-up or pull-down resistors or logic to apply a valid signal level.

Table 11-2: Spartan-3 Generation Bitstream Generator (BitGen) Options (Cont'd)

Option Name	Pins/Function Affected	Values (default)	Description
Persist	SelectMAP interface pins, Slave mode, Configuration	<i>No</i>	<i>Default.</i> All Slave mode configuration pins are available as user-I/O after configuration.
		Yes	This option is required for Readback using the SelectMAP interface. The SelectMAP interface pins (see “SelectMAP Data Loading,” page 182) are reserved after configuration and are not available as user-I/O.
Security	JTAG, SelectMAP, Readback	<i>None</i>	<i>Default.</i> Readback is available via the JTAG port or via the SelectMAP interface, if <i>Persist:Yes</i> .
		Level1	See “Basic FPGA Hardware-Level Security Options,” page 289.
		Level2	
		Level3	
Compress	FPGA bitstream size	<i>No</i>	<i>Default.</i> Bitstream is not compressed and will be the size shown in Table 1-4 .
		Yes	Possibly compress the FPGA bitstream by finding redundant configuration frame and using multi-frame write command during configuration. There is no guarantee of the amount of compression. Sparse designs or designs that do not use block RAM see the most benefit. See “Bitstream Format,” page 39.
Spartan-3 FPGA Family Configuration Pin Controls (see Table 2-9, page 63 and Table 2-11, page 64)			
HswapenPin	Spartan-3 FPGA only: HSWAP_EN pin	<i>Pullup</i>	<i>Default.</i> Internally connects a pull-up resistor between the Spartan-3 HSWAP_EN pin and V _{CCAUX} .
		Pulldown	Internally connects a pull-down resistor between the Spartan-3 HSWAP_EN pin and GND.
		Pullnone	No internal pull-up resistor on the Spartan-3 HSWAP_EN pin.
CclkPin	Spartan-3 FPGA only: CCLK pin	<i>Pullup</i>	<i>Default.</i> Internally connects a pull-up resistor or between CCLK pin and V _{CCAUX} .
		Pullnone	CCLK pin is high-impedance (floating). Define CCLK logic level externally.
M2Pin	Spartan-3 FPGA only: M2 pin	<i>Pullup</i>	<i>Default.</i> Internally connects a pull-up resistor or between M2 mode-select pin and V _{CCAUX} .
		Pulldown	Internally connects a pull-down resistor or between M2 mode-select pin and GND.
		Pullnone	M2 pin is high-impedance (floating). Define M2 logic level externally.

Table 11-2: Spartan-3 Generation Bitstream Generator (BitGen) Options (Cont'd)

Option Name	Pins/Function Affected	Values (default)	Description
M1Pin	Spartan-3 FPGA only: M1 pin	<i>Pullup</i>	<i>Default.</i> Internally connects a pull-up resistor or between M1 mode-select pin and V_{CCAUX} .
		Pulldown	Internally connects a pull-down resistor or between M1 mode-select pin and GND.
		Pullnone	M1 pin is high-impedance (floating). Define M1 logic level externally.
M0Pin	Spartan-3 FPGA only: M0 pin	<i>Pullup</i>	<i>Default.</i> Internally connects a pull-up resistor or between M0 mode-select pin and V_{CCAUX} .
		Pulldown	Internally connects a pull-down resistor or between M0 mode-select pin and GND.
		Pullnone	M0 pin is high-impedance (floating). Define M0 logic level externally.
DONE Pin Options			
See "DONE Pin," page 52.			
DonePin	DONE pin	<i>Pullup</i>	<i>Default.</i> Internally connects a pull-up resistor between DONE pin and V_{CCAUX} . An external 330 Ω pull-up resistor to V_{CCAUX} is still recommended. See DONE pin "ConfigRate: Bitstream Option for CCLK," page 60.
		Pullnone	No internal pull-up resistor on DONE pin. An external 330 Ω pull-up resistor to V_{CCAUX} is required.
DriveDone	DONE pin	<i>No</i>	<i>Default.</i> When configuration completes, the DONE pin stops driving Low and relies on an external 330 Ω pull-up resistor to V_{CCAUX} for a valid logic High. See DONE pin "ConfigRate: Bitstream Option for CCLK," page 60.
		Yes	When configuration completes, the DONE pin actively drives High. When using this option, an external pull-up resistor is no longer required. Only one device in an FPGA daisy chain should use this setting.
DonePipe	DONE pin	<i>No</i>	<i>Default.</i> The input path from DONE pin input back to the Startup sequencer is not pipelined. See DONE pin "ConfigRate: Bitstream Option for CCLK," page 60.
		Yes	This option adds a pipeline register stage between the DONE pin input and the Startup sequencer. Used for high-speed daisy-chain configurations when DONE cannot rise in a single CCLK cycle. Releases GWE and GTS signals on the first rising edge of StartupClk after the DONE pin input goes High.
Startup Sequencer Options			
See "Startup," page 248.			
DONE_cycle	DONE pin, Configuration Startup	1, 2, 3, <u>4</u> , 5, 6	Selects the Configuration Startup phase that activates the FPGA's DONE pin. See "Startup," page 248.

Table 11-2: Spartan-3 Generation Bitstream Generator (BitGen) Options (Cont'd)

Option Name	Pins/Function Affected	Values (default)	Description
GWE_cycle	All flip-flops, LUT RAMs, and SRL16 shift registers, Block RAM, Configuration Startup	1, 2, 3, 4, 5, <u>6</u>	<i>Default.</i> Selects the Configuration Startup phase that asserts the internal write-enable signal to all flip-flops, LUT RAMs and shift registers (SRL16). It also enables block RAM read and write operations. See “Startup,” page 248.
		Done	Waits for the DONE pin input to go High before asserting the internal write-enable signal to all flip-flops, LUT RAMs and shift registers (SRL16). Block RAM read and write operations are enabled at this time.
GTS_cycle	All I/O pins, Configuration	1, 2, 3, 4, <u>5</u> , 6	<i>Default.</i> Selects the Configuration Startup phase that releases the internal three-state control, holding all I/O buffers in high-impedance (Hi-Z). Output buffers actively drive, if so configured, after this point. See “Startup,” page 248.
		Done	Waits for the DONE pin input to go High before releasing the internal three-state control, holding all I/O buffers in high-impedance (Hi-Z). Output buffers actively drive, if so configured, after this point.
		Keep	Retains the current GTS_cycle setting.
LCK_cycle	DCMs, Configuration Startup	<i>NoWait</i>	<i>Default.</i> The FPGA does not wait for selected DCMs to lock before completing configuration.
		0, 1, 2, 3, 4, 5, 6	If one or more DCMs in the design have the STARTUP_WAIT=TRUE attribute, the FPGA waits for such DCMs to acquire their respective input clock and assert their LOCKED output. This setting selects the Configuration Startup phase where the FPGA waits for the DCMs to lock. See “Waiting for DCMs to Lock, DCI to Match,” page 250.
Match_cycle	Spartan-3 FPGA only: DCI	<i>Auto</i>	The BitGen software examines the FPGA design for any I/O standards that use DCI. If found, BitGen automatically sets <i>Match_cycle:2</i> , causing the Startup sequence to stall in state 2 while the DCI circuitry matches the target impedance. Otherwise, <i>Match_cycle:NoWait</i> .
		NoWait	The FPGA does not wait for DCI circuitry to match impedance.
		0, 1, 2, 3, 4, 5, 6	Specify the Startup cycle where the FPGA waits for the DCI circuitry to match the target impedance value, specified using external resistors.
DCIUpdateMode	Spartan-3 FPGA only: DCI	<i>AsRequired</i>	<i>Default.</i> DCI impedance adjustments are made only when needed to maintain tracking.
		Continuous	DCI impedance adjustments are made continuously.
		Quiet	After the initial DCI impedance match is achieved, no further adjustments occur.
JTAG-Related Options			
See Chapter 9, “JTAG Configuration Mode and Boundary-Scan.”			

Table 11-2: Spartan-3 Generation Bitstream Generator (BitGen) Options (Cont'd)

Option Name	Pins/Function Affected	Values (default)	Description
TckPin	JTAG TCK pin	<i>Pullup</i>	Default. Internally connects a pull-up resistor between JTAG TCK pin and V _{CCAUX} .
		Pulldown	Internally connects a pull-down resistor between JTAG TCK pin and GND.
		Pullnone	No internal pull-up resistor on JTAG TCK pin.
TdiPin	JTAG TDI pin	<i>Pullup</i>	Default. Internally connects a pull-up resistor between JTAG TDI pin and V _{CCAUX} .
		Pulldown	Internally connects a pull-down resistor between JTAG TDI pin and GND.
		Pullnone	No internal pull-up resistor on JTAG TDI pin.
TdoPin	JTAG TDO pin	<i>Pullup</i>	Default. Internally connects a pull-up resistor between JTAG TDO pin and V _{CCAUX} .
		Pulldown	Internally connects a pull-down resistor between JTAG TDO pin and GND.
		Pullnone	No internal pull-up resistor on JTAG TDO pin.
TmsPin	JTAG TMS pin	<i>Pullup</i>	Default. Internally connects a pull-up resistor between JTAG TMS pin and V _{CCAUX} .
		Pulldown	Internally connects a pull-down resistor between JTAG TMS pin and GND.
		Pullnone	No internal pull-up resistor on JTAG TMS pin.
UserID	JTAG User ID register	0xFFFFFFFF	The 32-bit JTAG User ID register value is loaded during configuration. The default value is all ones, 0xFFFFFFFF hexadecimal. To specify another value, enter an 8-character hexadecimal value.
Extended Spartan-3A Family Power-Saving Suspend Feature			
See XAPP480 Using Suspend Mode in Spartan-3 Generation FPGAs.			
en_suspend	Extended Spartan-3A FPGA only: Suspend mode	No	Default. Suspend mode not used. Connect the SUSPEND pin to GND.
		Yes	Enables the power-saving Suspend feature, controlled by the SUSPEND pin.
drive_awake	Extended Spartan-3A FPGA only: Suspend mode, AWAKE pin	No	Default. If Suspend mode is enabled, indicates the present status on AWAKE using an open-drain output. An external pull-up resistor or High signal is required to exit SUSPEND mode.
		Yes	If Suspend mode is enabled, indicates the present status by actively driving the AWAKE output.

Table 11-2: Spartan-3 Generation Bitstream Generator (BitGen) Options (Cont'd)

Option Name	Pins/Function Affected	Values (default)	Description
suspend_filter	Extended Spartan-3A FPGA only: Suspend mode, SUSPEND pin	<i>Yes</i>	<i>Default.</i> Enables the glitch filter on the SUSPEND pin.
		No	Disables the glitch filter on the SUSPEND pin.
en_sw_gsr	Extended Spartan-3A FPGA only: Suspend mode, wake-up timing	<i>No</i>	<i>Default.</i> The state of all clocked elements in the FPGA is preserved during Suspend mode.
		Yes	During wake-up from Suspend mode, the FPGA pulses the Global Set/Reset (GSR) signal, setting or resetting all clocked elements, as originally specified in the FPGA application. All state information prior to entering Suspend mode is lost. The GSR pulse occurs before the AWAKE pin goes High and before the <i>sw_gwe_cycle</i> and <i>sw_gts_cycle</i> settings are active.
sw_clk	Extended Spartan-3A FPGA only: Suspend mode, wake-up timing	<i>StartupClk</i>	<i>Default.</i> Uses the clock defined by the <i>StartupClk</i> bitstream setting to control the Suspend wake-up timing.
		InternalClk	Uses the internally generated "50 MHz" oscillator to control the Suspend wake-up timing. The clock frequency is the same as when <i>ConfigRate:50</i> , as described in the FPGA data sheet.
sw_gwe_cycle	Extended Spartan-3A FPGA only: Suspend mode, wake-up timing	1,...,5,...,1024	After the AWAKE pin is High, indicates the number of clock cycles as defined by the <i>sw_clk</i> setting, when the global write-protect lock is released for writable clocked elements (flip-flops, block RAM, etc.). The default value is five clock cycles after the AWAKE pin goes High. Generally, this value is equal to or greater than the <i>sw_gts_cycle</i> setting.
sw_gts_cycle	Extended Spartan-3A FPGA only: Suspend mode, wake-up timing	1,...,4,...,1024	After the AWAKE pin is High, indicates the number of clock cycles as defined by the <i>sw_clk</i> setting, when the I/O pins switch from their SUSPEND Constraint settings back to their normal functions. The default value is four clock cycles after the AWAKE pin goes High. Generally, this value is equal to or less than the <i>sw_gwe_cycle</i> setting.
Extended Spartan-3A Family MultiBoot Control Options			
See "Extended Spartan-3A Family MultiBoot," page 271.			
ICAP_Enable	Extended Spartan-3A FPGA only: ICAP, MultiBoot	<i>Auto</i>	<i>Default.</i> The BitGen software examines the FPGA design. If the ICAP primitive is instantiated in the design, BitGen automatically sets <i>ICAP_Enable:Yes</i> , enabling the ICAP port. Otherwise, <i>ICAP_Enable:No</i> .
		No	The ICAP port is disabled.
		Yes	The ICAP port is enabled.
next_config_addr	Extended Spartan-3A FPGA only: MultiBoot	0x0000000	Specifies the next MultiBoot start address as a 7-character hexadecimal value. The specified value is loaded into the <i>GENERAL1</i> and <i>GENERAL2</i> registers during configuration. See "Extended Spartan-3A Family MultiBoot," page 271 for details on use.

Table 11-2: Spartan-3 Generation Bitstream Generator (BitGen) Options (Cont'd)

Option Name	Pins/Function Affected	Values (default)	Description
Configuration CRC Checking Options			
See Chapter 16, "Configuration CRC."			
CRC	Configuration	<u>Enable</u>	<i>Default.</i> Enable CRC checking on the FPGA bitstream. If error detected, FPGA asserts INIT_B Low and DONE pin stays Low. In the Extended Spartan-3A family, Reset_on_err can be enabled to re-attempt configuration instead.
		Disable	Turn off CRC checking. Not recommended.
Reset_on_err	Extended Spartan-3A FPGA only: MultiBoot, CRC, watchdog timer	<u>No</u>	<i>Default.</i> The FPGA halts upon encountering a configuration CRC error.
		Yes	If a configuration CRC error occurs, the FPGA automatically re-initializes and retries the configuration process. Three retry attempts may occur before finally halting. Requires that CRC be enabled (default).
post_crc_en	Extended Spartan-3A FPGA only: Post-configuration CRC checker	<u>No</u>	<i>Default.</i> Disable the post-configuration CRC checker.
		Yes	Enable the post-configuration CRC checker.
post_crc_freq	Extended Spartan-3A FPGA only: Post-configuration CRC checker	1, 3, <u>6</u> , 7, 8, 10, 12, 13, 17, 22, 25, 27, 33, 44, 50, 100	Sets the clock frequency used for the post-configuration CRC checker. The available options are the same as for the ConfigRate bitstream option.
post_crc_keep	Extended Spartan-3A FPGA only: Post-configuration CRC checker	<u>No</u>	<i>Default.</i> Stop checking when error detected. Allows CRC signature to be read back.
		Yes	Continue to check for CRC errors after an error was detected.
glutmask	Extended Spartan-3A FPGA only: Post-configuration CRC checker	<u>Yes</u>	<i>Default.</i> Mask out the Look-Up Table (LUT) bits from the SLICEM logic slices. SLICEMs support writable functions such as distributed RAM and SRL16 shift registers, which generate CRC errors when bit locations are modified.
		No	Include the LUT bits from SLICEM logic slices. Use this option only if the application does not include any distributed RAM or SRL16 shift registers.

Sequence of Events

Overview

This chapter outlines the multi-stage configuration process for Spartan[®]-3 generation FPGAs.

While each FPGA configuration mode uses a slightly different interface, the basic steps involved are the same for all modes. Figure 12-1 shows the general Spartan-3 generation FPGA configuration process. The details of the bitstream will also include the formatting and command bits. The following subsections describe each step in detail, where the current step is highlighted at the beginning of each subsection.

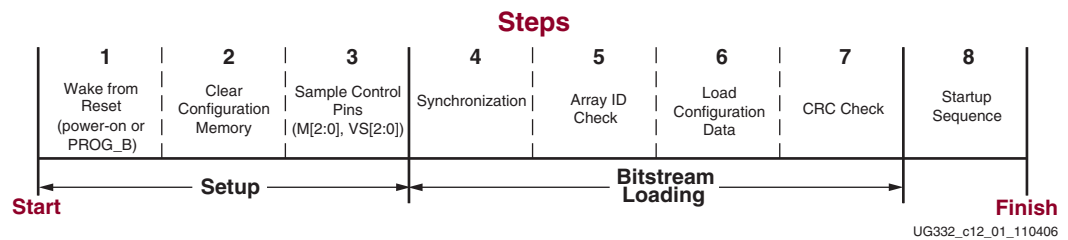


Figure 12-1: Spartan-3 Generation FPGA Configuration Process

Setup for Configuration (Steps 1-3)

The Setup process is similar for all configuration modes. The Spartan-3 generation FPGA first wakes from reset, initializes its internal configuration memory, and determines which configuration mode to use by sampling the mode pins.

Wake from Reset

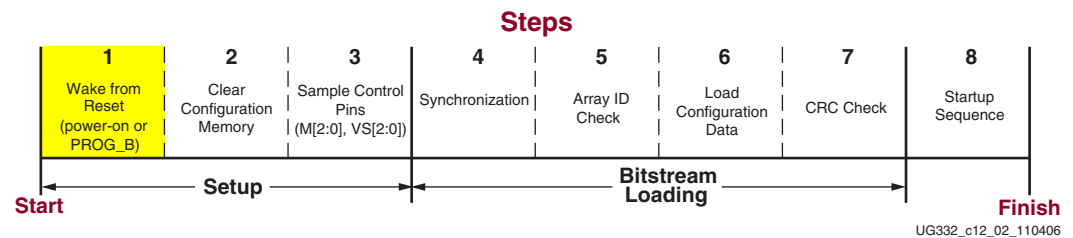


Figure 12-2: FPGA Wake from Reset

Spartan-3 generation FPGAs wake from reset in several possible ways.

1. The FPGA powers on and the FPGA's internal Power-On Reset (POR) circuit holds the FPGA in reset until the required voltage supplies reach appropriate levels.
2. The system pulses the **PROG_B** pin Low, which resets the FPGA.
3. The FPGA is reset via the dedicated JTAG interface using the **JPROGRAM** instruction.
4. The FPGA is reset via the Extended Spartan-3A family **REBOOT** command available using the SelectMAP, JTAG, or ICAP interfaces.
5. On Extended Spartan-3A family FPGAs, the FPGA is reset if the Configuration Watchdog Timer (CWDT) expires during configuration and less than three configuration retries have occurred.

Power-On Reset (POR)

As shown in [Figure 12-3](#), Spartan-3 generation FPGAs include a Power-On Reset (POR) circuit that holds the FPGA in reset until all of the supply rails required for configuration have reached their threshold levels. The three supplies required are the following.

1. V_{CCINT} , which supplies the internal FPGA core logic.
2. V_{CCAUX} , which supplies the dedicated configuration pins.
3. V_{CCO_2} on Extended Spartan-3A family and Spartan-3E FPGAs or V_{CCO_4} (or V_{CCO_BOTTOM} in some packages where V_{CCO_4} and V_{CCO_5} are connected) on Spartan-3 FPGAs, which supplies the interface pins connected to the external configuration data source (i.e., PROM or processor).

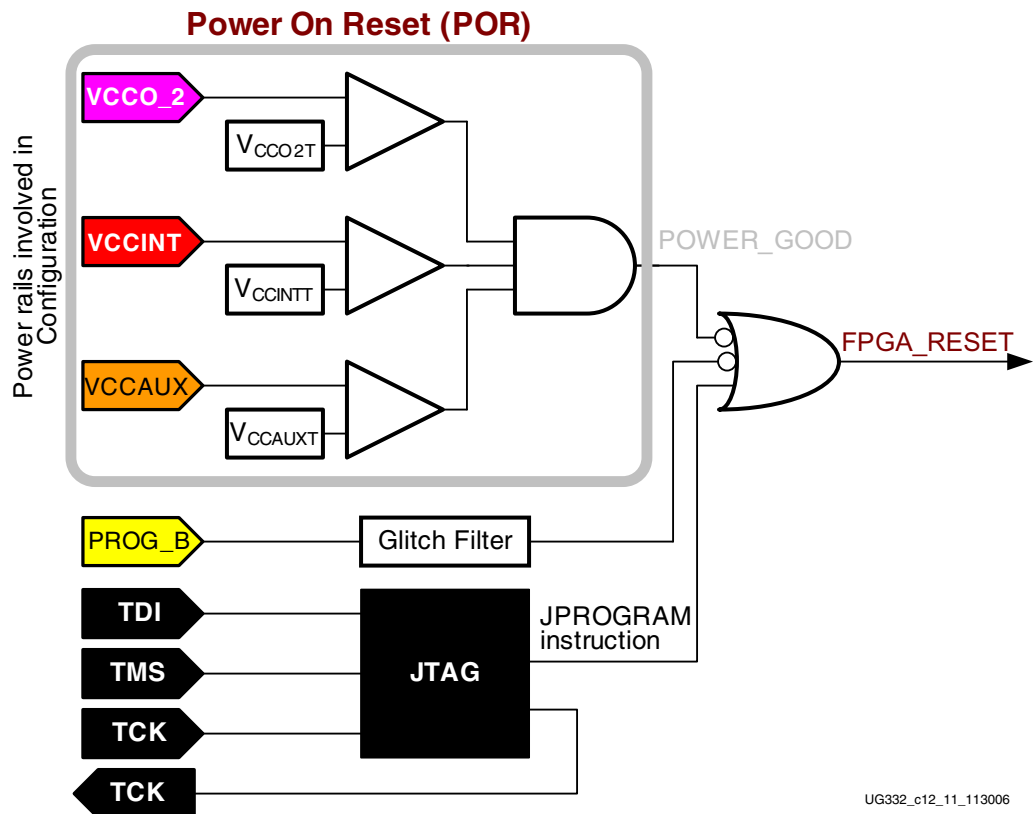


Figure 12-3: Extended Spartan-3A Family and Spartan-3E Reset Circuitry (Spartan-3 is similar)

The FPGA monitors all three supplies. Once all three supplies exceed the specified threshold voltage, summarized in [Table 12-1, page 241](#) from the associated FPGA data sheet, the POR circuit releases the internal reset and the FPGA can continue with the configuration process unless the [PROG_B](#) pin is Low.

Table 12-1: Power-On Reset Threshold Voltages

Voltage Supply	POR Threshold Specification	Spartan-3A/3AN Spartan-3A DSP FPGA		Spartan-3E FPGA		Spartan-3 FPGA		Units
		Min	Max	Min	Max	Min	Max	
V _{CCINT}	V _{CCINTT}	0.4	1.0	0.4	1.0	0.4	1.0	V
V _{CCAUX}	V _{CCAUXT}	1.0	2.0	0.8	2.0	0.8	2.0	V
V _{CCO_2}	V _{CCO2T}	1.0	2.0	0.4	1.0			V
V _{CCO_4} (or "V _{CCO_BOTTOM} ")	V _{CCO4T}					0.4	1.0	V

V_{CCINT} should rise monotonically within the specified ramp rate. If this is not possible, delay configuration by holding the [INIT_B](#) pin or the [PROG_B](#) pin Low (see "[Delaying Configuration,](#)" [page 243](#)) while the system power supplies reach the required POR threshold.

After successfully configuring, the POR circuit continues to monitor the V_{CCINT} and V_{CCAUX} supply inputs. Should either supply drop below the its associated threshold voltage, the POR circuit again resets the FPGA. V_{CCO_2} is not monitored after configuration so that the user can reduce it as needed for low-voltage standards.

Note that the Extended Spartan-3A family has a requirement that V_{CCO_2} reach 2.0V for successful power-on. This level is higher than in the Spartan-3 and Spartan-3E FPGAs, and helps make sure that external configuration memories are ready before the FPGA starts attempting access. If the design only requires 1.8V (or lower) I/Os in bank 2 then the V_{CCO_2} supply would temporarily need to rise to 2.0V and then could drop down to the 1.8V level for operation. See "Lowering V_{CCO_2} After Configuration for Extended Spartan-3A Family" in chapter 2.

PROG_B Pin

The [PROG_B](#) resets the FPGA, regardless of the current state of the FPGA. For additional information, see "[Program or Reset FPGA: PROG_B,](#)" [page 56](#).

Power-Up Timing

[Figure 12-4](#) shows the general power-up timing, showing the relationship between the input voltage supplies, the [INIT_B](#) pin, and the [PROG_B](#) pin.

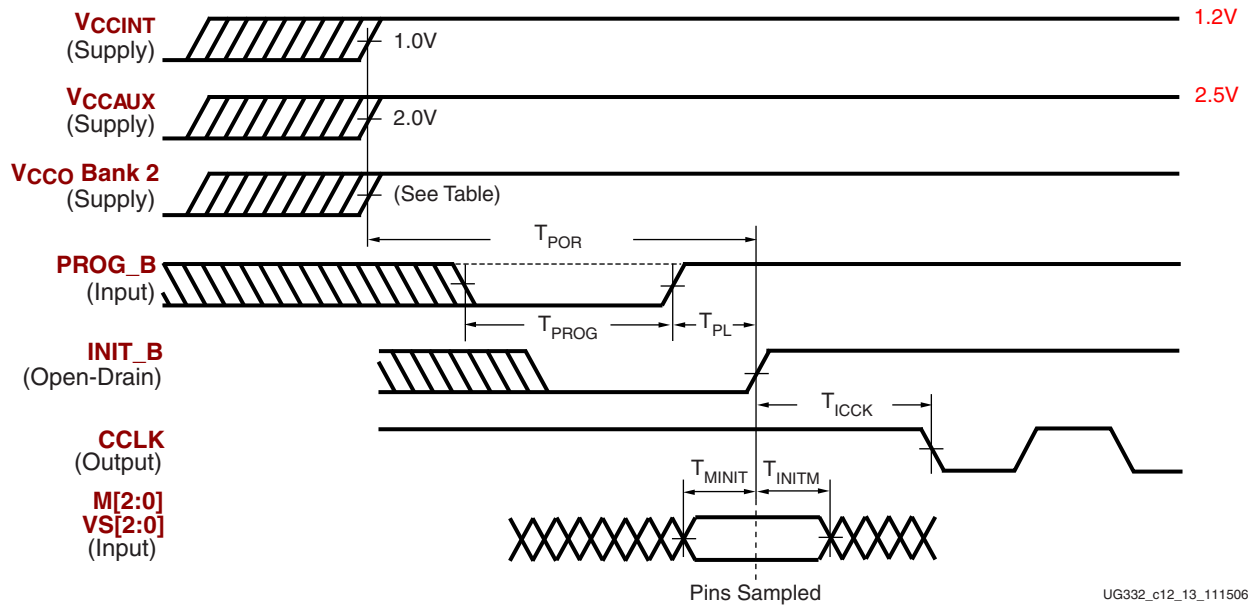


Figure 12-4: FPGA Power-Up Timing Waveforms (Master Modes)

Table 12-2 lists and describes the power-up timing specifications shown in Figure 12-4. Refer to the associated FPGA data sheet for any unlisted values.

Table 12-2: FPGA Power-Up Timing Specifications

Symbol	Description	Family	Value	Units
T_{POR}	Power-On Reset delay from when all three supplies reach their required threshold voltage until the FPGA completes clearing its configuration memory and INIT_B goes High.	Spartan-3	5 to 7	ms
		Spartan-3E	5 to 7	
		Spartan-3A	18	
T_{PL}	Delay from when PROG_B is released High until the FPGA completes clearing its configuration memory and INIT_B goes High.	Spartan-3	2 to 3	ms
		Spartan-3E	0.5 to 2	
		Spartan-3A	0.5 to 2	
T_{PROG}	Minimum PROG_B pulse width required to reset FPGA.	Spartan-3	300	ns
		Spartan-3E	500	
		Spartan-3A	500	
T_{ICCK}	For Master configuration modes, the time from the rising edge of INIT_B until CCLK output begins toggling.	All	0.5 to 4	μ s
T_{MINIT}	Setup time on M[2:0] mode-select pins and, in Master SPI mode, the setup time on VS[2:0] variant-select pins before the rising edge of INIT_B .	All	50	ns

Notes:

1. Spartan-3A represents the Spartan-3A, Spartan-3AN, and Spartan-3A DSP FPGA families.

Clear Configuration Memory (Initialization)

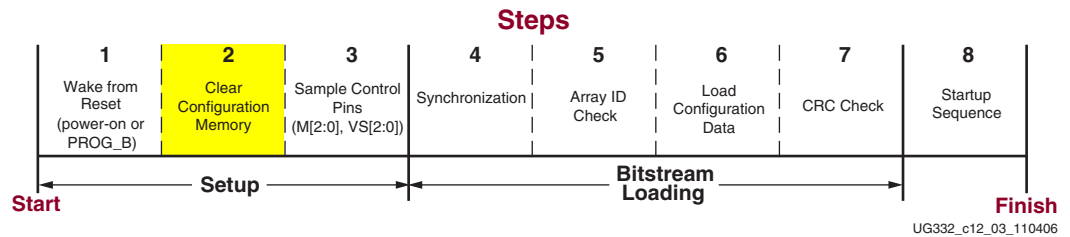


Figure 12-5: Clear Configuration Memory (Initialization)

Configuration memory is cleared automatically after the FPGA wakes from a reset event. During this time, I/Os are placed in a high-impedance (Hi-Z) state except for the dedicated Configuration and JTAG pins. The **INIT_B** pin actively drives Low during initialization, and then released after T_{POR} during a power-up event or after T_{PL} for other cases. See Figure 12-4. If the **INIT_B** pin is held Low externally, the FPGA waits at this point in the initialization process until the pin is released.

The minimum Low pulse time for **PROG_B** is defined by the T_{PROG} timing parameter. The **PROG_B** pin can be held active (Low) for as long as necessary.

Sample Control Pins

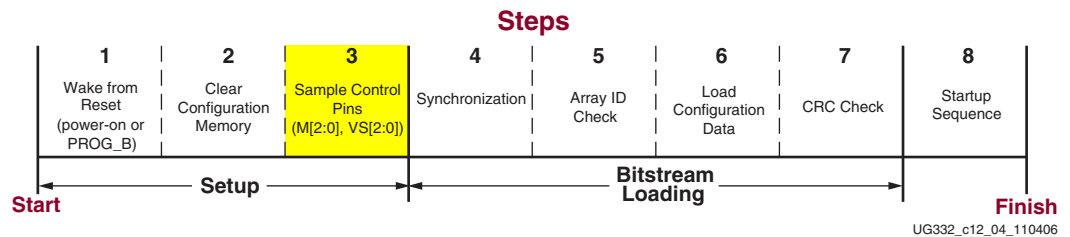


Figure 12-6: Sample Control Pins (Mode Select, Variant Select)

When the **INIT_B** pin returns High after initialization, the FPGA samples the **M[2:0]** mode select pins and the **VS[2:0]** variant select pins. Shortly after, the FPGA begins driving **CCLK** if the **M[2:0]** mode select pins define one of the Master configuration modes. The **VS[2:0]** values are only used in Master SPI configuration mode. At this point, the FPGA begins sampling the configuration data input pins on the rising edge of the configuration clock.

Delaying Configuration

There are three methods to delay configuration for Spartan-3 generation FPGAs.

1. Hold the **PROG_B** pin Low, which holds the FPGA in reset, Step 1 shown in Figure 12-2, page 239.
2. Hold the **INIT_B** pin Low during initialization, which stalls the configuration process in Step 2 shown in Figure 12-5, page 243. However, after the FPGA releases **INIT_B** High, the application cannot subsequently delay configuration by pulling **INIT_B** Low.
3. Hold the **DONE** pin Low, which prevents the FPGA from completing the Startup Sequence, shown as Step 8 in Figure 12-11, page 248.

Bitstream Loading (Steps 4-7)

The bitstream loading process is similar for all configuration modes; the primary difference between modes is the interface between the FPGA and the source of configuration data.

The important steps in the bitstream loading process are as follows.

- Synchronization
- Array ID check
- Loading configuration data
- CRC check

Each of these steps involves distinct parts of the configuration bitstream.

Synchronization

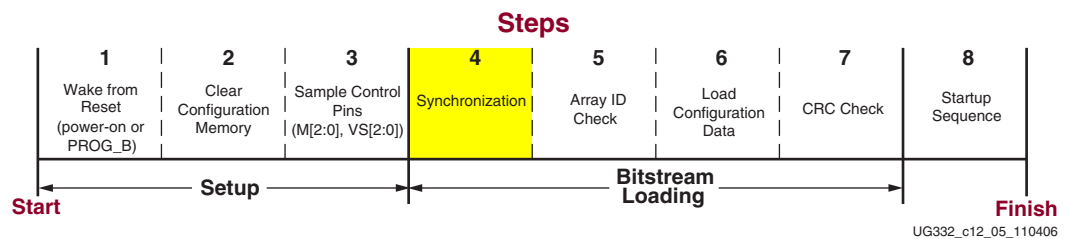


Figure 12-7: Synchronization

Embedded at the beginning of an FPGA configuration bitstream is a special *synchronization word*. The synchronization word alerts the FPGA to upcoming configuration data and aligns the configuration data with the internal configuration logic. Any data on the configuration input pins prior to synchronization is ignored. Because the synchronization word is automatically added by the Xilinx® bitstream generation software, this step is transparent in most applications.

The length and contents of the synchronization word differ between the Extended Spartan-3A family FPGA families and the Spartan-3 and Spartan-3E FPGA families, as outlined in [Table 12-3](#).

Table 12-3: Spartan-3 Generation FPGA Synchronization Word

FPGA Family	Length (bits)	Contents (hexadecimal)
Spartan-3A/3AN Spartan-3A DSP	16	0xAA99
Spartan-3 Spartan-3E	32	0xAA995566

Check Array IDCODE

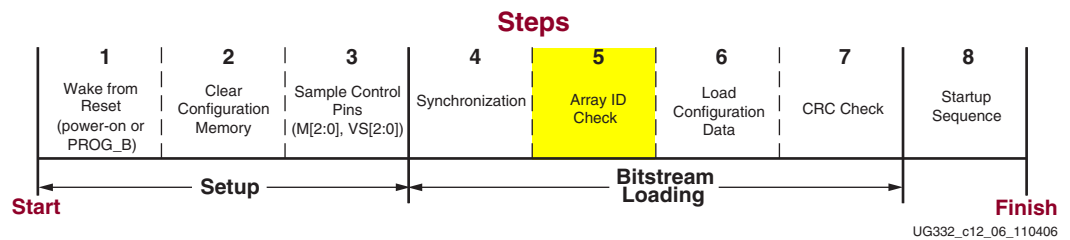


Figure 12-8: Check Array ID

After the FPGA is synchronized, the FPGA checks that the array ID embedded in the bitstream matches its internal array ID. This prevents the FPGA from mistakenly attempting to load configuration data intended for a different FPGA array. For example, the array ID check prevents an XC3S1000 from being configured with an XC3S200 bitstream.

The Spartan-3AN FPGA family can be configured with a Spartan-3A bitstream for the equivalent size device, since they are compatible.

The array ID check is built into the bitstream, making this step transparent to most designers. Table 12-4 shows the Spartan-3 generation array ID codes. Although the array ID code is identical to the JTAG IDCODE register value, the array ID check is performed using bitstream commands to the internal configuration loading logic, not through the JTAG IDCODE register.

The array identifier is a 32-bit value. Within the 32-bit value, 28 bits are unique to a specific FPGA array size while the additional four bits are a mask revision code, which varies between 0x0 to 0xF.

There are three components to the 28-bit vendor/array identifier value.

- The least-significant 12 bits, 0x093, represent the Xilinx vendor code (0x49), appended to the least-significant bit which is always '1', resulting in the value 0x093. These 12 bits are consistent for all Spartan-3 generation FPGAs.
- The most-significant 8 bits represent the FPGA family code.
 - ◆ 0x22: Spartan-3A family
 - ◆ 0x26: Spartan-3AN family
 - ◆ 0x38: Spartan-3A DSP family
 - ◆ 0x1C: Spartan-3E family
 - ◆ 0x14: Spartan-3 family
- The middle 8 bits represent an array-specific code.

Table 12-4: Spartan-3 Generation FPGA Array ID Codes

FPGA Family	FPGA Array	32-bit Array Identifier	
		4-bit Revision Code	28-bit Vendor/Array Identifier (hexadecimal)
Spartan-3A FPGAs	XC3S50A	0xX	0x22 10 093
	XC3S200A	0xX	0x22 18 093
	XC3S400A	0xX	0x22 20 093
	XC3S700A	0xX	0x22 28 093
	XC3S1400A	0xX	0x22 30 093
Spartan-3AN FPGAs	XC3S50AN	0xX	0x26 10 093
	XC3S200AN	0xX	0x26 18 093
	XC3S400AN	0xX	0x26 20 093
	XC3S700AN	0xX	0x26 28 093
	XC3S1400AN	0xX	0x26 30 093
Spartan-3A DSP FPGAs	XC3SD1800A	0xX	0x38 40 093
	XC3SD3400A	0xX	0x38 4E 093
Spartan-3E FPGAs	XC3S100E	0xX	0x1C 10 093
	XC3S250E	0xX	0x1C 1A 093
	XC3S500E	0xX	0x1C 22 093
	XC3S1200E	0xX	0x1C 2E 093
	XC3S1600E	0xX	0x1C 3A 093
Spartan-3 FPGAs	XC3S50	0xX	0x14 0C 093
	XC3S200	0xX	0x14 14 093
	XC3S400	0xX	0x14 1C 093
	XC3S1000	0xX	0x14 28 093
	XC3S1500	0xX	0x14 34 093
	XC3S2000	0xX	0x14 40 093
	XC3S4000	0xX	0x14 48 093
	XC3S5000	0xX	0x14 50 093

The FPGA indicates if the array value does not match by setting Bit 1 (**ID_Err**) in the STAT (Status) register, as shown in [Table 12-5](#). There are various methods to read the status register, including via JTAG using the Xilinx iMPACT software, or by using the SelectMAP interface.

Table 12-5: STAT Register

Name	Bit	Description
ID_Err	1	0: Array ID value matched expected value. 1: Array ID value embedded in bitstream does not match the value read from the FPGA

Load Configuration Data Frames

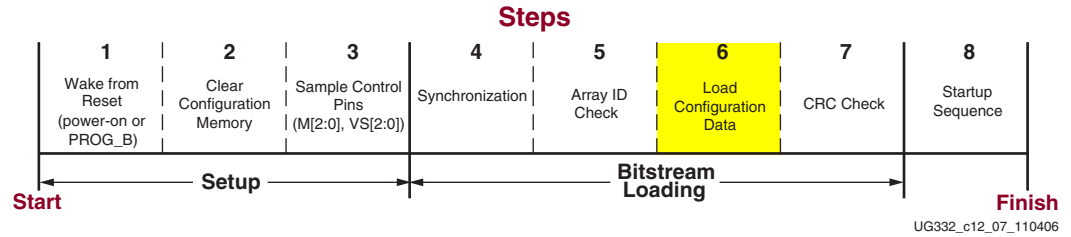


Figure 12-9: Load Configuration Data Frames

After the synchronization word is loaded and the array ID is checked, the configuration data frames are loaded.

Cyclic Redundancy Check

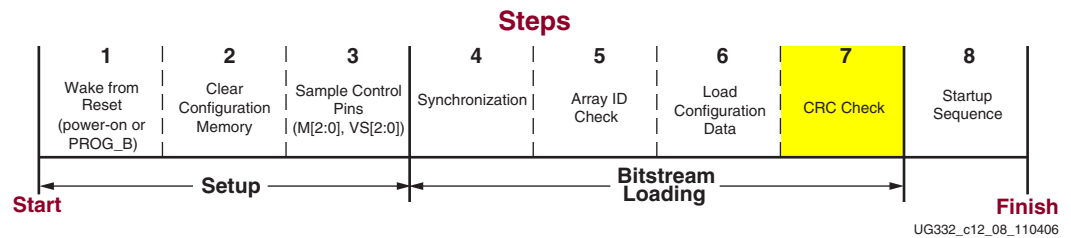


Figure 12-10: Cyclic Redundancy Check

As the configuration data frames are loaded, the FPGA calculates a Cyclic Redundancy Check (CRC) value from the configuration data packets. After the configuration data frames are loaded, the configuration bitstream, by default (*CRC:Enable*), issues a Check CRC instruction to the FPGA, followed by an expected CRC value. If the CRC value calculated by the FPGA does not match the expected CRC value in the bitstream, then the FPGA pulls *INIT_B* Low and aborts configuration.

Refer to “CRC Checking during Configuration,” page 309 for additional information.

Startup

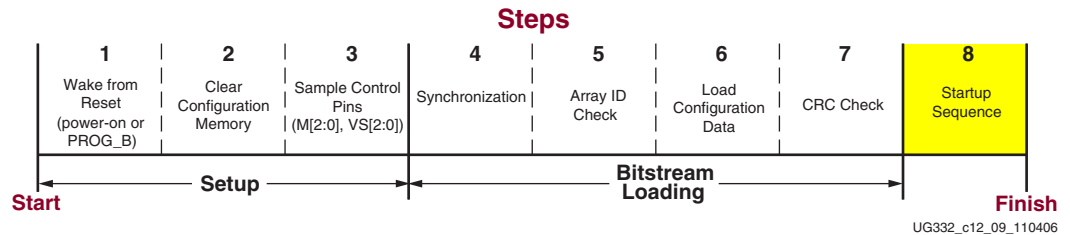


Figure 12-11: Startup Sequence

After successfully loading the configuration frames, the bitstream instructs the FPGA to enter the Startup sequence. The Startup sequence is controlled by an 8-phase (phases 0-7) sequential state machine. The startup sequencer performs the tasks outlined in Table 12-6.

Table 12-6: User-Selectable Cycle of Startup Events

Startup Event	Phase	BitGen Control
Wait for DCMs to Lock (optional)	1-6	<i>LCK_cycle</i>
Spartan-3 FPGA family only: Wait for DCI to Match (optional).	1-6	<i>Match_cycle</i>
Assert Global Write Enable (GWE), allowing RAMs and flip-flops to change state	1-6	<i>GWE_cycle</i>
Release the Global 3-State (GTS), activating I/O	1-6	<i>GTS_cycle</i>
Release DONE pin	1-6	<i>DONE_cycle</i>
End Of Startup (EOS)	7	N/A

The specific order of startup events, except for the End of Startup (EOS) is user-programmable through various bitstream generator options. Table 12-7 and Figure 12-12, page 249 show the general sequence of events, although the specific phase for each of these startup events is user-programmable. EOS is always the last phase. By default, startup events occur as shown in Table 12-7.

Table 12-7: Default BitGen Sequence of Startup Events

BitGen Control	Default Setting (Phase)	Event
<i>DONE_cycle</i>	4	Release DONE pin, indicating that the FPGA successfully completed configuration.
<i>GTS_cycle</i>	5	Release the global three-state control (GTS), activating I/O
<i>GWE_cycle</i>	6	Assert the global write-enable (GWE), allowing RAM and flip-flops to change state
N/A	7	Assert EOS

The FPGA automatically pulses the Global Set/Reset (GSR) signal when entering the Startup sequence, forcing all flip-flops and latches in a known state. The sequence and timing of how the FPGA switches over is programmable as is the clock source controlling the sequence.

The default start-up sequence appears in Figure 12-12, where the Global Three-State signal (GTS) is released one clock cycle after DONE goes High. This sequence allows the DONE signal to enable or disable any external logic used during configuration before the user application in the FPGA starts driving output signals. One clock cycle later, the Global Write Enable (GWE) signal is released. This allows signals to propagate within the FPGA before any clocked storage elements such as flip-flops and block ROM are enabled.

The function of the dual-purpose I/O pins, such as M[2:0], VS[2:0], HSWAP, PUDC_B, and A[25:0], also changes when the Global Three-State (GTS) signal is released. The dual-purpose configuration pins become user I/Os. The exception on Spartan-3E and Extended Spartan-3A family FPGAs is the CCLK pin, which becomes a user-I/O pin at the End of Startup (EOS).

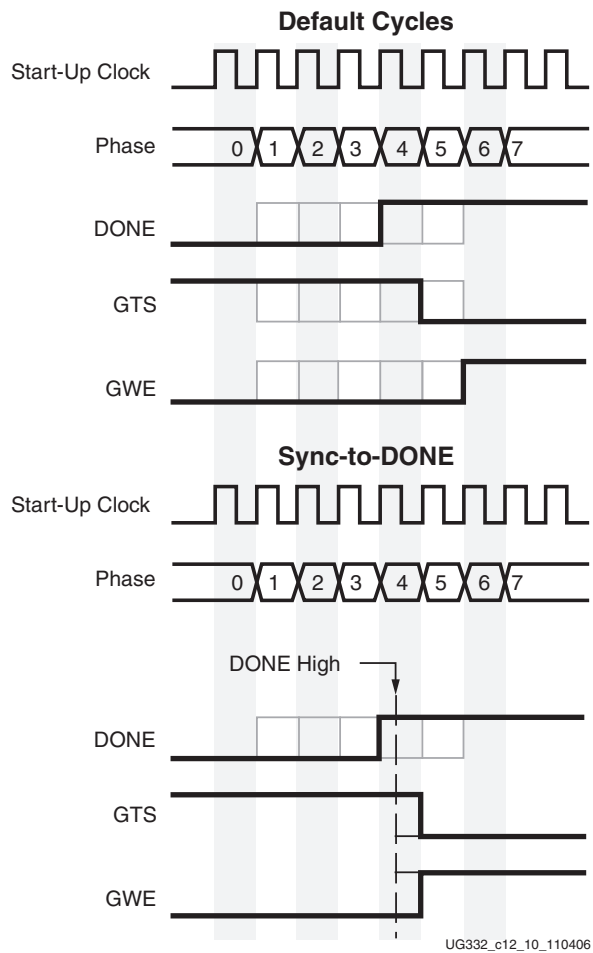


Figure 12-12: Default Start-Up Sequence

Startup Clock Source

There are three possible clock sources for the Startup sequencer, controlled by the *StartupClk* bitstream generator option.

1. By default, the start-up sequence is synchronized to CCLK. The *Cclk* option or the *UserClk* option is required for Master Mode or Slave Mode configuration.
2. Alternatively, the start-up sequence can be synchronized to a user-specified clock from within the FPGA application using the “Start-Up (STARTUP),” page 255 library primitive and by setting the *StartupClk:UserClk* bitstream generator option.
3. When using JTAG configuration, the start-up sequence must be synchronized to the TCK clock input (*StartupClk:JtagClk*).

Waiting for DCMs to Lock, DCI to Match

The startup sequence can be forced to wait for the DCMs to lock or for DCI to match with the appropriate BitGen options. These options are typically set to prevent DONE, GTS, and GWE from being asserted (preventing FPGA operation) before the DCMs have locked and/or DCI has matched.

The DONE signal is released by the startup sequencer on the cycle indicated in the bitstream, set by the *DONE_cycle* bitstream generator option. However, the Startup sequencer does not proceed beyond the specified Startup cycle until the DONE pin actually sees an external logic High. The DONE pin is an open-drain bidirectional signal by default. By releasing the DONE pin, the FPGA simply stops driving a logic Low and the pin goes into a high-impedance (Hi-Z) state. A pull-up resistor, either internal or external, is required for the DONE pin to reach a logic High in this case. Table 12-8 shows signals relating to the startup sequencer. Figure 1-12 shows the waveforms relating to the startup sequencer.

Table 12-8: Signals Relating to Startup Sequencer

Signal Name	Type	Access	Description
DONE	Bidirectional	DONE pin or Status Register	Indicates configuration is complete. Can be held Low externally to synchronize startup with other FPGAs.
Release_DONE	Status	Status Register	Indicates whether the FPGA has stopped driving the DONE pin Low. If the pin is held Low externally, Release_DONE can differ from the actual value on the DONE pin.
GWE			Global Write Enable (GWE). When deasserted, GWE disables the CLB and the IOB flip-flops as well as other synchronous elements on the FPGA.
GTS			Global 3-State (GTS). When asserted, GTS disables all the I/O drivers except for the configuration pins.
EOS			End of Startup (EOS). EOS indicates the absolute end of the configuration and startup process.
DCI_MATCH			<i>Spartan-3 FPGA Family only.</i> DCI_MATCH indicates when all the Digitally Controlled Impedance (DCI) controllers have matched their internal resistor to the external reference resistor.
DCM_LOCK			DCM_LOCK indicates when all the Digital Clock Managers (DCMs) have locked. This signal is asserted by default. It is active if the STARTUP_WAIT option is used on a DCM and the <i>LCK_cycle</i> option is set in the bitstream.

Figure 12-13 is a generalized block diagram of the configuration logic, showing the interaction of different device inputs and Bitstream Generator (BitGen) options.

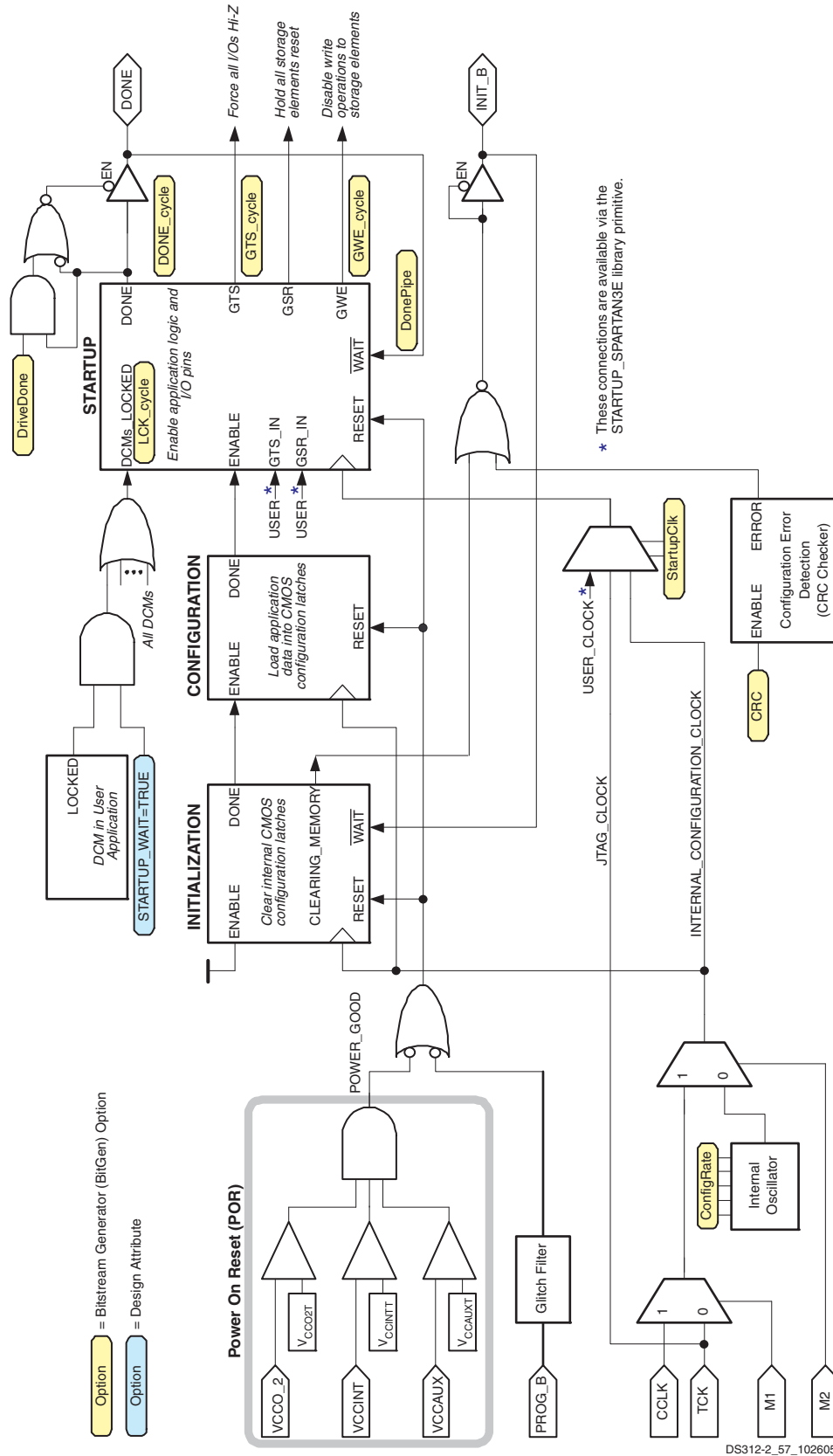


Figure 12-13: Extended Spartan-3A Family, Spartan-3E FPGA Configuration Logic Conceptual Block Diagram

Configuration-Related Design Primitives

The following configuration primitives provide access to FPGA configuration resources during or after FPGA configuration.

Boundary-Scan (BSCAN)

The BSCAN component, shown in [Figure 13-1](#), provides access to and from the JTAG Boundary Scan logic controller from internal FPGA logic, allowing communication between the internal FPGA application and the dedicated JTAG pins of the FPGA. The BSCAN primitive is not required for normal JTAG operations. It is only required when implementing private JTAG scan chains within the FPGA logic. Although the BSCAN primitive is functionally equivalent on all Spartan[®]-3 generation FPGAs, the primitive name varies by family, as shown in [Table 13-1](#), [page 254](#).

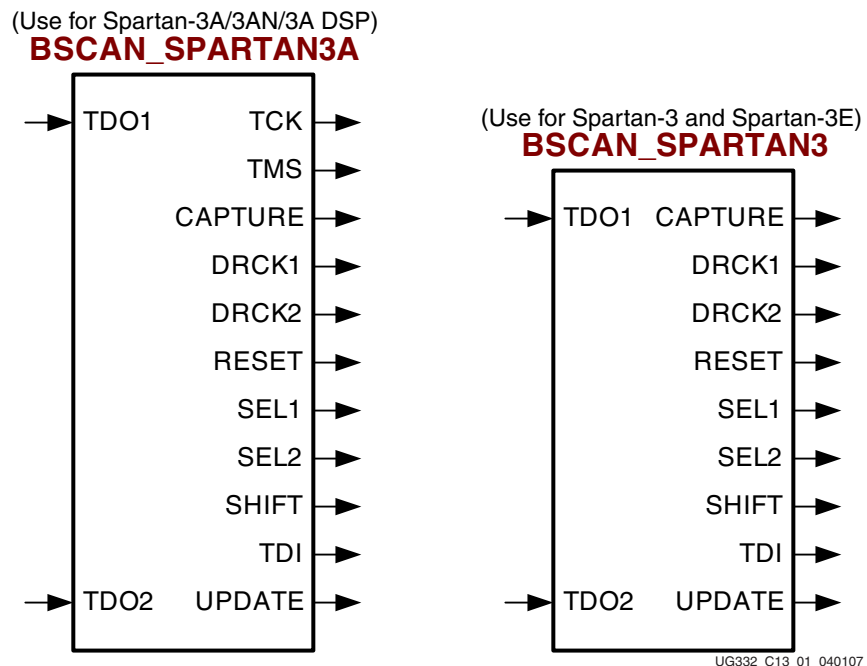


Figure 13-1: BSCAN Primitive for Extended Spartan-3A Family FPGAs

Table 13-1: BSCAN Primitives by FPGA Family

FPGA Family	Primitive
Spartan-3A/3AN FPGAs Spartan-3A DSP FPGAs	BSCAN_SPARTAN3A
Spartan-3E FPGAs	BSCAN_SPARTAN3
Spartan-3 FPGAs	

The BSCAN primitive on Spartan-3 generation FPGAs allows up to two internal, private boundary scan chains called USER1 and USER2.

A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active. The DRCK1 output provides USER1 access to the data register clock (generated by the TAP controller). The TDO2 and SEL2 pins perform a similar function for the USER2 instruction and the DRCK2 output provides USER2 access to the data register clock (generated by the TAP controller). The RESET, UPDATE, SHIFT, and CAPTURE pins represent the decoding of the corresponding state of the boundary scan internal state machine. The TDI pin provides access to the TDI signal of the JTAG port in order to shift data into an internal scan chain.

Usage

The BSCAN component is generally used with IP, such as the ChipScope™ analyzer tool, for communications via the JTAG pins of the FPGA to the internal device logic. When used with this IP, this component is generally instantiated as a part of the IP and nothing more is needed by the user to ensure it is properly used. However, the BSCAN component can be instantiated in any FPGA design although only one BSCAN component can be used in any single design.

Port Descriptions

Table 13-2: BSCAN Primitive Connections

Port Name	Direction	Function
TDI	Output	The value of the TDI input pin to the FPGA.
TCK	Output	The value of the TCK input pin to the FPGA.
TMS	Output	The value of the TMS input pin to the FPGA.
DRCK1, DRK2	Output	The value of the TCK input pin to the FPGA when the JTAG USER instruction is loaded and the JTAG TAP controller is in the SHIFT-DR state. DRCK1 applies to the USER1 logic while DRCK2 applies to USER2.
RESET	Output	Active upon the loading of the USER instruction. It asserts High when the JTAG TAP controller is in the TEST-LOGICRESET state.
SEL1, SEL2	Output	Indicates when the USER1 or USER2 instruction is loaded into the JTAG Instruction Register. SEL1 or SEL2 becomes active in the UPDATE-IR state, and stays active until a new instruction is loaded.

Table 13-2: BSCAN Primitive Connections (Cont'd)

Port Name	Direction	Function
SHIFT	Output	Active upon the loading of the USER instruction. It asserts High when the JTAG TAP controller is in the SHIFT-DR state.
CAPTURE	Output	Active upon the loading of the USER instruction. Asserts High when the JTAG TAP controller is in the CAPTURE-DR state.
UPDATE	Output	Active upon the loading of the USER instruction. It asserts High when the JTAG TAP controller is in the UPDATE-DR state.
TDO1, TDO2	Input	Active upon the loading of the USER1 or USER2 instruction. External JTAG TDO pin reflects data input to the component's TDO1 (USER1) or TDO2 (USER2) pin.

Start-Up (STARTUP)

The STARTUP primitive is used to either interface device pins and/or logic to the global asynchronous set/reset (GSR) signal, or for global, 3-state (GTS) dedicated routing. This primitive can also be used to specify a different clock for the device startup sequence at the end of configuring the device.

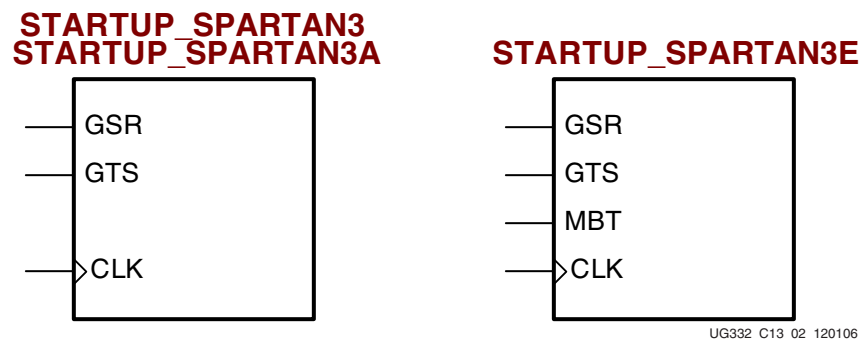


Figure 13-2: STARTUP Primitive for Extended Spartan-3A Family and Spartan-3E FPGAs

As shown in Figure 13-2, the STARTUP primitive is similar between Spartan-3 generation FPGA families, although the Spartan-3E STARTUP primitive has an additional input pin to support MultiBoot functions. The specific STARTUP primitive name also varies by family, as indicated in Table 13-3.

Table 13-3: STARTUP Primitives by FPGA Family

FPGA Family	Primitive
Spartan-3A/3AN FPGAs Spartan-3A DSP FPGAs	STARTUP_SPARTAN3A
Spartan-3E FPGAs	STARTUP_SPARTAN3E
Spartan-3 FPGAs	STARTUP_SPARTAN3

Usage

The STARTUP primitive must be instantiated into the design. To use the dedicated GSR circuitry, connect the sourcing pin or logic to the GSR pin. However, avoid using the GSR circuitry of this component unless certain precautions are taken first. Since the skew of the GSR net cannot be guaranteed, either use general routing for the set/reset signal in which routing delays and skew can be calculated as a part of the timing analysis of the design, or ensure that possible skew during the release of GSR will not interfere with proper circuit operation.

Similarly, if the dedicated global 3-state is used, connect the appropriate sourcing pin or logic to the GTS input pin of the primitive. In order to specify a user clock for the startup sequence of configuration, connect a clock from the design to the CLK pin of the STARTUP component.

Port Descriptions

Table 13-4: STARTUP Primitive Connections

Port Name	Direction	Function
GSR	Input	Active-High global set / reset (GSR) signal.
GTS	Input	Active-High global 3-state (GTS) signal.
MBT	Input	<i>Spartan-3E family only.</i> Active-Low, asynchronous MultiBoot trigger input.
CLK	Input	Optional clock input to the configuration Startup sequencer, selected using <i>StartupClk:UserClk</i> bitstream option.

Readback Capture (CAPTURE)

The CAPTURE primitive, shown in Figure 13-3, provides FPGA application control over when to capture register (flip-flop and latch) information for readback. Spartan-3 generation FPGAs provide the readback function through dedicated configuration port instructions.

CAPTURE_SPARTAN3A

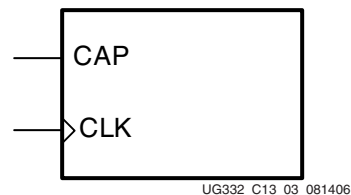


Figure 13-3: CAPTURE Primitive for Extended Spartan-3A Family FPGAs (other families are similar)

Caution! On Spartan-3E FPGAs, Readback is available on all devices except for the XC3S1200E and XC3S1600E in the -4 speed grade, in the commercial temperature range. Readback is supported on all Spartan-3E FPGAs available in the -5 speed grade or in the industrial temperature range.

Usage

The CAPTURE primitive is optional within a design. Without it, Readback is still performed, but the asynchronous capture function it provides for register states is not available.

Spartan-3 generation FPGAs only capture register (flip-flop and latch) states. Although LUT RAM, SRL, and block RAM bit values are read back, their values cannot be captured. To capture the register states, assert the CAP signal High. The state is captured on the next rising edge of CLK.

By default, data is captured after every trigger (transition on CLK while CAP is asserted). To limit the readback operation to a single data capture, add the ONESHOT attribute to CAPTURE devices.

Although the CAPTURE primitive functions equivalently on all Spartan-3 generation FPGA families, the required design primitive varies by family, as indicated in [Table 13-5](#).

Table 13-5: CAPTURE Primitive by FPGA Family

FPGA Family	Primitive
Spartan-3A/3AN FPGAs Spartan-3A DSP FPGAs	CAPTURE_SPARTAN3A
Spartan-3E FPGAs	CAPTURE_SPARTAN3
Spartan-3 FPGAs	

For more information on Readback and the CAPTURE primitive, see [XAPP452: Spartan-3 Advanced Configuration Architecture](#).

Port Description

Table 13-6: CAPTURE Primitive Connections

Port Name	Direction	Description
CLK	Input	Clock for sampling the CAP input.
CAP	Input	Active-High capture enable. The CAP input is sampled by the rising edge of CLK.

Attributes

[Table 13-7](#) describes the ONESHOT attribute available on the CAPTURE primitive.

Table 13-7: CAPTURE Attributes

Attribute	Type	Allowed Values	Default	Description
ONESHOT	Boolean	TRUE, FALSE	FALSE	Specifies the procedure for performing single readback operation per CAP trigger.

Internal Configuration Access Port (ICAP)

The Internal Configuration Access Port (ICAP), shown in [Figure 13-4](#), is only available on the Extended Spartan-3A family FPGA families.

Usage

The ICAP_SPARTAN3A primitive works similar to the Slave Parallel (SelectMAP) configuration interface except it is available to the FPGA application using internal routing connections. Furthermore, the ICAP primitive has separate read and write data ports, as opposed to the bidirectional bus on the Slave Parallel (SelectMAP) interface. ICAP allows the FPGA application to access configuration registers, readback configuration data, or to trigger a MultiBoot event after configuration successfully completes.

For additional information on the Slave Parallel (SelectMAP) interface, see [Chapter 7](#), “Slave Parallel (SelectMAP) Mode.”

For additional information on Extended Spartan-3A family MultiBoot, [Chapter 14](#), “Reconfiguration and MultiBoot.”

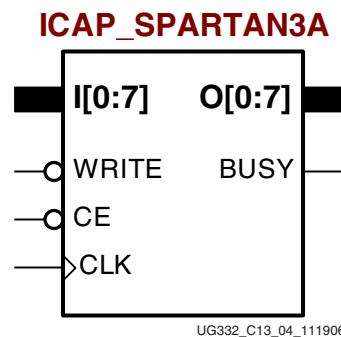


Figure 13-4: ICAP Primitive (only available on Extended Spartan-3A Family FPGAs)

Port Description

Caution! Xilinx convention defines I0 and O0 as the most-significant bits; I7 and O7 are the least-significant bits. This is different than conventions elsewhere. Watch out for bit reversals!

Table 13-8: ICAP_SPARTAN3A Primitive Connections

Signal Name	Equivalent SelectMAP Pin Name	Direction	Description
CLK	CCLK	Input	ICAP interface clock
CE	CS_B or CSI_B	Input	Active-Low select
WRITE	RDWR_B	Input	Read/Write control input 0 = WRITE 1 = READ
I[0:7]	D[0:7]	Input	Byte-wide ICAP write data bus
O[0:7]	D[0:7]	Output	Byte-wide ICAP read data bus
BUSY	DOUT	Output	Active-High busy status. Only used in read operations. BUSY remains Low during writes.

Device DNA Access Port (DNA_PORT)

The DNA_PORT primitive, shown in Figure 13-5 is only available on the Extended Spartan-3A family FPGA.

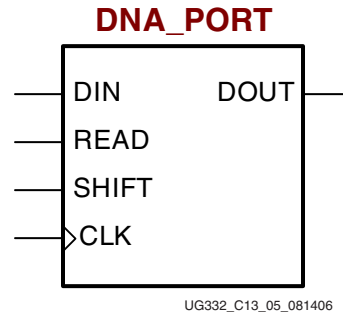


Figure 13-5: DNA_PORT Primitive (only available on Extended Spartan-3A Family FPGAs)

The DNA_PORT provides access to a dedicated shift register which can be loaded with the Device DNA data bits (unique ID) for a given Extended Spartan-3A family device. In addition to shifting out the DNA data bits, this component allows for the inclusion of supplemental data bits for additional user data or allow for the DNA data to rollover (repeat DNA data after initial data has been shifted out). This component is primarily used in conjunction with other circuitry to build anti-cloning protection for the FPGA bitstream from possible theft. See Chapter 15, “Protecting FPGA Designs,” for additional information.

Usage

The DNA_PORT component must be instantiated in order to be used in a design. To do so, use the instantiation template found within the ISE® software Project Navigator HDL Templates and place this instance declaration within the code. Connect all inputs and outputs to the design in order to ensure proper operation.

In order to access the Device DNA data, the shift register must first be loaded by setting the active high READ signal for one clock cycle. After the shift register is loaded, the data may be synchronously shifted out by enabling the active high SHIFT input and capturing the data out the DOUT output port. If desired, additional data may be appended to the end of the 57-bit shift register by connecting the appropriate logic to the DIN port. If DNA data rollover is desired, connect the DOUT port directly to the DIN port to allow for the same data to be shifted out after completing the 57-bit shift operation. If no additional data is necessary, the DIN port may be tied to a logic zero. The attribute SIM_DNA_VALUE may be optionally set to allow for simulation of a possible DNA data sequence. By default, the Device DNA data bits are all zeros in the simulation model.

See “Operation,” page 294 for additional information.

Port Descriptions

Table 13-9: DNA_PORT Primitive Connections

Port Name	Direction	Function
DOUT	Output	Serial shifted output data
DIN	Input	User data input to the shift register
READ	Input	Synchronous load of the shift register with the Device DNA data A READ operation overrides a SHIFT operation.
SHIFT	Input	Active high shift enable input
CLK	Input	Clock Input

Attributes

Table 13-10: DNA_PORT Attributes

Attribute	Type	Allowed Values	Default	Description
SIM_DNA_VALUE	57-bit vector	Any 57-bit value	All zeros	Specifies a DNA value for simulation purposes (the actual value will be specific to the particular device used)

Reconfiguration and MultiBoot

Overview

Because Spartan[®]-3 generation FPGAs are reprogrammable, in the system, some applications reload the FPGA with one or more bitstream images during normal operation. In this way, a single, smaller FPGA, reprogrammed multiple times, replaces a much larger and more expensive ASIC or FPGA programmed just once.

There are a variety of methods to reprogram the FPGA during normal operation. The downloaded configuration modes (see [Figure 1-2, page 30](#)) inherently provide this capability. Via an external “intelligent agent” such as a processor, microcontroller, computer, or tester, an FPGA can be reprogrammed numerous times. The downloaded modes are available on all Spartan-3 generation FPGA families.

The Spartan-3E and Extended Spartan-3A family FPGA families introduce a new capability, called *MultiBoot*, that allows the FPGA to selectively reprogram and reload its bitstream from an attached external memory.

The MultiBoot feature allows the FPGA application to load two or more FPGA bitstreams under the control of the FPGA application. The FPGA application triggers a MultiBoot operation, causing the FPGA to reconfigure from a different configuration bitstream. As shown in [Table 14-1](#), there are differences between MultiBoot on Spartan-3E and Spartan-3A/3AN/3A DSP FPGAs.

Once a MultiBoot operation is triggered, the FPGA restarts its configuration process as usual. The INIT_B pin pulses Low while the FPGA clears its configuration memory and the DONE output remains Low until the MultiBoot operation successfully completes.

For Spartan-3E FPGA applications, see “[Spartan-3E MultiBoot](#),” page 263. For Extended Spartan-3A family FPGA applications, see “[Extended Spartan-3A Family MultiBoot](#),” page 271.

MultiBoot Options Compared between Spartan-3 Generation FPGA Families

[Table 14-1](#) highlights the primary MultiBoot differences between Spartan-3 generation FPGA families. The MultiBoot feature is available only on the Spartan-3E and Spartan-3A FPGA families.

Table 14-1: MultiBoot Options on Spartan-3 Generation FPGA Families

	Spartan-3	Spartan-3E	Spartan-3A/3AN Spartan-3A DSP
Application complexity	MultiBoot not available on Spartan-3 FPGA family.	Simple	More complex, but also more capable and flexible
MultiBoot in BPI mode using parallel NOR Flash		Yes	Yes
MultiBoot in SPI mode using SPI serial Flash		No	Yes
MultiBoot from In-System Flash memory		No	Spartan-3AN only ⁽¹⁾
MultiBoot between different configuration modes		No	Yes
MultiBoot supports multi-FPGA configuration daisy chains		No, single FPGA only	Yes
How is MultiBoot triggered by FPGA application?		MBT input on STARTUP primitive	Via command sequence to ICAP primitive, JTAG interface, Slave Serial, or Slave Parallel (SelectMAP) interface
Maximum number of MultiBoot configuration images		2 (top and bottom of parallel Flash)	Limited only by the amount of configuration memory
Bitstream start locations and addressing direction		Ether at address 0 with incrementing addresses or highest PROM address with decrementing addresses	Any byte location, always with incrementing addresses
Initial MultiBoot image location		Controlled by M0 mode pin. 0 = Address 0 1 = Highest PROM address	Always at address 0
Can FPGA application specify MultiBoot start address?		No (always top and bottom of parallel Flash)	Yes
Configuration watchdog timer automatically reconfigures FPGA starting at address 0 if MultiBoot operation fails	No	Yes	

Notes:

1. See [Spartan-3AN Errata](#) regarding limitations using MultiBoot after configuration from the in-system Flash.

Spartan-3E MultiBoot

After the FPGA configures itself using BPI mode from one end of the parallel Flash PROM, then the FPGA can trigger a MultiBoot event and reconfigure itself from the opposite end of the parallel Flash PROM. MultiBoot is only available when using BPI mode and only for applications using a single Spartan-3E FPGA. MultiBoot does not support multi-FPGA configuration daisy chains.

By default, the MultiBoot feature is disabled. To use MultiBoot in an application, the FPGA design must first include a `STARTUP_SPARTAN3E` design primitive, described in “Start-Up (`STARTUP`),” page 255. To trigger a MultiBoot event, assert a Low pulse lasting at least 300 ns on the MultiBoot Trigger (MBT) input to the primitive. When the MBT signal returns High after the Low pulse, the FPGA automatically reconfigures from the opposite end of the parallel Flash memory.

Figure 14-1 illustrates a simple MultiBoot design example. At power up, the FPGA loads itself from the attached parallel Flash PROM. In this specific example, the M0 mode pin is Low so the FPGA configures starting at Flash address 0 and increments through the PROM memory locations. After the FPGA completes configuration, this example FPGA application performs a board-level or system test using FPGA logic. If the test is successful, the FPGA then triggers a MultiBoot event, causing the FPGA to reconfigure from the opposite end of the Flash PROM memory, in this case starting at address `0xFFFF`. The FPGA actually starts at address `0xF_FFFF` but the upper four address bits, `A[23:20]`, are not connected to the PROM in this example. The FPGA addresses the second configuration image, which in this example contains the FPGA application for normal operation.

Similarly, the second FPGA application could trigger another MultiBoot event at any time to reload the diagnostics design from address 0, and so on.

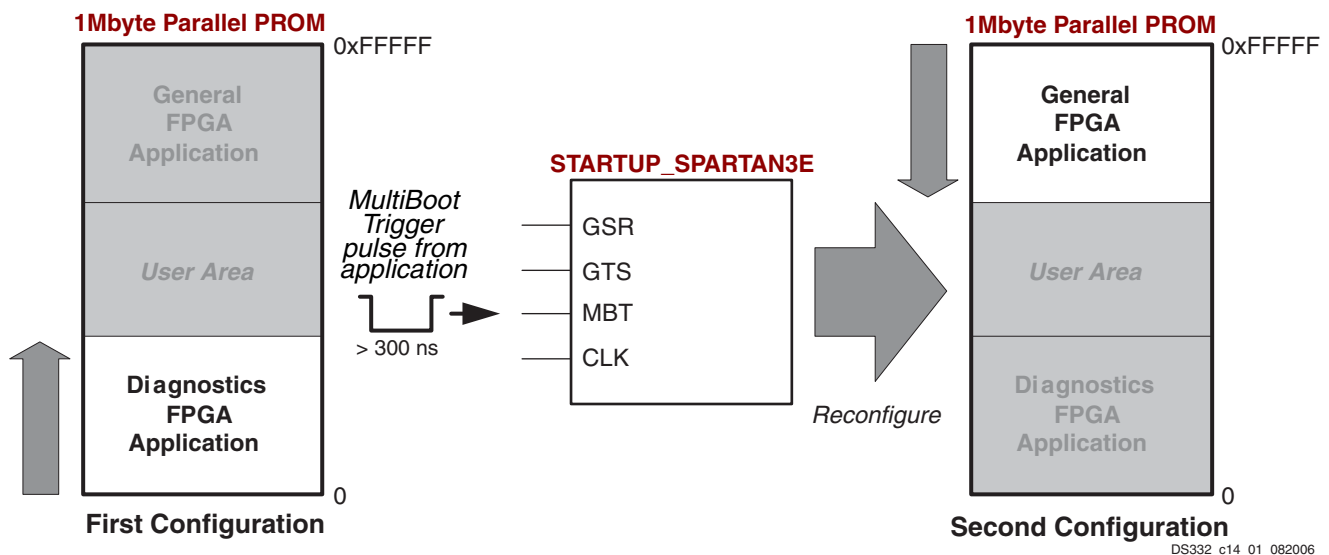


Figure 14-1: Example Spartan-3E MultiBoot Application using 1Mbyte Parallel Flash PROM

In another potential application, the initial design loaded into the FPGA image contains a “golden” or “fail-safe” configuration image, which then communicates with the outside world and checks for a newer FPGA configuration image. If there is a new configuration revision and the new image verifies as good, the “golden” configuration triggers a MultiBoot event to load the new image.

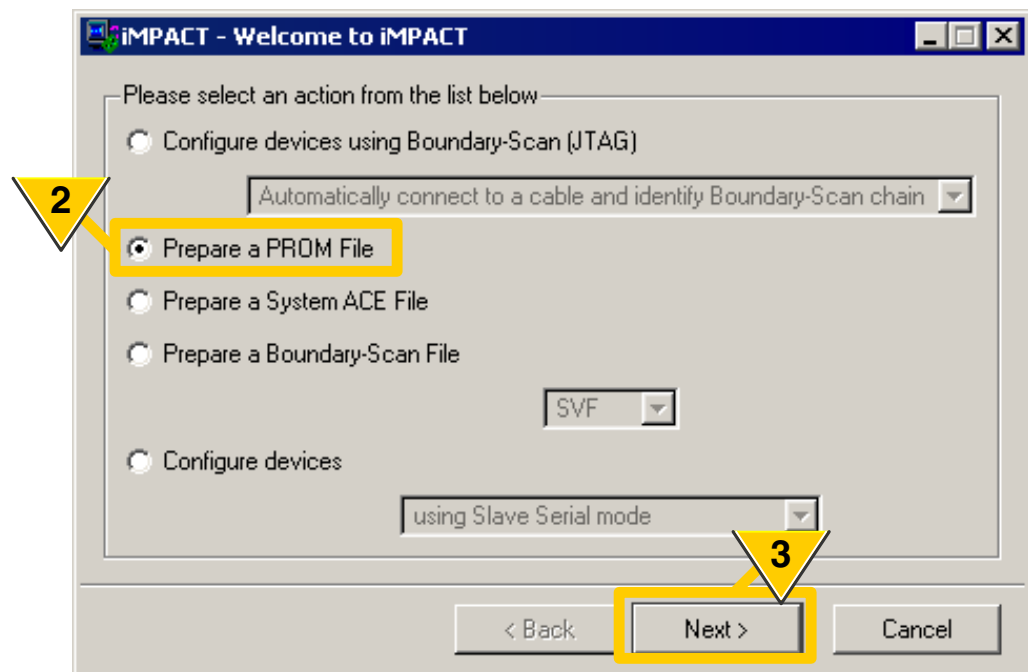
When a MultiBoot event is triggered, the FPGA then again drives its configuration pins as described in [Chapter 5, “Master BPI Mode.”](#) However, the FPGA does not assert the PROG_B pin. The system design must ensure that no other device drives on these same pins during the reconfiguration process. The FPGA’s DONE, LDC[2:0], or HDC pins can be used to temporarily disable any conflicting drivers during reconfiguration.

Asserting the PROG_B pin Low overrides the MultiBoot feature and forces the FPGA to reconfigure starting from the end of memory defined by the mode pins, shown in [Table 5-2, page 147.](#)

Generating a Spartan-3E MultiBoot PROM Image using iMPACT

The iMPACT programming software provides a graphical, step-by-step approach to create a MultiBoot PROM file. Similar functionality is also available from the command line or via scripts using the PROMGen utility, shown in [Figure 14-10.](#) Follow the steps outlined below to create a MultiBoot PROM file using the iMPACT software. The steps assume an example application like that shown in [Figure 14-1.](#)

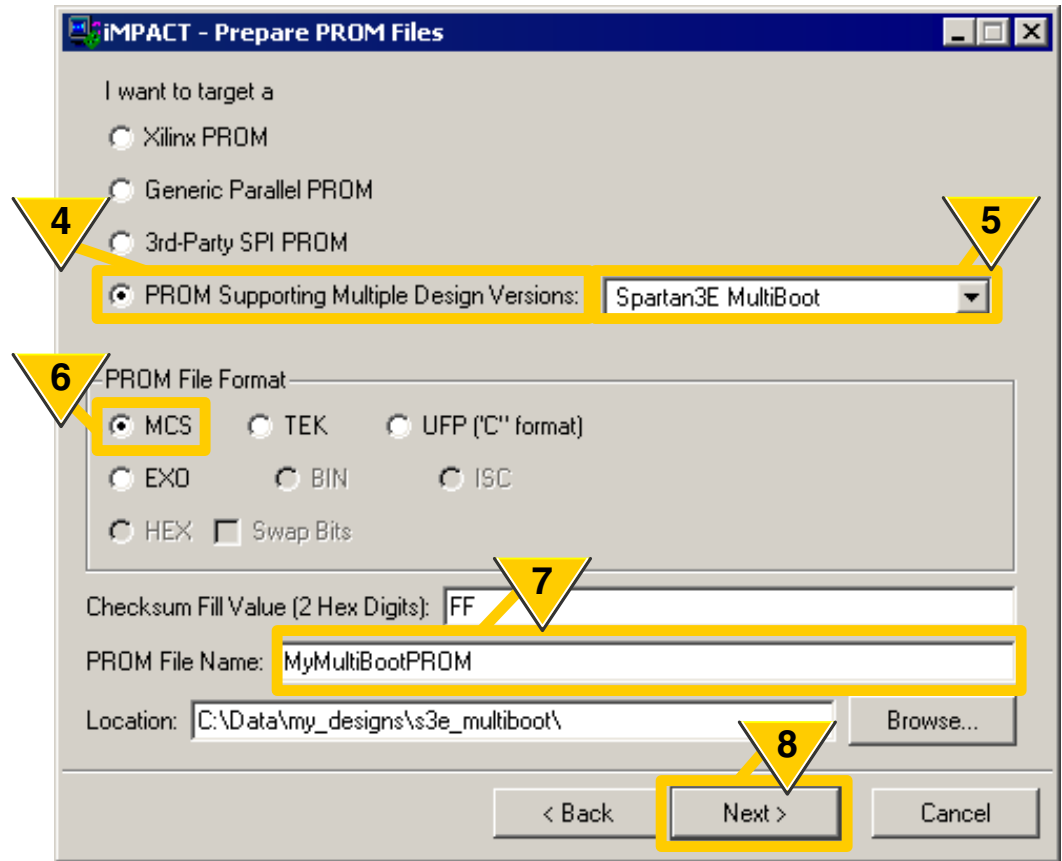
1. Invoke the iMPACT programming software.
2. As shown in [Figure 14-2,](#) choose **Prepare a PROM File.**



UG332_c14_04_112906

Figure 14-2: Prepare a MultiBoot PROM Image

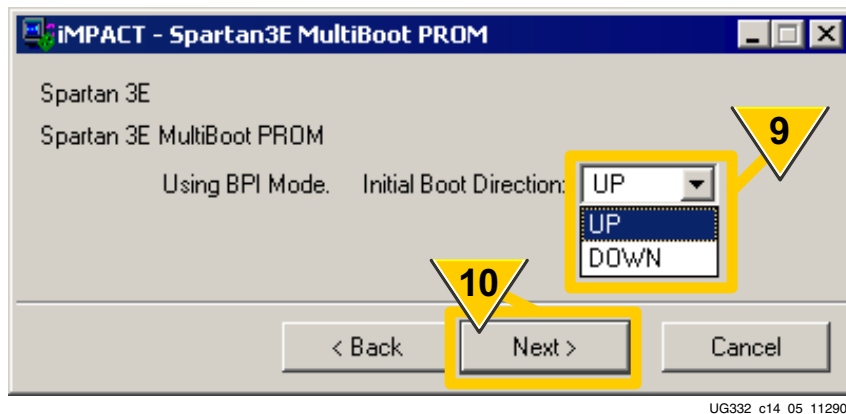
3. Click Next.
4. As shown in [Figure 14-3,](#) target a **PROM Supporting Multiple Design Revisions.**



UG332_c14_03_112906

Figure 14-3: Select a PROM Supporting MultiBoot for Spartan-3E FPGAs

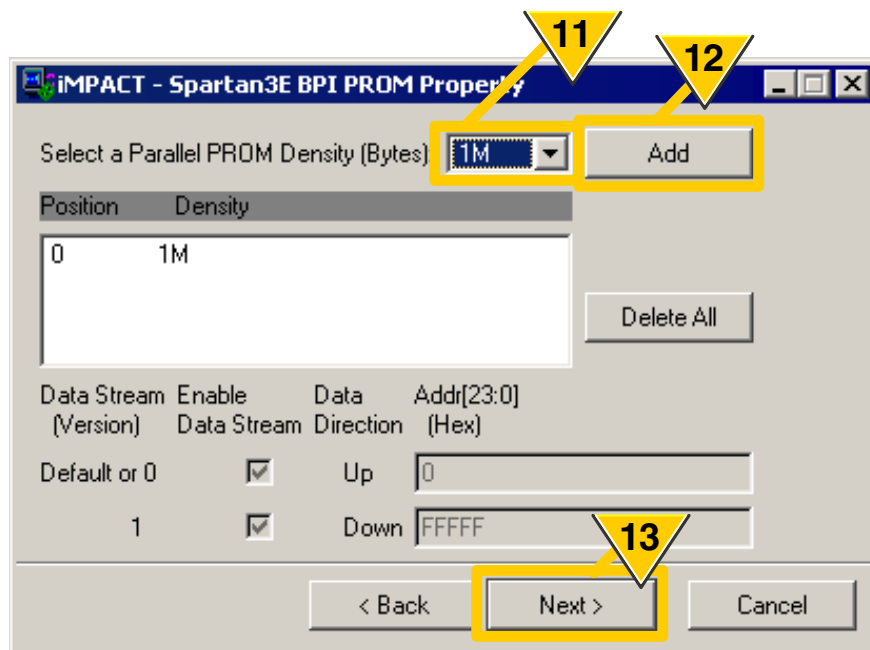
5. Choose the **Spartan3E MultiBoot** method.
6. Select a **PROM File Format**. The MCS format is supported by a variety of programmers, but other options are available.
7. Enter a **PROM File Name**.
8. Click **Next**.
9. As shown in Figure 14-4, select the **Initial Boot Direction**. This is the location from where the first configuration image loads. The initial location depends on the BPI mode pin settings.



UG332_c14_05_112906

Figure 14-4: Select the Configuration Direction of the First MultiBoot Image

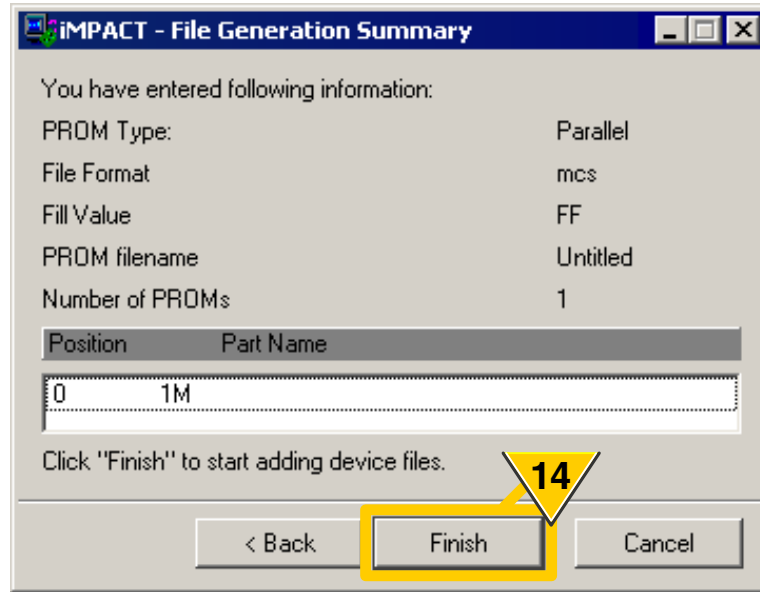
10. Click Next.
11. As shown in Figure 14-5, Select a Parallel PROM Density, measured in bytes.



UG332_c14_06_112906

Figure 14-5: Select a PROM Size and Add It to the Design

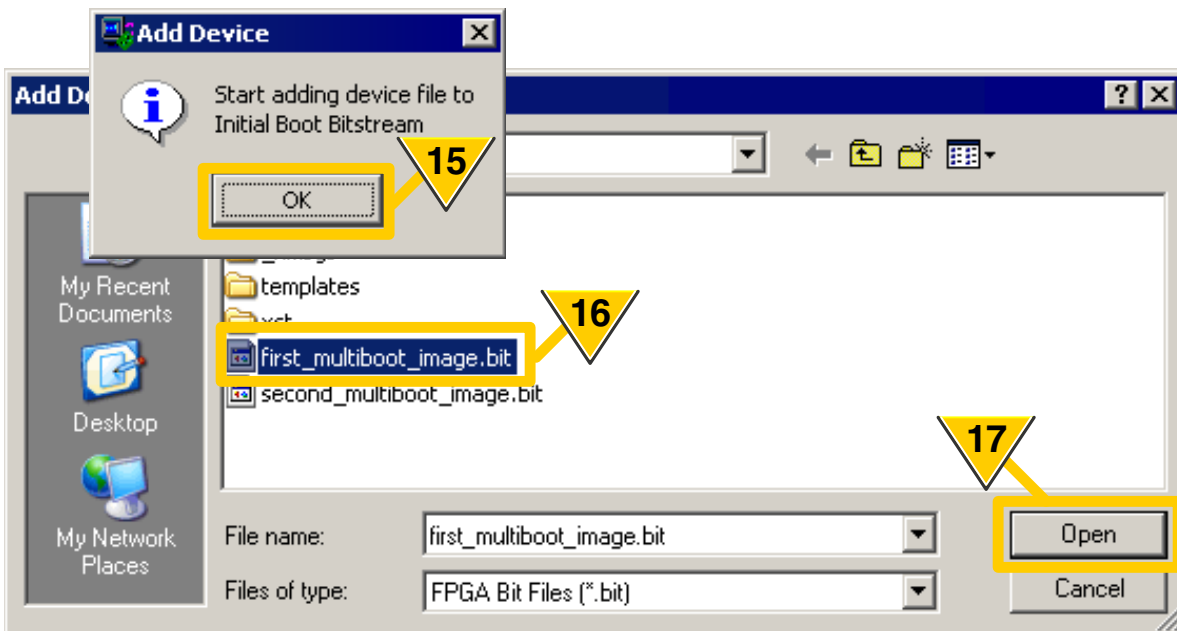
12. Click **Add** to use the PROM density specified in Step 11. In Spartan-3E MultiBoot mode, only a single PROM is allowed. The PROM density also determines the highest PROM address location.
13. Click **Next**.
14. The iMPACT software summarizes the current settings, as shown in Figure 14-6. Click **Finish** to continue.



UG332_c14_07_112906

Figure 14-6: Confirm the PROM Settings

- As shown in Figure 14-7, start selecting the FPGA configuration bitstream for the design that initially loads at power-up or when the PROG_B input is pulsed Low.



UG332_c14_08_081906

Figure 14-7: Select the First MultiBoot Configuration Image

- Using the file selection mechanism for your operating system, choose the initial bitstream. This bitstream is loaded at the initial PROM location specified in Step 9.
- Click **Open**.

- As shown in [Figure 14-8](#), the iMPACT software then prompts for the second MultiBoot configuration image.

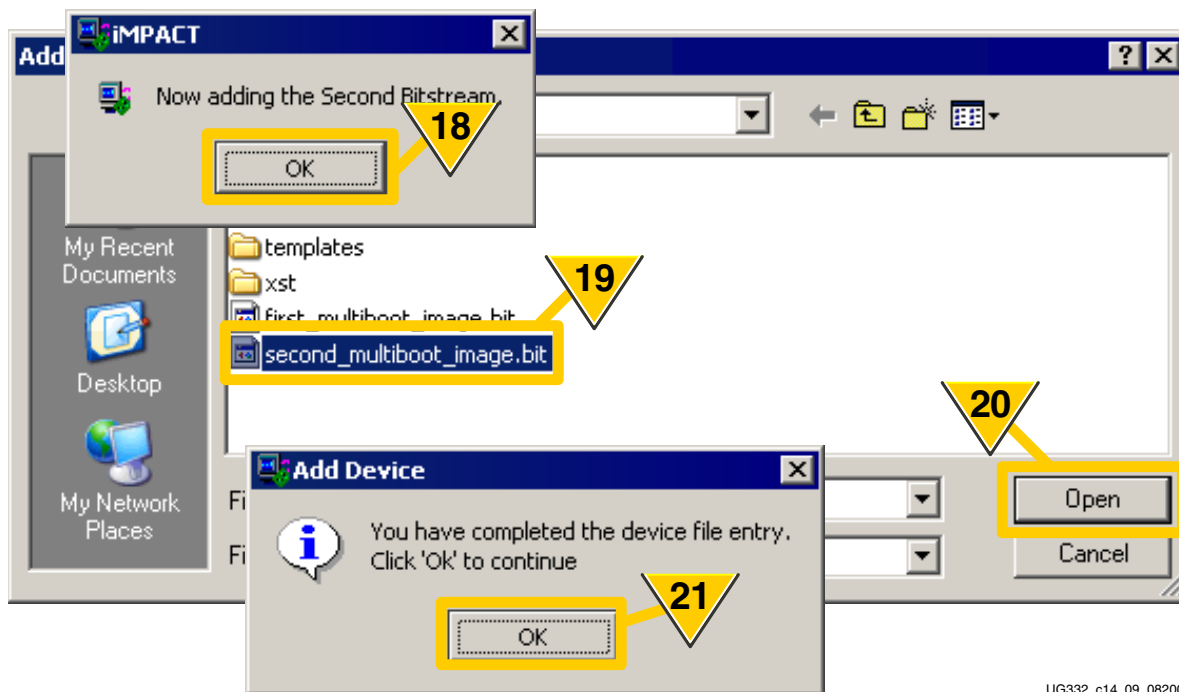
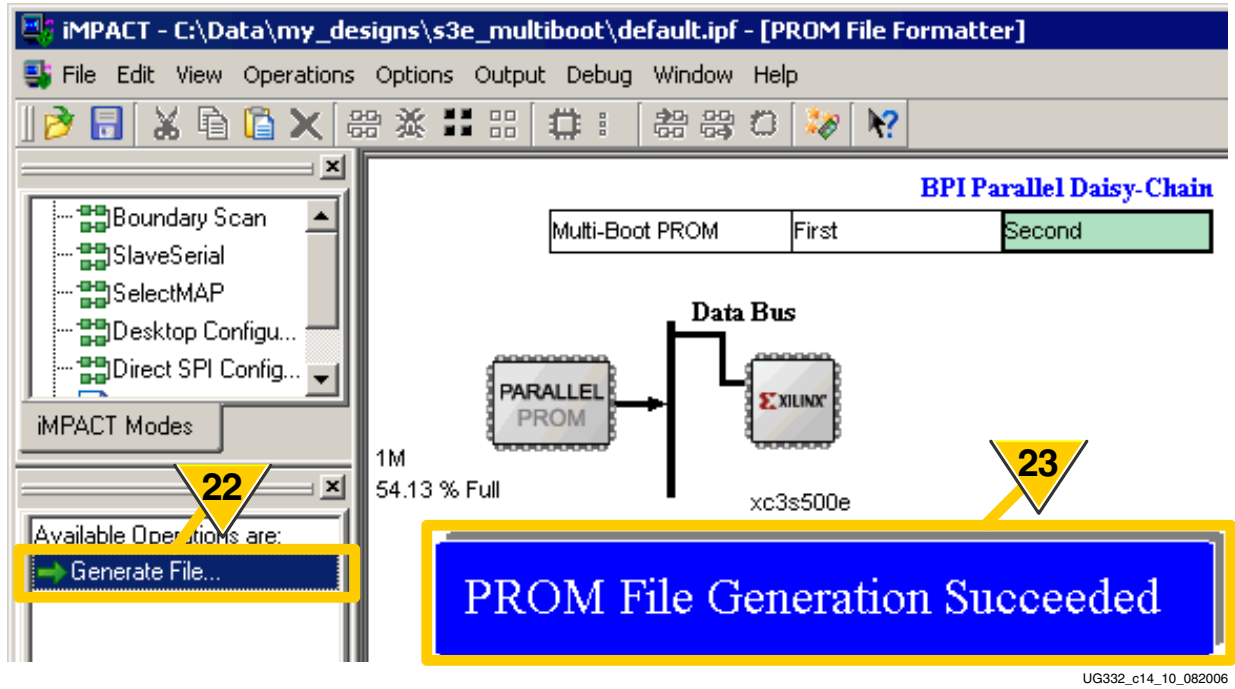


Figure 14-8: Select the Second MultiBoot Configuration Image

- Select the `.bit` file for the second image.
- Click **Open**.
- The iMPACT software then confirms that all the necessary files are entered.
- As shown in [Figure 14-9](#), the iMPACT software reports how much of the PROM is consumed by the FPGA configuration bitstream files. Double-click **Generate File**.



UG332_c14_10_082006

Figure 14-9: Generate the PROM File Using the Specified Parameters

- 23. The iMPACT software successfully generates a PROM file using the name specified in Step 7 with the format and file extension specified in Step 6. The file is created in the current directory. A “PROMGen Report File” is also created.

PROMGen Report File

The iMPACT software creates the PROM file using the PROMGen command-line program. The PROMGen software also creates a report file with an *.prm file extension, as shown in Figure 14-10.

```

PROMGEN: Xilinx Prom Generator I .32
Copyright (c) 1995-2006 Xilinx, Inc. All rights reserved.
1 promgen -w 2 -p mcs -c FF -o 3 MyMultiBootPROM -u 0 4 first_multiboot_image.bit
5 -d fffff second_multiboot_image.bit
6 -s 1024

PROM MyMultiBootPROM .prm

Calculating PROM checksum with fill value ff

Format      Mcs 86 (32-bit)
Size        1024K
PROM start  0000:0000
PROM end    000f:ffff
PROM checksum 07515767

7 Addr1      Addr2      Date File(s)
8 0000:0000  0004:547f  Aug 18 22:38:14 2006 first_multiboot_image.bit
  000f:ffff  000b:ab80  Aug 18 22:37:07 2006 second_multiboot_image.bit

```

UG332_c14_11_082006

Figure 14-10: PROMGen Report File (*.prm)

The following items correspond to the markers in Figure 14-10.

1. PROMGen is the command-line program that generates PROM programming files using the specified format.
2. Various formats are available. The Intel MCS format is one of the popular options.
3. The base output file name. The extension depends on the selected format.
4. In the example shown above, the first MultiBoot file is loaded for the BPI Up mode, meaning that the file starts at address 0.
5. The second MultiBoot file is loaded at the opposite end of memory, in this case at the maximum PROM address and loaded downward.
6. The PROM size is specified in kilobytes (K). In the example, the PROM is 1Mbyte or 1024K.
7. The first MultiBoot image is loaded starting at PROM address 0 and ends at hexadecimal address 0x4547F.
8. The second MultiBoot image is loaded starting at the highest PROM address, which is at hexadecimal 0xFFFFF for a 1Mbyte PROM. The image is loaded downward (decrementing address) and ends at hexadecimal address 0xBAB80.

Spartan-3E MultiBoot using Xilinx Platform Flash PROMs

While the Spartan-3E MultiBoot feature was primarily designed to leverage commodity parallel NOR Flash PROMs, it is also possible to use Xilinx® Parallel Platform Flash PROMs, specifically the XCFxxP PROM family. The final 'P' in the product name indicates the Parallel version. See XAPP483 for additional details.

- **XAPP483: Multiple-Boot using Platform Flash PROMs**
http://www.xilinx.com/support/documentation/application_notes/xapp483.pdf

Extended Spartan-3A Family MultiBoot

Starting with the Spartan-3A FPGA family, MultiBoot is expanded and enhanced to provide additional flexibility and capabilities.

- Extended Spartan-3A family MultiBoot supports multiple FPGA bitstream images beyond just the two images supported on Spartan-3E FPGAs.
- The maximum number of FPGA images supported is limited either by the size of the configuration PROM or the total number of address bits.
 - ◆ BPI mode supports up to 26 address bits, which addresses up to 64M bytes or 512M bits.
 - ◆ SPI mode supports up to 24 address bits, which addresses up to 16M bytes or 128M bits.
 - ◆ An Extended Spartan-3A family bitstream, depending on device logic density, ranges between approximately 0.5M to 11.5M bits per FPGA.
- Extended Spartan-3A family FPGAs can MultiBoot between different configuration modes. For example, the FPGA can initially configure from parallel Flash using BPI mode, then MultiBoot to a configuration image stored in SPI serial Flash using Master SPI mode. For the Spartan-3AN Engineering Samples, see the [Spartan-3AN Errata](#) for limitations on MultiBoot after configuring from internal SPI Flash.
- The initial configuration image is always located at address 0, regardless of configuration mode.
- Subsequent MultiBoot images can be located anywhere in memory, aligned to a byte location, with some restrictions.
 - ◆ If the FPGA is set to wait for the Digital Clock Managers (DCMs) to lock before finishing configuration, then there must be sufficient padding between images to allow for this time. The padded region can contain data, but it cannot contain a valid configuration synchronization word.
 - ◆ Individual bitstream images may be aligned to a sector or page boundary within the attached Flash memory device.
- A built-in configuration watchdog timer prevents a MultiBoot operation from “hanging” on an invalid FPGA configuration image.
 - ◆ If no synchronization word is detected within the watchdog time-out period, the FPGA automatically returns to and reloads the default, initial configuration image.

Specifying the Next MultiBoot Configuration Address

The initial FPGA configuration bitstream is always loaded at address 0 from the attached configuration PROM, regardless of mode. For MultiBoot operations, there are two methods for the FPGA application to load the address of the next MultiBoot configuration image.

1. **Fixed, Known Address:** If the next address is predefined and known at design time, the next MultiBoot address can be preloaded within the current FPGA bitstream using the [next_config_addr](#) bitstream generator (BitGen) option. The parallel NOR Flash address or the SPI serial Flash address is specified as an seven-character hexadecimal string.

```
next_config_addr = 0x0000000
```

2. **Variable or Calculated Address:** The FPGA application itself can supply the address of the next MultiBoot image by writing a command sequence to the FPGA's ICAP_SPARTAN3A design primitive.

Required Data Spacing between MultiBoot Images

Extended Spartan-3A family MultiBoot addressing is flexible enough to allow a bitstream to begin at any byte boundary. However, there are a few practical limitations, based on specific application requirements.

Flash Sector, Block, or Page Boundaries

Extended Spartan-3A family FPGAs load MultiBoot configuration images from an external Flash PROM. All Flash PROMs have an internal memory architecture that arranges the memory into sectors, blocks, or pages. Nearly all PROMs have multiple sectors. Some architectures provide additional granularity, splitting a sector into smaller blocks, or even smaller still, pages.

Ideally, an Extended Spartan-3A family MultiBoot configuration image should be aligned to a sector, block, or page boundary. The specific requirement depends on the Flash PROM architecture. If the smallest erasable element in the Flash PROM is a sector, then align the FPGA bitstream to a sector boundary. This way, one FPGA bitstream can be updated without affecting others in the PROM.

Additional Memory Space Required for DCM_WAIT

An Extended Spartan-3A family application may contain one or more Digital Clock Managers (DCMs). Each DCM provides an option setting that, during configuration, causes the FPGA to wait for the DCM to acquire and lock to its input clock frequency before the DCM allows the FPGA to finish the configuration process. The lock time, which is specified in the Extended Spartan-3A family data sheet, depends on the DCM mode, and the input clock frequency.

Even if the FPGA is waiting for one or more DCMs to lock before completing configuration, the FPGA's configuration controller continues searching for the next synchronization word. If two adjacent MultiBoot images are placed one immediately following the other, and the first FPGA bitstream contains a DCM with the DCM_WAIT option set, then potential configuration problems can occur. If the controller sees the synchronization word in the second FPGA bitstream before completing the current configuration, it starts interpreting data from the second bitstream. However, the FPGA's configuration logic may complete the current configuration even though the FPGA has read data from the second bitstream.

Caution! FPGA applications that use the DCM_WAIT option on a DCM must ensure sufficient spacing between Extended Spartan-3A family MultiBoot configuration images!

Spacing MultiBoot bitstreams sufficiently apart in memory prevents the FPGA from ever seeing the second synchronization word. The following are some points to consider.

- Is the DCM_WAIT option being used in the FPGA application? The potential issue only occurs if DCM_WAIT=TRUE.
- Which DCM outputs are used? There are two lock time specifications in the data sheet: LOCK_DLL specifies the lock time for the DLL outputs from the DCM (CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKDV) and LOCK_DFS specifies the lock time for the DFS outputs (CLKFX, CLKFX180).

- The specified lock time also depends on the input clock frequency. Again, consider both the DLL and DFS specifications. The lock time is longest at 5 ms for input frequencies below 15 MHz
- The amount of spacing between bitstreams also depends on the *ConfigRate* bitstream option setting in the bitstream and the maximum frequency of CCLK at that *ConfigRate* setting.
- The number of spacing bits required also depends on the configuration mode. The SPI Flash mode receives one bit per clock while the BPI mode receives eight bits or one byte per clock.

Example

A Spartan-3A MultiBoot application includes an FPGA bitstream that contains at least one DCM with the DCM_WAIT option set TRUE. The FPGA application uses a DLL output from the DCM. The input clock frequency to the DCM is 33 MHz. The data sheet lock time specification (LOCK_DLL) for DCM clocks faster than 15 MHz is 600 μ s. The FPGA bitstream has the *ConfigRate* option set to 25. According to [DS529: Spartan-3A FPGA Family Data Sheet](#), setting *ConfigRate:25* means that CCLK will never have a period shorter than 45 ns. The MultiBoot application configures from an SPI serial Flash.

Dividing the 600 μ s lock time by the 45 ns CCLK period yields 13,334 clock cycles. In SPI mode, the FPGA receives one bit per clock cycle. Consequently, under these conditions, two MultiBoot configuration images must be placed more than 13,334 *bit* locations from each other in memory.

If the FPGA configured from parallel Flash, then the FPGA receives 8 bits per clock cycle. Consequently, the application must space the two configurations apart by more than 13,334 *byte* locations, which is equivalent to 106,672 bits.

The memory space between two configuration images can contain data as long as it does not contain a valid Spartan-3A configuration synchronization word, shown in [Table 12-3, page 244](#). Alternatively, leave the space between locations programmed with 0xFF, which is the same state as an unprogrammed Flash location.

MultiBoot Command Sequence (ICAP Example)

The following steps are required to initiate a MultiBoot reconfiguration event from within the FPGA application using the ICAP design primitive. MultiBoot events can also be issued via JTAG, the Slave Serial, or the Slave Parallel (SelectMAP) interface. The specific bit sequences supplied below are for the ICAP interface, but the same general approach also applies for the other interfaces.

Caution! By Xilinx convention, data bit D0 is the most-significant bit. In many other conventions, data bit D7 is the most-significant bit. In the application, ensure that the correct value is being written to the ICAP interface, either by adjusting the data written to the interface or by reversing the wiring connections to the interface.

Design Specification

- Enable the ICAP interface, which is required for MultiBoot functionality, in the FPGA configuration bitstream, using the *ICAP_Enable:Auto* or *ICAP_Enable:Yes* bitstream generator option setting.

Caution! The ICAP interface will not be available until the first configuration has completed startup, including the End of Startup cycle. Allow a few additional clock cycles after the end of configuration before beginning the ICAP MultiBoot sequence. See “Startup” in [Chapter 12](#).

- If the next address is fixed and already known at design time, preload the [GENERAL1](#) and [GENERAL2](#) registers with default values by setting the [next_config_addr](#) bitstream generation (BitGen) option.

FPGA Application Run Time

- Issue the synchronization start word to the ICAP interface.
- If the FPGA application calculates the next MultiBoot configuration start address, load the [GENERAL1](#) and [GENERAL2](#) registers via ICAP with the start address of the next MultiBoot configuration image.
- **OPTIONAL:** If rebooting from a different configuration source, write the appropriate values to the MODE_REG register. See “[Switching between MultiBoot Configuration Memory Types](#),” page 286 for more information.
- Issue the [REBOOT](#) command to the [CMD](#) register.
- Issue a NoOp command to the ICAP interface.

MultiBoot from an Address Preloaded during Configuration

[Table 14-2](#) shows the command sequence to initiate a MultiBoot event, assuming the following.

- The [GENERAL1](#) and [GENERAL2](#) registers are preloaded during configuration via the [next_config_addr](#) bitstream generation (BitGen) option.
- The next MultiBoot address is in the same memory originally used to configure the FPGA or the same memory used during the last MultiBoot operation.

Each 16-bit command is written as two bytes to the ICAP interface, with the high byte presented first, followed by the low byte. Note that D0 is the most-significant bit (msb) for the ICAP interface, which is the opposite direction from most processors.

Table 14-2: Command Sequence to Initiate MultiBoot from a Preloaded Address

CLK Cycle	Command	High or Low Byte	D0	D1	D2	D3	D4	D5	D6	D7	Hex
1	SYNC WORD	High	1	0	1	0	1	0	1	0	0xAA
2		Low	1	0	0	1	1	0	0	1	0x99
3	Type 1 Write CMD (1 Word)	High	0	0	1	1	0	0	0	0	0x30
4		Low	1	0	1	0	0	0	0	1	0xA1
5	REBOOT Command	High	0	0	0	0	0	0	0	0	0x00
6		Low	0	0	0	0	1	1	1	0	0x0E
7	No Op	High	0	0	1	0	0	0	0	0	0x20
8		Low	0	0	0	0	0	0	0	0	0x00

MultiBoot to a Address Specified by the FPGA Application

[Table 14-3](#) shows an example where the FPGA application specifies the address of the next MultiBoot image. This specific example is for SPI serial Flash, but parallel NOR Flash is similar with slightly different definitions of the bits written to the [GENERAL2](#) register (CLK cycles 9 and 10).

Each 16-bit command is written as two bytes to the ICAP interface, with the high byte presented first, followed by the low byte. Note that D0 is the most-significant bit (MSB) for the ICAP interface, which is the opposite direction from most processors.

The sequence in [Table 14-3](#) uses 16 steps, and consequently 16 CLK cycles and 16 memory locations. The sequence can be shortened to 12 CLK cycles by making the following simple changes.

- Align the next MultiBoot address to a 16-bit (64K) boundary and pre-assign the contents of the [GENERAL1](#) register to 0x0000 by setting the [next_config_addr](#):00000000 bitstream generator option. The next MultiBoot address is then selectable solely by writing to the [GENERAL2](#) register. This eliminates the four steps between CLK cycles 3 and 6.

Table 14-3: Command Sequence to Initiate MultiBoot to a Specified Address

CLK Cycle	Command	High or Low Byte	D0	D1	D2	D3	D4	D5	D6	D7	Hex
1	SYNC WORD	High	1	0	1	0	1	0	1	0	0xAA
2		Low	1	0	0	1	1	0	0	1	0x99
3	Type 1 Write GENERAL1 (1 Word)	High	0	0	1	1	0	0	1	0	0x32
4		Low	0	1	1	0	0	0	0	1	0x61
5	Lower 16 bits of MultiBoot Address	High	A15	A14	A13	A12	A11	A10	A9	A8	
6		Low	A7	A6	A5	A4	A3	A2	A1	A0	
7	Type 1 Write GENERAL2 (1 Word)	High	0	0	1	1	0	0	1	0	0x32
8		Low	1	0	0	0	0	0	0	1	0x81
9	Upper bits of MultiBoot Address (SPI mode example)	High	C7	C6	C5	C4	C3	C2	C1	C0	
10		Low	A23	A22	A21	A20	A19	A18	A17	A16	
11	Type 1 Write CMD (1 Word)	High	0	0	1	1	0	0	0	0	0x30
12		Low	1	0	1	0	0	0	0	1	0xA1
13	REBOOT Command	High	0	0	0	0	0	0	0	0	0x00
14		Low	0	0	0	0	1	1	1	0	0x0E
15	No Op	High	0	0	1	0	0	0	0	0	0x20
16		Low	0	0	0	0	0	0	0	0	0x00

MultiBoot using SelectMAP

ICAP is the internal mirror of the SelectMAP interface. The command sequence for MultiBoot using the SelectMAP interface is the same as for the ICAP interface. You must control CS_B/CSI_B and RDWR_B for any write to the SelectMAP interface; it is important to avoid an Abort. An Abort is triggered when RDWR_B is toggled while CS_B/CSI_B is asserted.

MultiBoot using Slave Serial

The command sequence for MultiBoot using the Slave Serial interface is the same as the ICAP command sequence except the sequence starts with the LSB first.

MultiBoot using JTAG

The easiest method for MultiBoot using the JTAG interface is with an SVF file (see [XAPP503 SVF and XSVF File Formats for Xilinx Devices](#)). The command sequence is the same as the ICAP command sequence and can be loaded into configuration memory using a CFG_IN instruction. The following is an example of how to construct the CFG_IN instruction.

```
// MultiBoot command sequence
//FFFF => ffff Dummy word shifted in first
//AA99 => 1010 1010 1001 1001 => 1001 1001 0101 0101 => 9955 SYNC
//30A1 => 0011 0000 1010 0001 => 1000 0101 0000 1100 => 850a Type 1 Write to CMD
//000E => 0000 0000 0000 1110 => 0111 0000 0000 0000 => 7000 REBOOT command
//2000 => 0010 0000 0000 0000 => 0000 0000 0000 0100 => 0004 NOOP
//2000 => 0010 0000 0000 0000 => 0000 0000 0000 0100 => 0004 NOOP
// Append the commands and put them into an SDR for CFG_IN
// This will load the command sequence to config memory in the same way that the ICAP would
// For SDR command, 24 hex characters: 24 x 4 = 96 bit shift
// Loading device with a `cfg_in` instruction
SIR 6 TDI (05);
// Loads the instruction to the IR
//SDR 96 TDI (0004 0004 7000 850a 9955 ffff) SMASK (ffff ffff ffff ffff ffff ffff ffff ffff)
SDR 96 TDI (000400047000850c9955ffff) SMASK (ffffffffffffffffffffffffffffffff);
STATE RESET;
```

MultiBoot Registers

Generally, there are three ICAP registers involved in a MultiBoot application. The address of the next MultiBoot bitstream is stored in registers [GENERAL1](#) and [GENERAL2](#), although they can be preloaded via BitGen option [next_config_addr](#). To trigger a MultiBoot event, the FPGA application must issue a [REBOOT](#) command using the [CMD](#) register.

Next MultiBoot Start Address ([GENERAL1](#), [GENERAL2](#))

The start address of the next MultiBoot configuration image is stored in two 16-bit registers, called [GENERAL1](#) and [GENERAL2](#). These registers can also be preloaded using the bitstream generation (BitGen) option [next_config_addr](#).

The [GENERAL1](#) and [GENERAL2](#) registers are not cleared or modified during a MultiBoot event.

The [GENERAL1](#) register holds the lower 16 bits of the next MultiBoot address, as shown in [Table 14-4](#).

Table 14-4: **GENERAL1 Register Definition**

GENERAL1

Address 01_0011 (0x13)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

The context of the [GENERAL2](#) register depends on whether the next MultiBoot address is in an external parallel NOR Flash (BPI mode) or an external SPI serial Flash (SPI mode).

In BPI mode, the [GENERAL2](#) register contains the upper 10 bits of the 26-bit BPI address, as shown in [Table 14-5](#). The upper six bits of the register are reserved.

Table 14-5: GENERAL2 Register Definition for BPI Mode Options

GENERAL2

Address 01_0100 (0x14)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
—	—	—	—	—	—	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16

In SPI mode, the GENERAL2 register contains the upper 8 bits of the 24-bit SPI address, as shown in Table 14-6. The upper eight bits of the register contain the specific byte-wide read command for the attached external SPI serial Flash device.

Table 14-6: GENERAL2 Register Definition for SPI Mode Options

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI Flash Read Command								Upper Byte of 24-bit SPI Read Address							
C7	C6	C5	C4	C3	C2	C1	C0	A23	A22	A21	A20	A19	A18	A17	A16

Command Register (CMD)

Configuration commands control the operation of the configuration state machine. Each command consists of five bits, as shown in Table 14-7.

Table 14-7: CMD Register Definition

CMD															
Address 00_0101 (0x05)															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved											REBOOT Command				
											0	1	1	1	0

Only one command is required for MultiBoot operations, the REBOOT command, which is binary 01110.

Configuration Mode Register (MODE_REG)

The configuration mode register, [MODE_REG](#), defines which configuration mode the FPGA uses upon the next MultiBoot trigger event. The [NEW_MODE](#) bit defines whether the FPGA uses the M[2:0] mode settings defined by the M[2:0] pins of the FPGA or whether the FPGA uses the settings defined by the [BOOTMODE](#) bits. Setting [NEW_MODE](#) = 1 allows the FPGA to MultiBoot to a different type of attached memory. This is an advanced option that is not recommended for most situations. Note that SPI fallback is not supported when using [NEW_MODE](#)=1.

[Table 14-8](#) describes the bit options available in the [MODE_REG](#) register.

Table 14-8: MODE_REG Bit Options

MODE_REG			
Address 01_0101 (0x15)			
Name	Bit(s)	Description	Default
Reserved	[15:7]	Reserved	0
NEW_MODE	6	0: Sample M[2:0] pins and, if in SPI mode, the VS[2:0] pins to determine MultiBoot configuration mode. 1: Use BOOTMODE value to determine MultiBoot configuration mode.	0
BOOTMODE	5:3	Define the M[2:0] configuration mode select setting for the next MultiBoot event. Requires NEW_MODE = 1. Available options are identical to FPGA mode pin settings, M[2:0], shown in Table 2-1, page 50 . Default SPI mode is not recommended since it does not support fallback when using NEW_MODE =1.	001 (SPI)
Reserved	2:0	Reserved	111

Generating an Extended Spartan-3A Family MultiBoot PROM Image using iMPACT

Note: For Spartan-3AN, see also “[Preparing an In-System Flash Programming File](#),” [page 220](#).

The iMPACT programming software provides a graphical, step-by-step approach to create a MultiBoot PROM file. Similar functionality is also available from the command line or via scripts using the PROMGen utility, shown in [Figure 14-10](#). Follow the steps outlined below to create an Extended Spartan-3A family MultiBoot PROM file using the iMPACT software. [Figure 14-11, page 279](#) shows the Extended Spartan-3A family MultiBoot design used in the following example.

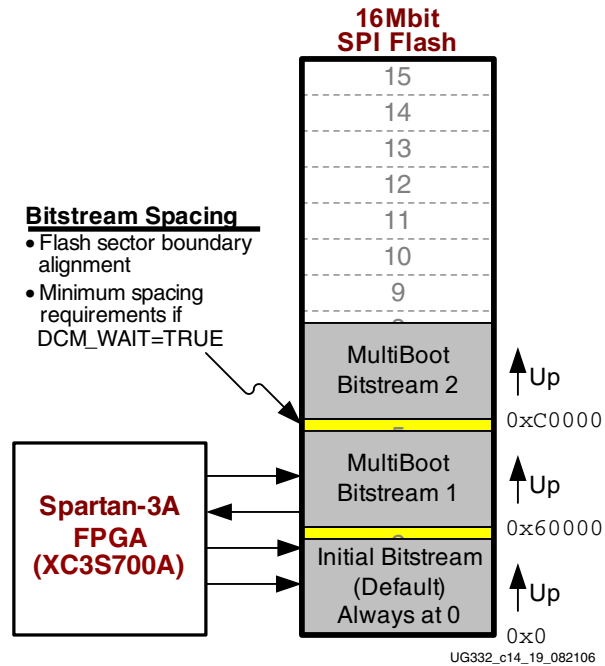


Figure 14-11: Spartan-3A MultiBoot Example using XC3S700A and SPI Flash

1. Invoke the iMPACT programming software.
2. As shown in Figure 14-3, choose **Prepare a PROM File**.

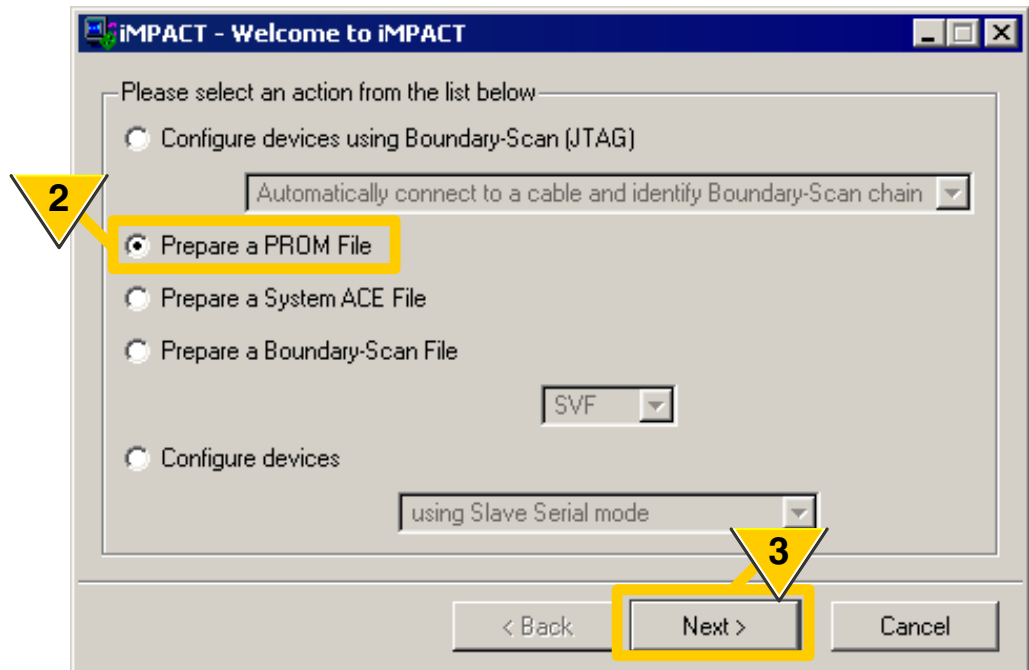
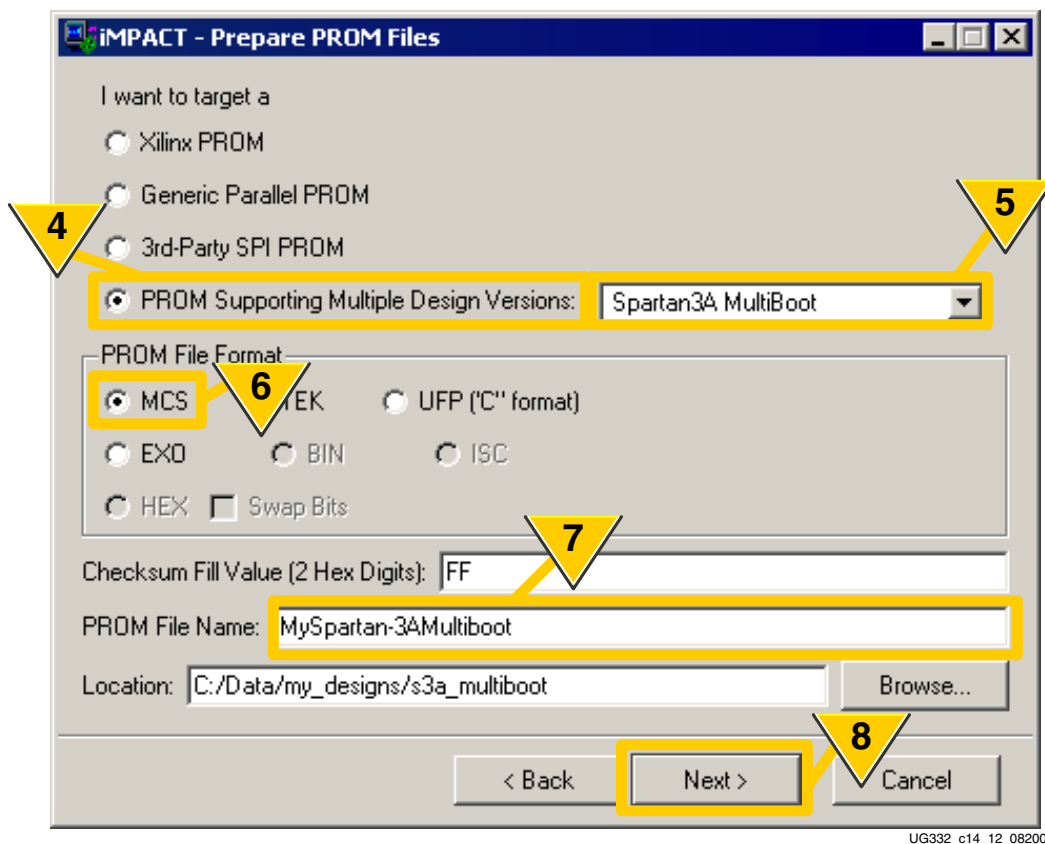


Figure 14-12: Prepare a MultiBoot PROM Image

3. Click Next.

- As shown in [Figure 14-13](#), target a **PROM Supporting Multiple Design Revisions**.



UG332_c14_12_082006

Figure 14-13: Select a PROM Supporting MultiBoot for Extended Spartan-3A Family FPGAs

- Choose the **Spartan3A MultiBoot** method.
- Select a **PROM File Format**. The MCS format is supported by a variety of programmers, but other options are available.
- Enter a **PROM File Name**.
- Click **Next**.

- As shown in [Figure 14-14](#), choose whether to create an Extended Spartan-3A family MultiBoot image for SPI serial Flash or for parallel NOR Flash, using BPI mode. This example uses an SPI PROM.

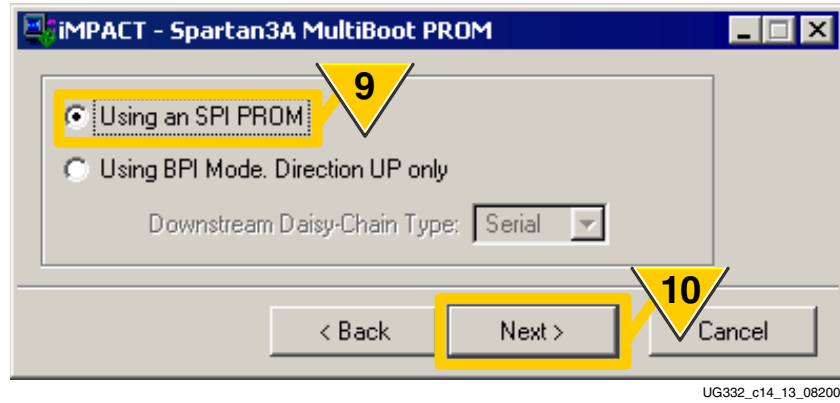


Figure 14-14: SPI or Parallel Flash PROMs are Supported

- Click Next.
- As shown in [Figure 14-15](#), Select SPI PROM Density, which is always specified in bits. This example uses a 16 Mbit PROM.

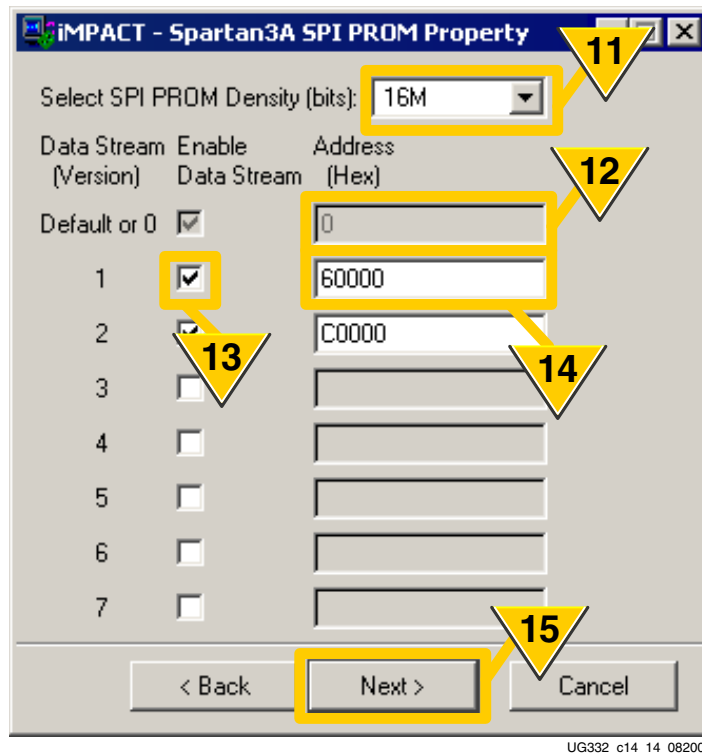


Figure 14-15: Enter a PROM Density and Specify MultiBoot Image Start Locations

- The initial MultiBoot image is *always* loaded starting at address 0.
- To add additional images, check **Enable Data Stream**.
- Specify the starting address of each MultiBoot image using hexadecimal notation.

- ◆ What is the size of the FPGA configuration bitstream? Is the bitstream for a single FPGA or a multi-FPGA daisy chain? Extended Spartan-3A family FPGAs do support daisy chains when using MultiBoot.
- ◆ What are the page or sector boundaries of the Flash device? Ideally, the FPGA bitstream should start on a Flash sector boundary.
- ◆ If using the DCM_WAIT option on a Digital Clock Manager (DCM) with the FPGA application, is there enough additional spacing between images to accommodate the extra lock time?

An uncompressed Spartan-3A XC3S700A FPGA configuration bitstream requires 2,732,640 bits. Dividing that number by eight provides the required number of bytes, 341,580 bytes. Divide the number of bytes by 1,024 to determine the number of kilobytes, or 333.57K. The Atmel AT45DB161D serial Flash uses 128Kbyte sectors. Consequently, a single XC3S700A configuration bitstream occupies ~2.6 sectors. The first bitstream always starts at address 0. The next Extended Spartan-3A family MultiBoot image should be placed on a following Flash sector boundary. The next available boundary after the first configuration image begins at address hexadecimal address 0x60000. Place the second bitstream at this address or any subsequent sector boundary. With an image at 0x60000, a third image starts at 0xC0000.

15. Click **Next**.
16. The iMPACT software summarizes the current settings, as shown in [Figure 14-16](#). Click **Finish** to continue.

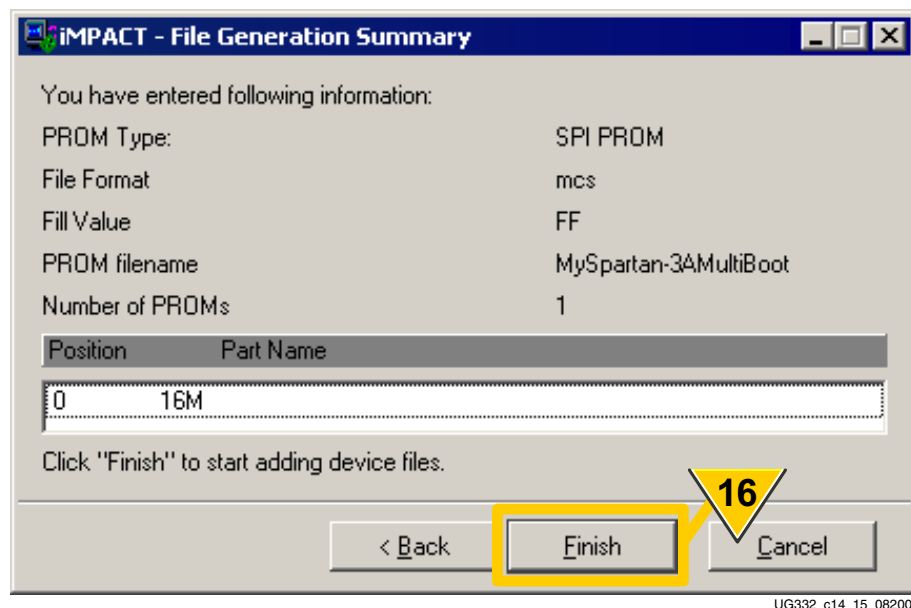


Figure 14-16: Confirm PROM Settings

17. As shown in [Figure 14-17](#), add the initial FPGA bitstream. This is the design that is loaded into the FPGA at power-up, or whenever the PROG_B pin is pulsed Low. This is also the default bitstream that is automatically loaded if the Configuration Watchdog Timer (CWDT) expires during a MultiBoot operation.

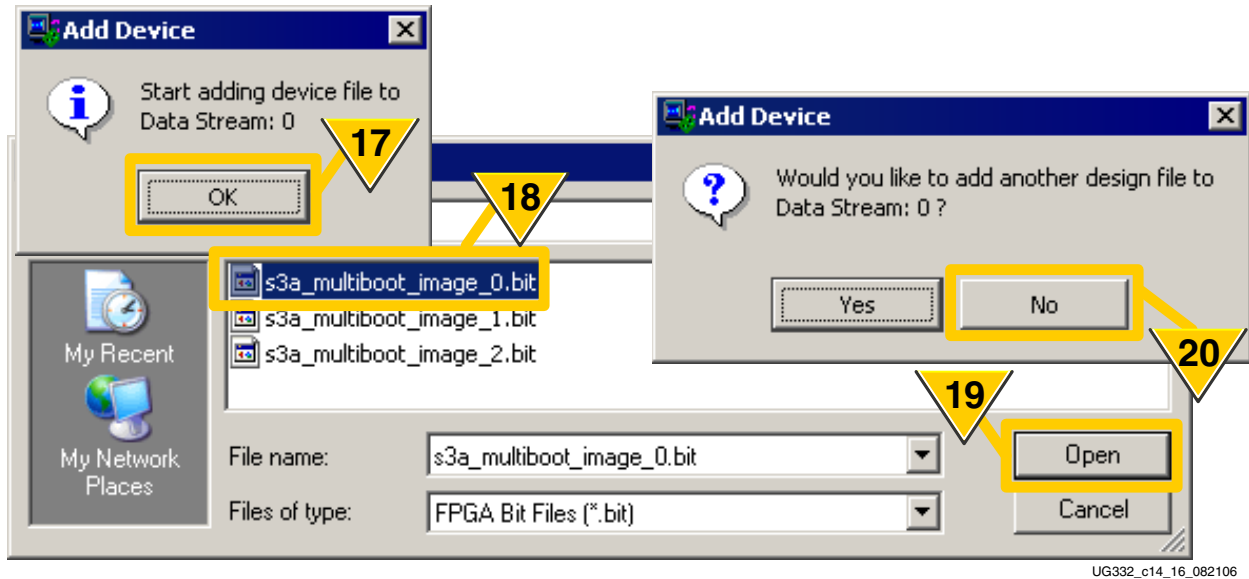


Figure 14-17: Select the First (Default) Configuration Image

18. Select the first FPGA configuration bitstream.
19. Click **Open**.
20. Click **No**.
21. Perform Steps 17 through 20, but this time for the second FPGA configuration bitstream.
22. As shown in Figure 14-18, start adding the third FPGA configuration bitstream.

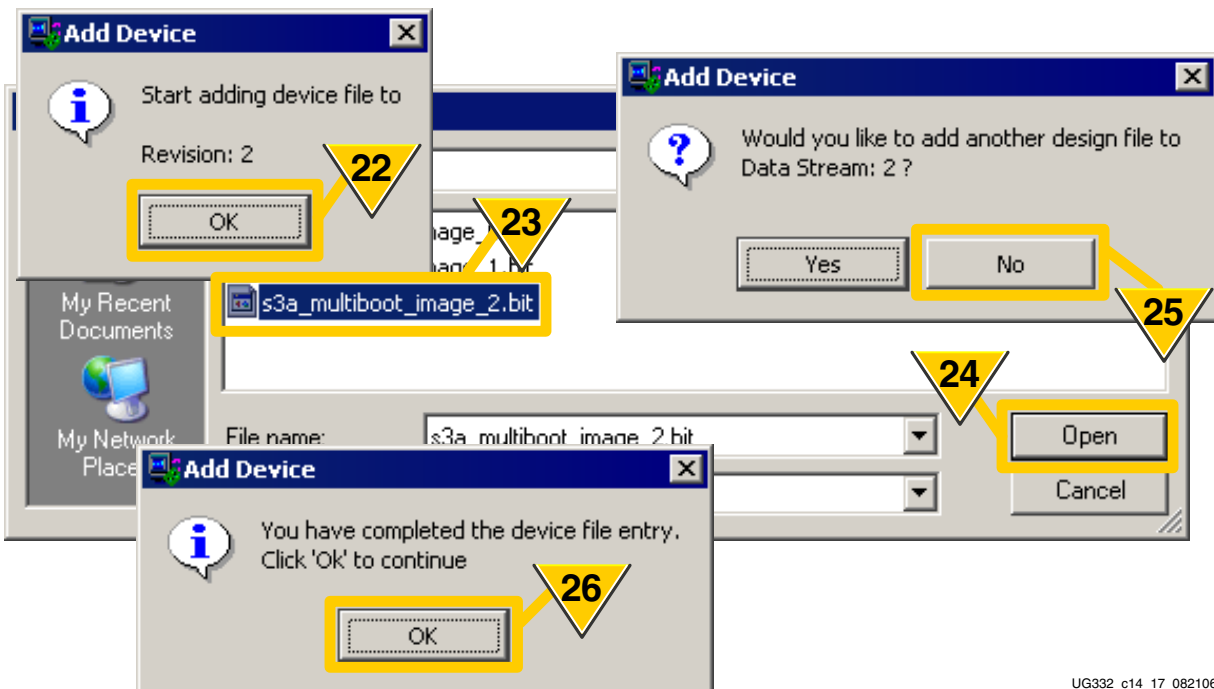


Figure 14-18: Select the Third MultiBoot Configuration Image

23. Select the third FPGA configuration bitstream.
24. Click **Open**.
25. Click **No**.
26. File selection is complete. Click **OK**.
27. As shown in [Figure 14-19](#), the iMPACT software reports how much of the PROM is consumed by the FPGA configuration bitstream files. Double-click **Generate File**.

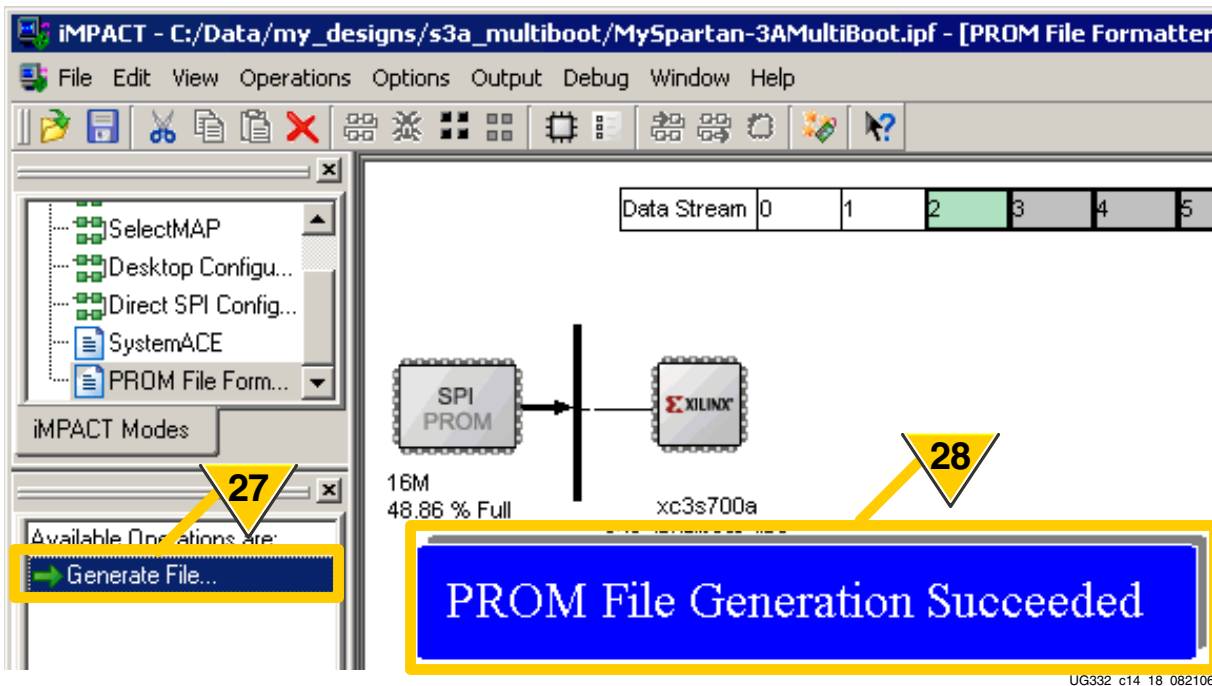


Figure 14-19: Generate the PROM File Using the Specified Parameters

28. The iMPACT software successfully generates a PROM file using the name specified in [Step 7](#) with the format and file extension specified in [Step 6](#). The file is created in the current directory. A “PROMGen Report File” is also created.

Configuration Fallback

The Spartan-3A, Spartan-3AN, and Spartan-3A DSP families include logic to automatically “fallback” and re-start configuration after a configuration failure. These features are particularly useful when providing the FPGA with “live” Flash updates. This feature is especially useful in MultiBoot applications, including applications that will be updated in the field, to help protect against failed updates. Two events may cause the device to fallback.

1. A timeout without seeing the configuration synchronization word AA99 (fallback always enabled)
2. A CRC error (fallback optional)

Configuration Watchdog Timer (CWDT) and Fallback

Extended Spartan-3A family FPGAs contain a configuration watchdog timer (CWDT). The CWDT provides protection against errant configuration operations such as the following.

- Configuration or MultiBoot operations to an invalid start location
- Configuration or MultiBoot operations to a valid start location, but loaded with an incomplete or partially valid configuration bitstream.

The CWDT is a 16-bit counter, clocked by the [CCLK](#) configuration clock signal. Upon any FPGA configuration operation, be it from power-up, a [PROG_B](#) pulse, or a MultiBoot event, the CCLK clock begins operation at its lowest [ConfigRate](#) setting, which is approximately 1 MHz. The CWDT expires 64K clock cycles after the start of configuration, or in approximately 65 ms.

If, during a MultiBoot operation, the FPGA does not see a valid configuration synchronization word before the CWDT expires, then the FPGA will automatically *fallback* to the default bitstream located at address 0. The FPGA automatically reconfigures from the default bitstream, even resending the appropriate SPI Flash read command if using the SPI configuration mode.

CRC Error and Fallback

Similarly, an Extended Spartan-3A family FPGA can also recover from a MultiBoot operation to a bitstream that has a correct synchronization word, but that eventually issues a CRC error for some reason. Set the [Reset_on_err:Yes](#) bitstream option to cause the FPGA to automatically re-initialize and retry the first configuration at address 0 should a CRC error occur. CRC checking must also be enabled, which is the default.

Note that the [Reset_on_err:Yes](#) bitstream option is included in the first part of the next bitstream. For best support of fallback, [Reset_on_err:Yes](#) should be set in each bitstream used in a MultiBoot application. If a synchronization word pattern is found but the [Reset_on_err:Yes](#) bitstream option is not seen due to corrupt data, the device will not fallback but will indicate a configuration error on [INIT_B](#) and wait for reconfiguration. Also note that the CRC check is done at the end of the bitstream; therefore, if the end of the bitstream is missing or corrupted the device will not perform the CRC check and automatic fallback will not occur. Fallback helps automatically recover from minor corruptions in otherwise valid bitstreams, but a significantly corrupted or invalid configuration file may not permit fallback to occur. Therefore it is still important to make sure that a valid configuration image is being read, while using fallback to recover from minor errors.

Fallback Limited to 3 Additional Tries

In BPI and SPI modes, if reconfiguration has failed and done fallback three times, then on the fourth failure the FPGA halts and drives the [INIT_B](#) pin Low. Pulsing the [PROG_B](#) pin or cycling power restarts the configuration process from the beginning.

The counter that keeps track of the failed configurations is reset only when [PROG_B](#) is pulsed or power is cycled; it is not reset after a successful configuration. The FPGA will stop attempting configuration if the initial design is good but the MultiBoot bitstream is bad, after the fourth attempt at MultiBoot. Note that when configuring via SPI or BPI modes and using the [Reset_on_err:Yes](#) bitstream option, any combination of successful and failed configurations, over any period of time, will halt after the fourth failed configuration, and require assertion of [PROG_B](#) or power cycling to reconfigure. It is good design practice to have the ability to assert [PROG_B](#) to reset configuration if necessary.

Advanced Capabilities

Switching between MultiBoot Configuration Memory Types

The Extended Spartan-3A family MultiBoot feature also provides the advanced capability to jump between configuration modes and hence different types of external memory. This feature is not recommended for most applications because switching to the SPI mode does not support the Configuration Watchdog Timer.

As shown in [Figure 14-20](#), during a MultiBoot event, the Extended Spartan-3A family internal configuration controller determines which FPGA configuration mode to execute. By default, the FPGA uses the mode select values physically defined on the FPGA's M[2:0] mode select pins. Similarly, if the FPGA mode pins specify the Master SPI Flash mode, then the controller uses the Read Command associated with the variant select values, VS[2:0], defined by the associated FPGA pins.

However, by setting the control bit `NEW_MODE = 1` in the `MODE_REG` register, the internal configuration controller uses the configuration mode specified by the `BOOTMODE` bits. If `BOOTMODE = 001` to specify the Master SPI Flash mode, then the controller uses the Read Command specified in the higher-order byte {15:8} of the `GENERAL2` register, and the remaining lower-order byte of the `GENERAL2` register provides the upper 8 bits of the 24-bit MultiBoot address. If `BOOTMODE = 010` to specify the BPI Flash mode, then the lower 10 bits of the `GENERAL2` register become the upper 10 bits of the 26-bit BPI MultiBoot address. In both cases, the `GENERAL1` register provides the lower 16 bits of the MultiBoot address (see “[Next MultiBoot Start Address \(GENERAL1, GENERAL2\)](#),” page 276).

Caution! Fallback is not supported when switching to the SPI mode during MultiBoot.

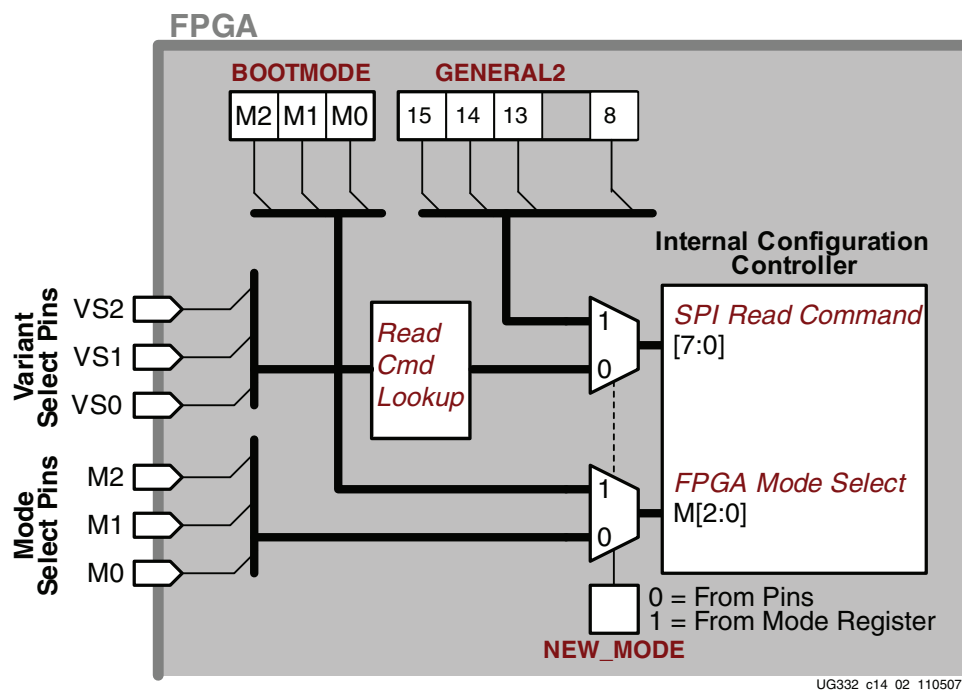


Figure 14-20: Extended Spartan-3A Family MultiBoot Configuration Mode Control

MultiBoot Design Examples

The reference designs for the Spartan-3A/3AN FPGA Starter Kit board include MultiBoot design examples:

http://www.xilinx.com/products/boards/s3astarter/reference_designs.htm

[XAPP468](#) *Fail-safe MultiBoot Reference Design* includes algorithms to test bitstream integrity and to select the bitstream image to load.

Protecting FPGA Designs

Similar to a processor, a Spartan®-3 generation FPGA receives its configuration information, i.e., its application program, from an external memory source. The exposed external interface makes both processor code and FPGA bitstreams potentially vulnerable to copying or cloning.

Unlike a processor, there are no simple “reverse assemblers” for FPGA applications. Processors have a defined, fixed instruction set and instruction length, making a reverse assembler for a processor a straightforward task. However, reverse engineering an entire FPGA design and then converting it to a human-understandable form is exceedingly difficult. An FPGA configuration bitstream contains millions of interrelated bits. Furthermore, the Xilinx® bitstream format is both proprietary and confidential.

While reverse engineering an FPGA bitstream is difficult, directly copying an FPGA bitstream without understanding its underlying function, is rather straightforward. This chapter describes the available, low-cost solutions to protect a design against cloning and even to protect an intellectual property (IP) core implemented within an FPGA.

This chapter covers the following design security topics.

- “Basic FPGA Hardware-Level Security Options,” page 289
- “Approaches to Design Security,” page 292
- “Extended Spartan-3A Family Unique Device Identifier (Device DNA),” page 294
- “Authentication Design Examples,” page 297
- “U.S. Legal Protection of FPGA Configuration Bitstream Programs,” page 306

Basic FPGA Hardware-Level Security Options

Spartan-3 generation FPGAs provide advanced debugging capabilities via a function called Readback. Similarly, the FPGA generally allows full access to all configuration operations. However, for security-conscious applications, the Readback function and configuration operations, especially via JTAG, provide a potential point of attack.

Fortunately, the FPGA bitstream optionally restricts access to configuration and readback operations. By default, there are no restrictions and the JTAG port is always active, providing access to configuration and Readback.

The SelectMAP configuration interface, which can also be used to perform Readback, is disabled by default and is not available unless specifically enabled by setting the *Persist:Yes* bitstream option.

The only way to remove a security setting in a configured FPGA is to clear the FPGA program by asserting PROG_B or cycling power.

Spartan-3 and Spartan-3E Security Levels

Table 15-1 shows the available security levels on Spartan-3 and Spartan-3E FPGAs. Extended Spartan-3A family FPGAs provide an extra level, as shown in Table 15-2.

Table 15-1: Spartan-3 and Spartan-3E Security Levels

Security Level	Description
None	Default. Unrestricted access to all configuration and Readback functions
Level1	Disable all Readback functions from either the SelectMAP or JTAG port.
Level2	Disable all configuration and Readback functions from all configuration and JTAG ports.

Extended Spartan-3A Family Security Levels

Extended Spartan-3A family FPGAs provide an additional security level, as shown in Table 15-2. Readback can be optionally disabled completely or disabled except for internal access from the FPGA application via the Internal Configuration Access Port (ICAP).

Table 15-2: Extended Spartan-3A Family BitGen Security Levels

Security Level	Description
None	Default. Unrestricted access to all configuration and Readback functions
Level1	Disable all Readback functions from both the SelectMAP or JTAG ports. Readback via the Internal Configuration Access Port (ICAP) allowed.
Level2	Disables all Readback operations on all ports.
Level3	Disable all configuration and Readback functions from all configuration and JTAG ports.

Setting the Security Level in the Bitstream

There are two ways to set the security level in the bitstream, either from the ISE® software Project Navigator or from the BitGen command-line utility.

ISE Software Project Navigator

Set the security level in the FPGA bitstream, as shown in Figure 15-1.

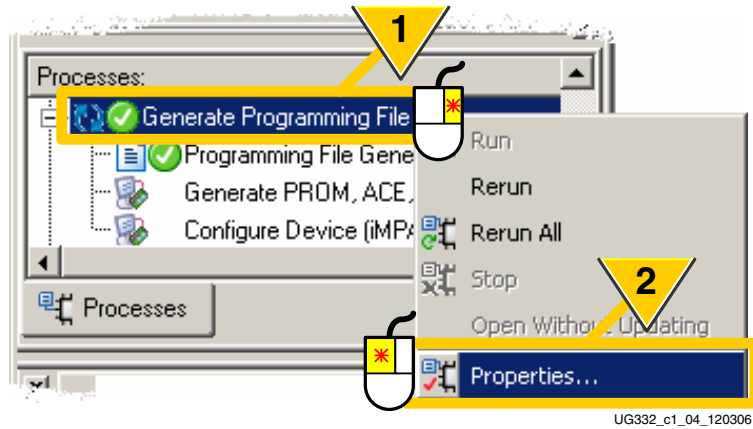


Figure 15-1: Setting Bitstream Generator Options from ISE Project Navigator

1. Right-click **Generate Programming File**.
2. Select **Properties**.

From the Process Properties dialog box shown in Figure 15-2, set the following options.

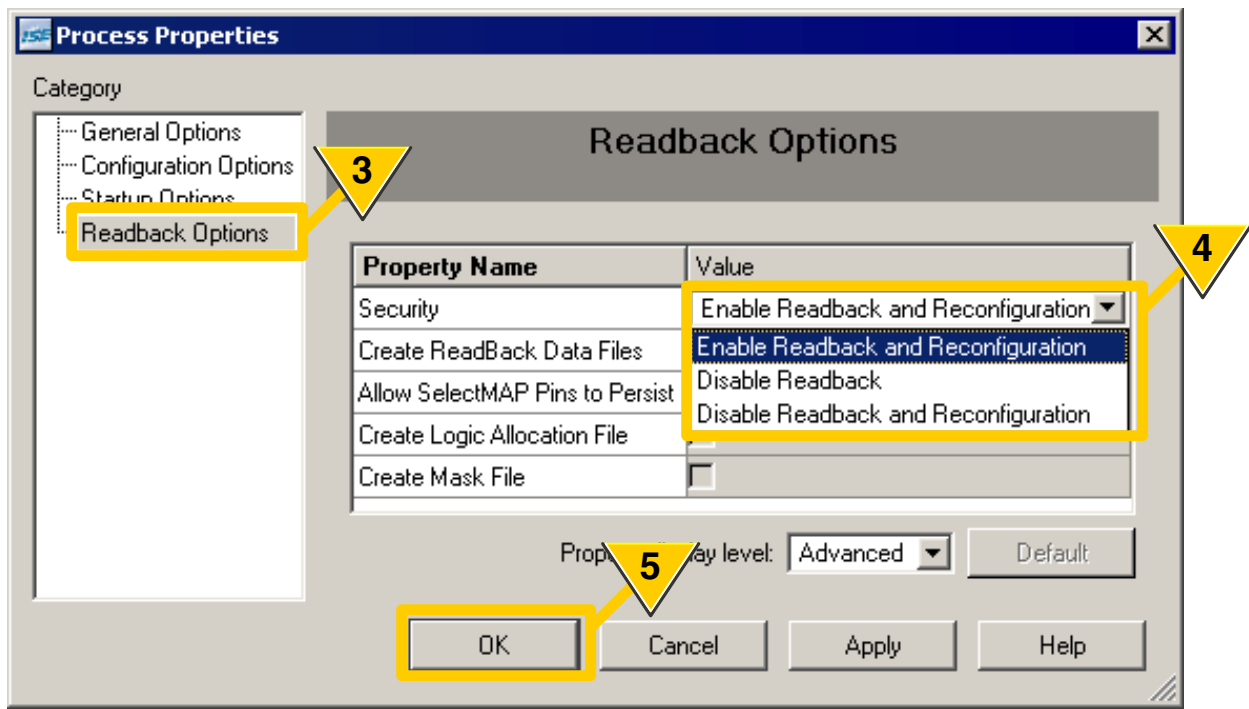


Figure 15-2: Bitstream Generator Security Options

3. Select the **Readback Options** category.
4. Choose the **Security** level value that best meets the needs of the application.

Table 15-3: Relation between ISE Project Navigator and BitGen Options

Value	Spartan-3/ Spartan-3E FPGAs	Spartan-3A/3AN Spartan-3A DSP FPGAs
Enable Readback and Reconfiguration (default)	None	None
Disable Readback	Level1	Level1
Disable Readback and Reconfiguration	Level2	Level3

BitGen Command-Line Utility

The security options are also available via the BitGen command-line utility, as shown below. The available Security options are provided in [Table 15-1](#) or [Table 15-2](#), depending on the Spartan-3 generation FPGA family used. [Table 15-3](#) shows how the options entered via the ISE Project Navigator relate to the BitGen command-line options. An example that disables Readback is provided below.

```
bitgen -g Security:Level1 [remaining options]
```

Approaches to Design Security

Xilinx programmable logic devices incorporate a variety of approaches to design security, as summarized in [Table 15-4](#). Xilinx employs each of these security options in different product families. The Spartan-3 generation FPGAs introduce a new option called Authentication, which is described throughout the remainder of this chapter.

Table 15-4: Programmable Logic Security Options Compared

	Security Bits	Encryption	Authentication
Xilinx product family that uses this security option	Xilinx CPLDs	Virtex®-II, Virtex-II Pro, Virtex-4, Virtex-5 FPGAs	Spartan-3A/3AN/ 3A DSP FPGA (but variations possible in Spartan-3/3E FPGAs)
Is bitstream or programming file visible after being secured?	No	No, only in encrypted form	Yes, but cannot be used except in an authenticated system
Does security method provide bitstream (design) security?	Yes	Yes	Yes
What happens when an unauthorized or unencrypted bitstream loaded into FPGA?	N/A	Does not configure	Behavior defined by FPGA application (see “Handling Failed Authentications”)
Does the security method provide an option to secure application data?	No	No	Yes
Does the security method provide an option to provide “Digital Rights Management”	No	No	Yes
Technical Limitations	Requires a large amount of on-chip nonvolatile memory	Key management	Requires logic in the FPGA application to authenticate design

Security Bits

Complex PLD (CPLD) designs are programmed into on-chip, nonvolatile memory, similar to simple microcontrollers. As such, CPLDs and microcontrollers typically offer a “security” bit or bits that locks the internal memory array, preventing the array from being read. Locking the array prevents the design from being easily copied.

Encryption

Some FPGAs employ bitstream encryption. Encryption essentially scrambles the external bitstream so that it is unusable unless loaded into an FPGA containing the correct “key” to decrypt the bitstream. The encryption circuitry is typically a dedicated embedded function on the FPGA, consuming valuable silicon area. Applications that do not use encryption pay for the feature regardless.

Encryption is considered highly secure, as implemented with battery back-up on the Xilinx Virtex, Virtex-II Pro, Virtex-4, and Virtex-5 FPGA families.

The built-in encryption circuitry only protects the FPGA bitstream and is typically not available after configuration to protect application data.

The primary downside of encryption is key management and key distribution.

Authentication

Authentication is another protection technique, widely used in a variety of applications. Authentication is distinctly different than using either “security bits” or encryption. Here are a few examples of everyday applications using authentication.

- When you access an Automated Teller Machine (ATM), you insert your bank card and authenticate your identity by entering a Personal Identification Number (PIN). If someone steals your ATM card, they cannot use it without also having your PIN number.
- When you log onto your computer network, you enter your login name and your password. The password authenticates your identity. An imposter must have both your login name and your password to access the network from your account.
- Many software programs, including the Xilinx ISE development software, require an authorization code before they operate on your computer. You can freely copy the DVD but it can only be used when unlocked by the authorization code.

To be ideally effective, authentication requires an identity or authorization code with these two essential attributes.

1. Unique
2. Not easily cloned, copied, or duplicated

Weaknesses in either of these elements potentially compromise security. For example, if someone has your ATM card and your PIN number, kiss your cash goodbye. The PIN number, once learned, is easily cloned. This is one of the reasons behind the move to biometric authentication. While it is easy to learn a simple PIN number, it is presently quite difficult to clone a human iris or a fingerprint.

Extended Spartan-3A Family Unique Device Identifier (Device DNA)

Each Spartan-3A, Spartan-3AN, and Spartan-3A DSP FPGA contains an embedded, unique device identifier. The identifier is nonvolatile, permanently programmed into the FPGA, and is unchangeable making it tamper resistant.

This identifier is called the *Device DNA*. The FPGA application accesses the identifier value using the “*Device DNA Access Port (DNA_PORT)*” design primitive, shown in Figure 15-3.

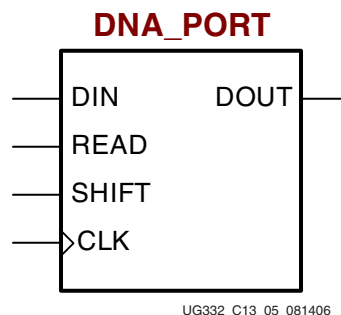


Figure 15-3: Extended Spartan-3A Family DNA_PORT Design Primitive

Identifier Value

As shown in Figure 15-4, the Device DNA value is 57 bits long. The two most-significant bits are always ‘1’ and ‘0’. The remaining 55 bits are unique to a specific Extended Spartan-3A family FPGA.

Operation

Figure 15-4 shows the general functionality of the DNA_PORT design primitive. An FPGA application must first instantiate the DNA_PORT primitive, shown in Figure 15-3, within a design.

To read the Device DNA, the FPGA application must first transfer the identifier value into the DNA_PORT output shift register. Assert the READ input during a rising edge of CLK, as shown in Table 15-5. This action parallel loads the output shift register with all 57 bits of the identifier. Because bit 56 of the identifier is always ‘1’, the DOUT output is also ‘1’. The READ operation overrides a SHIFT operation.

To continue reading the identifier values, assert SHIFT followed by a rising edge of CLK, as shown in Table 15-5. This action causes the output shift register to shift its contents toward the DOUT output. The value on the DIN input is shifted into the shift register.

Caution! Avoid a Low-to-High transition on SHIFT when CLK is High as this causes a spurious initial clock edge. Ideally, only assert SHIFT when CLK is Low or on a falling edge of CLK.

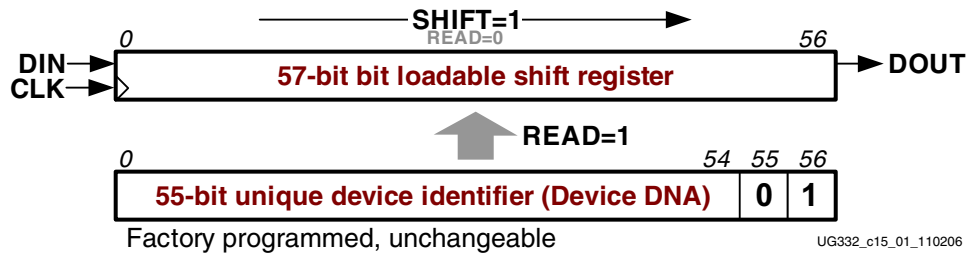


Figure 15-4: DNA_PORT Operation

If both READ and SHIFT are Low, the output shift register holds its value and DOUT remains unchanged.

Table 15-5: DNA_PORT Operations

Operation	DIN	READ	SHIFT	CLK	Shift Register	DOUT
HOLD	X	0	0	X	Hold previous value	Hold previous value
READ	X	1	X	↑	Parallel load with 57-bit ID	Bit 56 of identifier, which is always '1'
SHIFT	DIN	0	1	↑	Shift DIN into bit 0, shift contents of Shift Register toward DOUT	Bit 56 of Shift Register

Notes:

X = Don't care
 ↑ = Rising clock edge

The [Spartan-3A Starter Kit](#) board has a design example that demonstrates how to read the Device DNA value.

- Extended Spartan-3A Family Device DNA Reader Design Example
www.xilinx.com/products/boards/s3astarter/reference_designs.htm#dna_reader

Interface Timing

Table 15-6 provides the interface timing for the DNA_PORT design primitive. The timing is the same regardless of the FPGA's speed grade. As always, please refer to the associated data sheet for official timing values.

Table 15-6: DNA_PORT Interface Timing

Symbol	Description	Min	Max	Unit
t _{DNASSU}	Setup time on SHIFT before the rising edge of CLK	1.0	—	ns
t _{DNASH}	Hold time on SHIFT after the rising edge of CLK	0.5	—	ns
t _{DNADSU}	Setup time on DIN before the rising edge of CLK	1.0	—	ns
t _{DNADH}	Hold time on DIN after the rising edge of CLK	0.5	—	ns
t _{DNARSU}	Setup time on READ before the rising edge of CLK	5.0	10,000	ns
t _{DNARH}	Hold time on READ after the rising edge of CLK	0	—	ns

Table 15-6: DNA_PORT Interface Timing (Cont'd)

Symbol	Description	Min	Max	Unit
$t_{DNADCKO}$	Clock-to-output delay on DOUT after rising edge of CLK	0.5	1.5	ns
$t_{DNACLKF}$	CLK frequency	0	100	MHz
$t_{DNACLKL}$	CLK High time	1.0	∞	ns
$t_{DNACLKH}$	CLK Low time	1.0	∞	ns

Identifier Memory Specifications

Figure 15-4 presents the general characteristics of the DNA identifier memory. The unique FPGA identifier value is retained for a minimum of ten years of continuous usage under worst-case recommended operating conditions. The identifier can be read, using the READ operation defined in Table 15-5, a minimum of 30 million cycles, which roughly correlates to one read operation every 11 seconds for the operating lifetime of the Spartan-3A/3AN/3A DSP FPGA.

Table 15-7: Identifier Memory Characteristics

Symbol	Description	Minimum	Units
DNA_CYCLES	Number of READ operations, as defined in Figure 15-3 or JTAG ISC_DNA read operations. Unaffected by HOLD or SHIFT operations.	30,000,000	Read cycles

Extending Identifier Length

As shown in Figure 15-5a, most applications that use the DNA_PORT primitive tie the DIN data input to a static value. However, other options are possible.

As shown in Figure 15-5b, the length of the identifier can be extended by feeding the DOUT serial output port back into the DIN serial input port. This way, the identifier can be extended to any possible length. However, there are still only 55 unique bits, with a 57-bit repeating pattern.

It is also possible to add additional bits to the identifier using FPGA logic resources. As shown in Figure 15-5c, the FPGA application can insert additional bits via the DNA_PORT DIN serial input. The additional bits provided by the logic resources could take the form of an additional fixed value or a variable computed from the Device DNA.

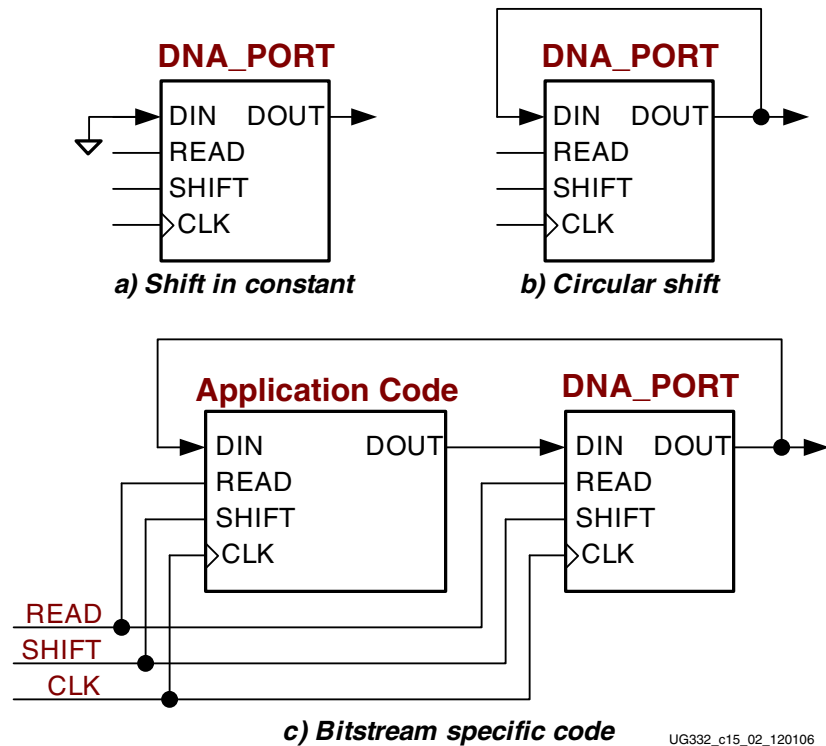


Figure 15-5: Possible Options for DIN Input

JTAG Access to Device Identifier

The FPGA's internal device identifier, plus any values shifted in on the DIN input, can be read via the JTAG port using the private ISC_DNA command. This is a 1532 command and therefore it requires the ISC_ENABLE to be loaded before the ISC_DNA command is issued. JTAG access to the Device DNA value is only available before configuration of the FPGA.

Bit 56 of the identifier, shown in Figure 15-4, appears on the TDO JTAG output following the ISC_DNA command when the device enters the Shift-DR state. The remaining Device DNA bits and any data on the input to the register are shifted out sequentially while the JTAG controller is left in the Shift-DR state.

iMPACT Access to Device Identifier

The iMPACT software in the ISE 10.1 and later tools can also read the Device DNA value. `"readDna -p <position>"` is the batch command that will read the Device DNA from the unconfigured FPGA.

Authentication Design Examples

Authentication can take various forms in an application, as described in the examples below. Some of these examples configure from an attached PROM, others are downloaded into the FPGA.

- ["Extended Spartan-3A Family FPGA: Imprinting or Watermarking the Configuration PROM with Device DNA,"](#) page 298

- “Spartan-3E FPGA: Leveraging Security Features in Select Commodity Flash PROMs,” page 299
- “Extended Spartan-3A Family FPGA: Authenticating a Downloaded Design,” page 302
- “Authenticating any FPGA Design Using External Secure PROM,” page 303

Extended Spartan-3A Family FPGA: Imprinting or Watermarking the Configuration PROM with Device DNA

The Extended Spartan-3A family FPGA in Figure 15-6 configures using one of the Master configuration modes from an associated configuration PROM. The PROM contains both the FPGA configuration bitstream and a previously generated authentication check value. The PROM itself does not require any special features, just enough memory to contain both the FPGA bitstream and the authentication check value. The Extended Spartan-3A family FPGA has an internal unique Device DNA value.

At power-up or when PROG_B is pulsed Low, the FPGA configures normally.

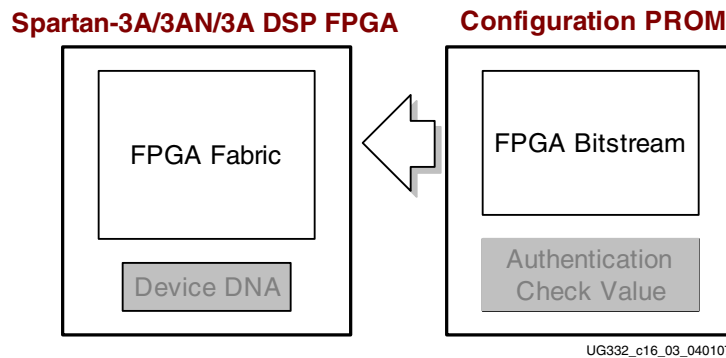


Figure 15-6: Extended Spartan-3A Family FPGA Configures Normally

As shown in Figure 15-7, part of the FPGA application includes circuitry that validates that the bitstream programmed into the PROM is authorized to operate on the associated Extended Spartan-3A family FPGA. In reality, the Device DNA and the authentication check value are both multi-bit binary values. However, for the sake of clarity, this example uses symbolic values. In this example, the FPGA’s Device DNA is “Blue” and the configuration PROM is programmed with the check value “Blueberry.”

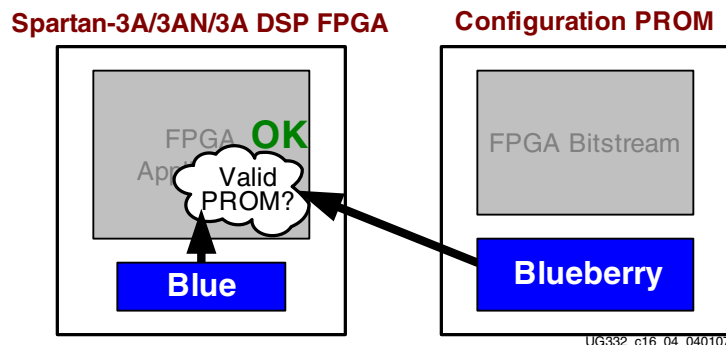


Figure 15-7: Extended Spartan-3A Family FPGA Authenticates the PROM Image Against Device DNA

After configuration, the FPGA checks that the value contained in the PROM matches the value expected by the FPGA application. In this example, the FPGA validates that a “Blueberry” is indeed “Blue.” The bitstream loaded from the PROM is authentic, and the FPGA application is enabled for full operation.

What happens if an attacker copies the contents of an authenticated PROM, shown in [Figure 15-7](#), and uses it with a different, similarly sized Extended Spartan-3A family FPGA? If the check value in the PROM does not match the value expected by the FPGA application, then the FPGA application decides how to handle this unauthorized copy. There are a variety of potential scenarios, as described in “[Handling Failed Authentications](#),” page 304. In this example, the PROM image fails because the FPGA application checks that a “Blueberry” is not “Yellow.”

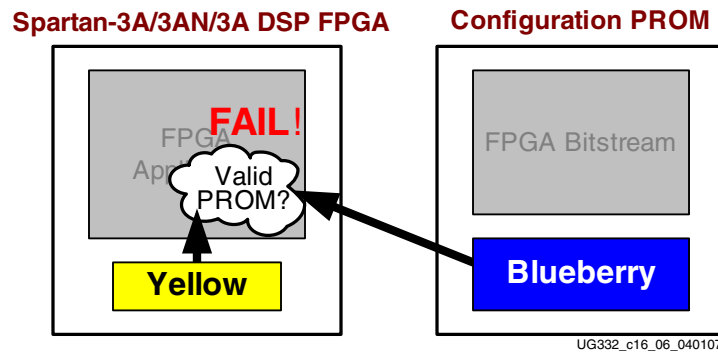


Figure 15-8: Authentication Fails Using an Unauthorized Copy

Spartan-3E FPGA: Leveraging Security Features in Select Commodity Flash PROMs

Only Extended Spartan-3A family FPGAs support the internal unique identifier feature. The feature is not available on Spartan-3 or Spartan-3E FPGAs. However, Spartan-3E and Spartan-3 FPGAs support a similar authentication method using commodity Flash PROMs that have their own device ID values. [Table 15-8](#) provides example devices; there are likely others. The identifiers are only available in certain Flash PROM families and usually in the larger-density members of the family. The size of the identifier also varies by vendor and product family, from 64 bits to 256 bytes. Similarly, some devices also have a user-defined field that can be used to extend the size of the unique ID.

Table 15-8: Example Flash PROMs with Embedded Unique Identifiers

Vendor	Family	Data Format	Density	Unique ID Field	User Field
STMicro (Numonyx)	M29W	Parallel	16 Mbit and larger	64 bits	—
Spansion	S29A	Parallel	32 Mbit and larger	256 bytes (ESN)	—
Atmel	AT45DBxxxD	Serial	All	64 bytes	64 bytes
Atmel	AT45BV	Parallel	8 Mbit and larger	64 bits	64 bits

Table 15-8: Example Flash PROMs with Embedded Unique Identifiers

Vendor	Family	Data Format	Density	Unique ID Field	User Field
Intel (Numonyx)	Embedded Flash (J3 v. D)	Parallel	All	64 bits	64 bits
Intel (Numonyx)	S33	Serial	All	64 bits	64 bits + 3,920 bits
Macronix	MX29	Parallel	32 Mbit and larger	128 word or 64K bytes	—

Figure 15-9 shows an authentication example using a Spartan-3E FPGA and a commodity Flash PROM with an embedded device identifier. In this example, the configuration PROM *must* contain a unique identifier. The PROM also contains the FPGA configuration bitstream and the authentication check value, specific to this implementation.

At power-up or when PROG_B is pulsed Low, the FPGA configures normally.

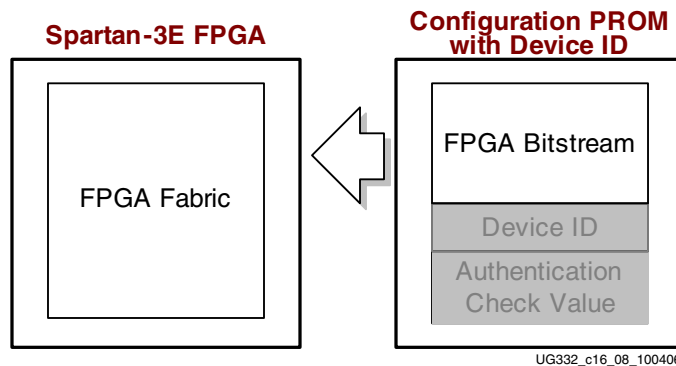


Figure 15-9: Spartan-3E FPGA Authentication Example using Commodity Flash PROM with Identifier

As shown in Figure 15-10, part of the FPGA application includes circuitry that validates that the bitstream programmed into the PROM is authorized to load. The PROM's Device ID and the authentication check value are both multi-bit binary values. For the sake of clarity, this example uses symbolic values. The PROM's Device ID is "Blue" and the configuration PROM is programmed with the check value "Blueberry." The Flash ID plus the authentication check value should be as large as practical. A larger number of bits thwarts a possible "spoof" or "middleman" attack using an extra interposing device or devices that intercepts the access to the off-FPGA identifier and check value. The interposing device or devices mimics the response from an authentic PROM. This technique requires additional components and a new printed circuit board, the additional development and component costs of which act as a suitable deterrent.

If the FPGA authentication application accesses a large data field or check value, then the interposing device or devices must be more sophisticated and consequently more expensive. This potential vulnerability also highlights the advantage of the Extended Spartan-3A family Device DNA, which is securely accessed from inside the FPGA.

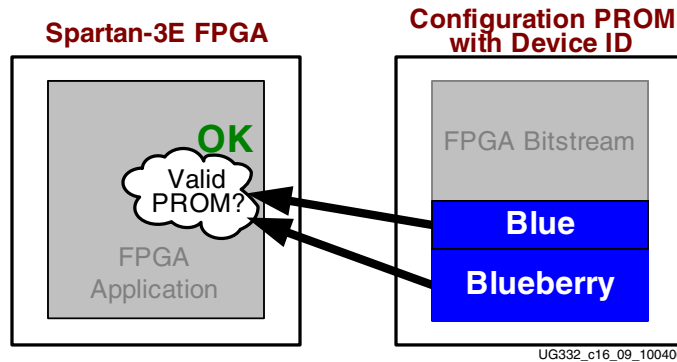


Figure 15-10: **Spartan-3E FPGA Authenticates the PROM Image Against the PROM's Device ID**

After configuration, the FPGA checks that the value contained in the PROM matches the value expected by the FPGA application. In this example, the FPGA validates that a "Blueberry" is indeed "Blue." The bitstream loaded from the PROM is authentic, and the FPGA application is enabled for full operation.

The Spartan-3E Starter Kit board includes a design example demonstrating this technique. This same method also applies for Spartan-3A, Spartan-3AN, Spartan-3A DSP FPGAs.

- **Low-Cost Design Authentication for Spartan-3E FPGAs**

www.xilinx.com/products/boards/s3estarter/reference_designs.htm#authentication

Extended Spartan-3A Family FPGA: Authenticating a Downloaded Design

Authentication also works when downloading an FPGA design. In [Figure 15-11](#), an intelligent host such as a microprocessor, microcontroller, or JTAG tester downloads a bitstream into an Extended Spartan-3A family FPGA. The bitstream is stored somewhere in the system, either in local memory, a disk drive, or obtained via a network connection. The downloaded FPGA application is not yet fully authenticated, but is partially functional to support the authentication process.

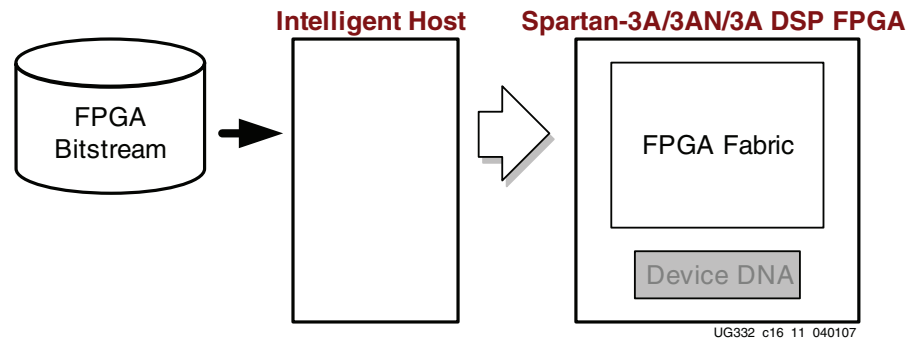


Figure 15-11: Intelligent Host Downloads a Spartan-3A Bitstream

In [Figure 15-12](#), the intelligent host reads the FPGA's Device DNA identifier, either through the FPGA fabric or via the FPGA's JTAG port. Using the Device DNA value, the host either computes an authentication check value locally or communicates to a remote host that generates the check value or looks up the value in a list of authenticated devices.

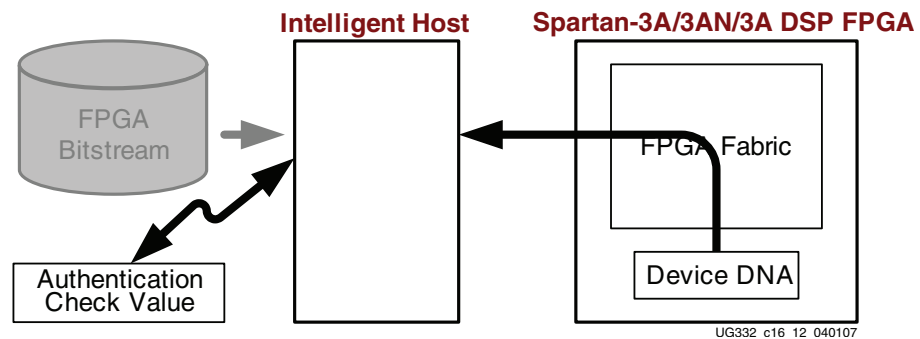


Figure 15-12: Host Reads Device DNA, Generates Authentication Value

In [Figure 15-13](#), the intelligent host writes the resulting authentication check value back into the FPGA. The FPGA then uses this value and the Device DNA value to authenticate the bitstream. If deemed authentic, the FPGA application is enabled for full operation.

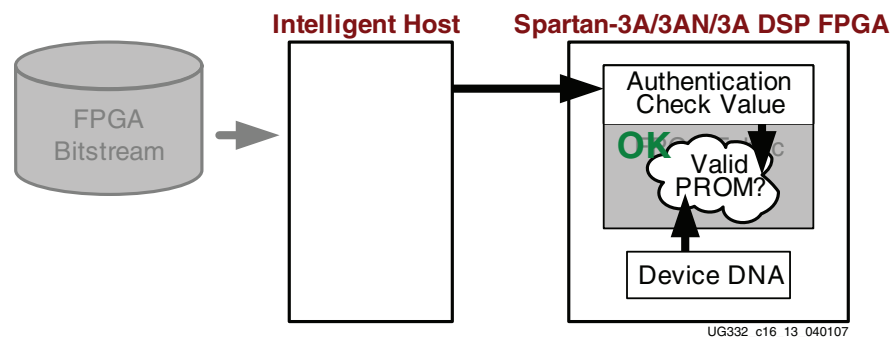


Figure 15-13: Host Writes Authentication Value to Enable FPGA Application

Authenticating any FPGA Design Using External Secure PROM

Authentication techniques are possible on any FPGA using an external secure PROM. Xilinx provides an example design using a Dallas Semiconductor/Maxim DS2432 SHA-1 Secure EEPROM, as shown in Figure 15-14. This technique works with any Xilinx FPGA family with block RAM.

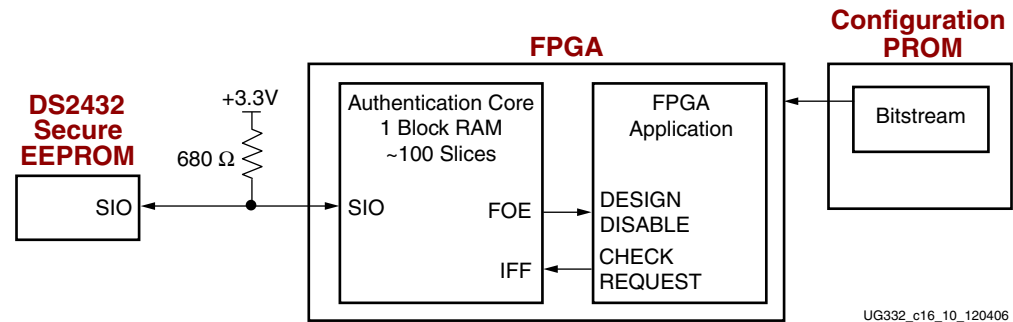


Figure 15-14: FPGA Authentication Using SHA-1 Secure EEPROM

The FPGA configures normally from any configuration PROM. Alternatively, the FPGA bitstream can be downloaded using one of the Slave configuration modes.

The FPGA application contains an Authentication Core that communicates to an external DS2432 secure EEPROM. The authentication challenge between the FPGA and the EEPROM uses a random number and SHA-1 hashing to thwart attacks. If the authentication challenge fails, the FPGA application is disabled. Similarly, the FPGA application can re-authenticate the design at any time, during normal operation.

This application is discussed in more detail in Xilinx application note XAPP780. The Spartan-3E Starter Kit board includes all the necessary components.

- **XAPP780: FPGA IFF Copy Protection Using Dallas Semiconductor/Maxim DS2432 Secure EEPROMs**
http://www.xilinx.com/support/documentation/application_notes/xapp780.pdf
- **DS2432 1Kb Protected 1-Wire EEPROM with SHA-1 Engine**
http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2914
- **Spartan-3E Starter Kit**
<http://www.xilinx.com/s3estarter>

Although not supported by an application note or example design, similar solutions are possible using external components with similar security features, such as the following.

- **STMicroelectronics Krypto™ Secure Parallel Flash Memories**
http://www.st.com/stonline/products/families/memories/fl_nor_emb/fl_m28w_fs.htm
- **Atmel Crypto and Secure Memories**
<http://www.atmel.com/products/SecureMem/>

Handling Failed Authentications

One of the strengths of the Spartan-3 authentication scheme is that the designer and the application decides how best to respond to a failed authentication. A spectrum of solutions is possible, including the following.

- No functionality
- Limited functionality
- Full functionality for a limited period of time
- Active defense against tampering

No Functionality

The simplest way to respond to an unauthorized copy is for the application to stop functioning. This is easily accomplished using features already on the FPGA, such as the following.

- Assert the Global Set /Reset (GSR) signal on the STARTUP design primitive, which holds all flip-flops reset. See “[Start-Up \(STARTUP\)](#),” page 255. The signal driving GSR must be either a logic-based latch or from an SRL16 shift register, neither of which are affected by the GSR signal.
- Assert the global three-state control on the STARTUP design primitive, which forces all output pins to high-impedance (Hi-Z).
- Disable global clock signals using a BUFGCE global clock primitive that has an enable input, which prevents the clock signal from being distributed within the design.
- Assert the reset input to a Digital Clock Manager (DCM).
- Drive the set or reset inputs to key logic in Configurable Logic Blocks (CLBs).
- Use a gating signal to disable key logic in Configurable Logic Blocks (CLBs).
- Selectively disable CLB flip-flops using the clock enable input.
- Any or all of the above.

The disadvantage of this approach is that it immediately tells an attacker whether an attempted breach was successful or not.

Limited Functionality

Limited functionality provides partial or basic functionality. This approach allows a 3rd party test house or contract manufacturers (CM) to build and test the unauthenticated systems. This technique allows the CM to program the configuration PROM but does not provide them authentication capability, eliminating the risk of potential overbuilding.

Disable key functions or special IP using one or more of the techniques described in “[No Functionality](#)”. Optionally, degrade the performance of key features. For example, drop to a lower communications data rate or a lower display resolution.

Full Functionality with Time Out

This technique allows an unauthenticated design to fully operate for a limited amount of time. This approach is most useful when a 3rd party test house or contract manufacturer requires full functionality to complete system testing. However, this technique does not provide the contract manufacturer with the ability to create authentic copies, which reduces the risk of potential overbuilding.

In addition, the time out function makes a potential attack significantly more difficult. If the system functions for awhile before failing, significantly more time is required to attack the system using a brute force approach. Similarly, using a random time out value makes it difficult for an attacker to determine if he or she cracked the system or whether there is an inherent system design problem.

Active Defense

The final level of protection against unauthorized copying is an active defense. The active defense can take many forms, again depending on the application requirements. For example, the application can track the number of failed authentication attempts. Once the number of failed attempts reaches a predefined threshold, the application can take more drastic protection means such as erasing the configuration PROM or permanently locking down sectors in the PROM.

Authentication Algorithm

The obvious question is “What is the authentication algorithm?” The answer is “It’s a secret.” Something in the authentication process must be secret, either the authentication algorithm or the authentication values. In the examples using the Extended Spartan-3A family Device DNA or the Flash PROM with a Device ID, the authentication algorithm must be kept secret.

Because the authentication algorithm is implemented using FPGA logic, the algorithm is flexible and changeable. The algorithm need only be as simple or complex as required by the application being protected. The algorithm can be changed between design releases or versions. Similarly, multiple and different authentication checks can co-exist in the same application. This approach tunes the “cost” and complexity of security to the needs of the application.

Manufacturing Logistics

Authentication simplifies manufacturing logistics, especially for high-volume applications, built at contract manufacturers.

- There are no special keys that need to be programmed into the FPGA. There is a programming step where the PROM is “married” to or authenticated with either the FPGA’s Device DNA, the PROM’s Unique ID, or both but this operation does not affect the FPGA bitstream.
- The FPGA bitstream is common to all units. There is no need to match a bitstream to a specific FPGA or a set of FPGAs. The authentication step can be completely separate from bitstream programming.
- Configuration PROMs can be bulk programmed. There is no need to match a PROM to a specific FPGA or a set of FPGAs during high-volume manufacturing. The authentication step can occur at any time, such as in final system test, in a secure facility or at the end customer site.
- Using the “[Limited Functionality](#),” [page 304](#) or “[Full Functionality with Time Out](#),” [page 304](#) techniques described early, the contract manufacturer can build and test the end product without risk of overbuilding or unauthorized cloning.

Additional Uses of Authentication and Device ID

The authentication techniques described in this chapter primarily protect the FPGA application. However, these techniques serve other potential purposes in an application.

Protecting Intellectual Property (IP)

FPGAs are a common deployment vehicle for intellectual property (IP) cores. Authentication solves a key concern for IP vendors allowing them to protect the IP core from unauthorized copying. The techniques described above also allow a vendor to protect key IP but still allow potential customers to try the core before purchase.

Furthermore, the Device DNA feature in Extended Spartan-3A family FPGAs provides full traceability, allowing an IP vendor to track unit shipments by a customer in order to determine royalty-based payments for an IP core.

Code and Data Security

The Extended Spartan-3A family FPGA's Device DNA identifier provides an additional level of security for embedded applications. The Device DNA forms a key used to encrypt and decrypt both code and data to protect an embedded processor application.

Figure 15-15 shows an example MicroBlaze™ processor application. The Extended Spartan-3A family Device DNA identifier forms a key to encrypt and decrypt both code and data.

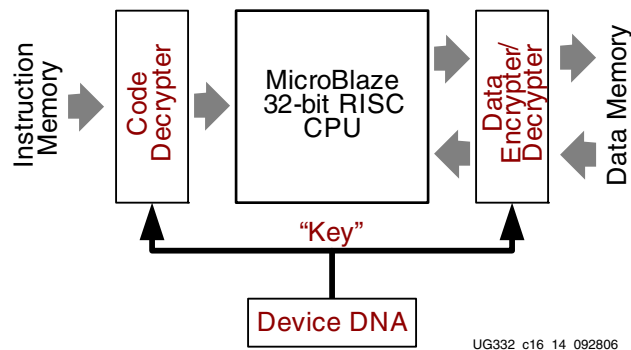


Figure 15-15: Extended Spartan-3A Family Device DNA Used as a Key to Protect Embedded Processing Applications

U.S. Legal Protection of FPGA Configuration Bitstream Programs

The FPGA configuration bitstream program may qualify as a “computer program” as defined in Section 101, Title 17 of the United States Code, and as such may be protected under the copyright law. It may also be protected as a trade secret if it is identified as such.

Xilinx suggests that a company wishing to claim copyright and/or trade secret protection in the FPGA configuration bitstream consider taking the following steps.

Place an appropriate copyright notice on the FPGA or adjacent to it on the printed circuit board (PCB) giving notice to third parties of the copyright. For example, because of space limitations, this notice on the FPGA device could read “©2006 XYZ Company” or, if on the PCB, could read “Bitstream © 2006 XYZ Company”.

File an application to register the copyright claim for the bitstream program with the U.S. Copyright Office.

If practical, given the size of the printed circuit board, notice should also be given that the user is claiming that the bit-stream program is the company's trade secret. A statement could be added to the PCB such as: "Bitstream proprietary to XYZ Company. Copying or other use of the bitstream program except as expressly authorized by XYZ Company is prohibited."

To the extent that documentation, data books, or other literature accompanies the FPGA-based design, appropriate wording should be added to this literature providing third parties with notice of the user's claim of copyright and trade secret in the bitstream program.

For example, this notice could read: "Bitstream ©2006 XYZ Company. All rights reserved. The bitstream program is proprietary to XYZ Company and copying or other use of the bit-stream program except as expressly authorized by XYZ Company is expressly prohibited."

To help prove unauthorized copying by a third party, additional nonfunctional code should be included at the end of the bitstream program. Therefore, should a third party copy the bitstream program without proper authorization, if the non-functional code is present in the copy, the copier cannot claim that the bitstream program was independently developed.

These are only suggestions, and Xilinx makes no representations or warranties with respect to the legal effect or consequences of the above suggestions. Each end-user company is advised to consult legal counsel with respect to seeking protection of a bitstream program and to determine the applicability of these suggestions to the specific circumstances.

Additional Information

For additional information see the following references on xilinx.com.

- **Xilinx Design Security Solutions**
http://www.xilinx.com/products/design_resources/security/index.htm
- **Xilinx Spartan-3 Generation Device DNA Security**
http://www.xilinx.com/products/design_resources/security/devicedna.htm
- **WP266: Security Solutions Using Spartan-3 Generation FPGAs**
http://www.xilinx.com/support/documentation/white_papers/wp266.pdf
- **WP267: Advanced Security Schemes for Extended Spartan-3A Family FPGAs**
http://www.xilinx.com/support/documentation/white_papers/wp267.pdf

Configuration CRC

All Spartan[®]-3 generation FPGAs have an embedded cyclic-redundancy checker (CRC) circuit designed to flag errors when loading the configuration bitstream. The configuration CRC circuit is always active during configuration unless specifically disabled in the configuration bitstream. Extended Spartan-3A family FPGAs also optionally allow the CRC circuit to continue operating after configuration.

CRC Checking during Configuration

As the configuration data frames are loaded, the FPGA calculates a Cyclic Redundancy Check (CRC) value from the configuration data packets. After all the configuration data frames are loaded, the configuration bitstream issues a Check CRC command to the FPGA, followed by an expected CRC value. The CRC uses a 32-bit value in the Spartan-3 and Spartan-3E families and a 22-bit value in the Extended Spartan-3A family. If the CRC check values match, the FPGA continues the configuration process by progressing to the Startup phase. If the CRC value does not match, then there are slightly different behaviors between the various Spartan-3 generation product families, as described below.

Spartan-3 and Spartan-3E Configuration CRC Errors

If the CRC value calculated by the FPGA does not match the expected CRC value in the bitstream, the FPGA drives the INIT_B pin Low and aborts configuration. When a CRC error occurs, the CCLK output goes to the high-impedance state (Hi-Z), unless the HSWAP or HSWAP_EN pin is Low, in which case the CCLK output is pulled High.

Configuration CRC Enabled by Default

The CRC check is included in the configuration bitstream by default (**CRC:Enable**). However it is possible to disable the check, which should only be done in rare circumstances and with great caution. If the CRC check is disabled, there is a risk of loading incorrect configuration data frames, causing incorrect design behavior or damage to the FPGA. The fallback capability is also disabled when CRC is disabled.

Possible CRC Escapes

There is a scenario where errors in transmitting the configuration bitstream can be missed by the CRC check. Certain clocking errors, such as double-clocking, can cause loss of synchronization between the bitstream packets and the configuration logic. Once synchronization is lost, any subsequent commands are not understood by the FPGA, including the command that performs the CRC check. In this situation, configuration fails with the FPGA's DONE pin Low and the INIT_B pin High because the CRC was ignored. In Extended Spartan-3A family BPI mode, the address counter eventually overflows or

underflows to cause wraparound, which triggers reconfiguration if the `Reset_on_err:Yes` bitstream option is set.

Extended Spartan-3A Family Configuration CRC Errors and Configuration Watchdog Timer

Extended Spartan-3A family FPGAs include a Configuration Watchdog Timer (CWDT) function. If the FPGA configures in one of the Master modes, and if the `Reset_on_err:Yes` bitstream option is set, then the Extended Spartan-3A family FPGA automatically re-initializes itself and attempts to reconfigure if a CRC error occurs during configuration. In BPI and SPI modes, if reconfiguration fails three times, then the FPGA halts and drives the INIT_B pin Low. The CCLK output goes to the high-impedance state (Hi-Z), unless the HSWAP or HSWAP_EN pin is Low, in which case the CCLK output is pulled High. Pulsing the PROG_B pin or power cycling restarts the configuration process from the beginning. The JTAG interface remains responsive and the device is still alive, only the BPI/SPI interface is inoperable.

The counter that keeps track of the three failed configurations is reset only when PROG_B is pulsed or power is cycled; it is not reset after a successful configuration. Note that when configuring via SPI or BPI modes and using the `Reset_on_err:Yes` bitstream option, any combination of successful and failed configurations, over any period of time, will halt after the third failed configuration, and require assertion of PROG_B or power cycling to reconfigure. It is good design practice to have the ability to assert PROG_B to reset configuration if necessary.

Robust CMOS Configuration Latches (CCLs)

FPGA configuration data is stored in robust CMOS configuration latches (CCLs). Despite being readable and writable like static RAM (SRAM), CCLs are designed primarily for stability, resulting in improved stability over voltage and temperature. CCLs also exhibit 10 to 100 times better immunity to single-event upset (SEU) phenomenon than traditional SRAM memories.

Xilinx is a world-leader in measuring and mitigating SEU effects on FPGAs. Extensive proton-beam and atmospheric data is available upon request.

Post-Configuration CRC (Extended Spartan-3A Family Only)

Despite the robust stability of the CMOS configuration latches (CCLs) that hold the FPGA configuration data, some high-reliability, high-demand applications require continuous checking of all configuration memory locations. Extended Spartan-3A family FPGAs offer this capability. The configuration CRC checker can be enabled so that it continues to monitor the FPGA bitstream after configuration.

Overview

[Figure 16-1, page 311](#) provides a conceptual overview of the post-configuration CRC checker circuit.

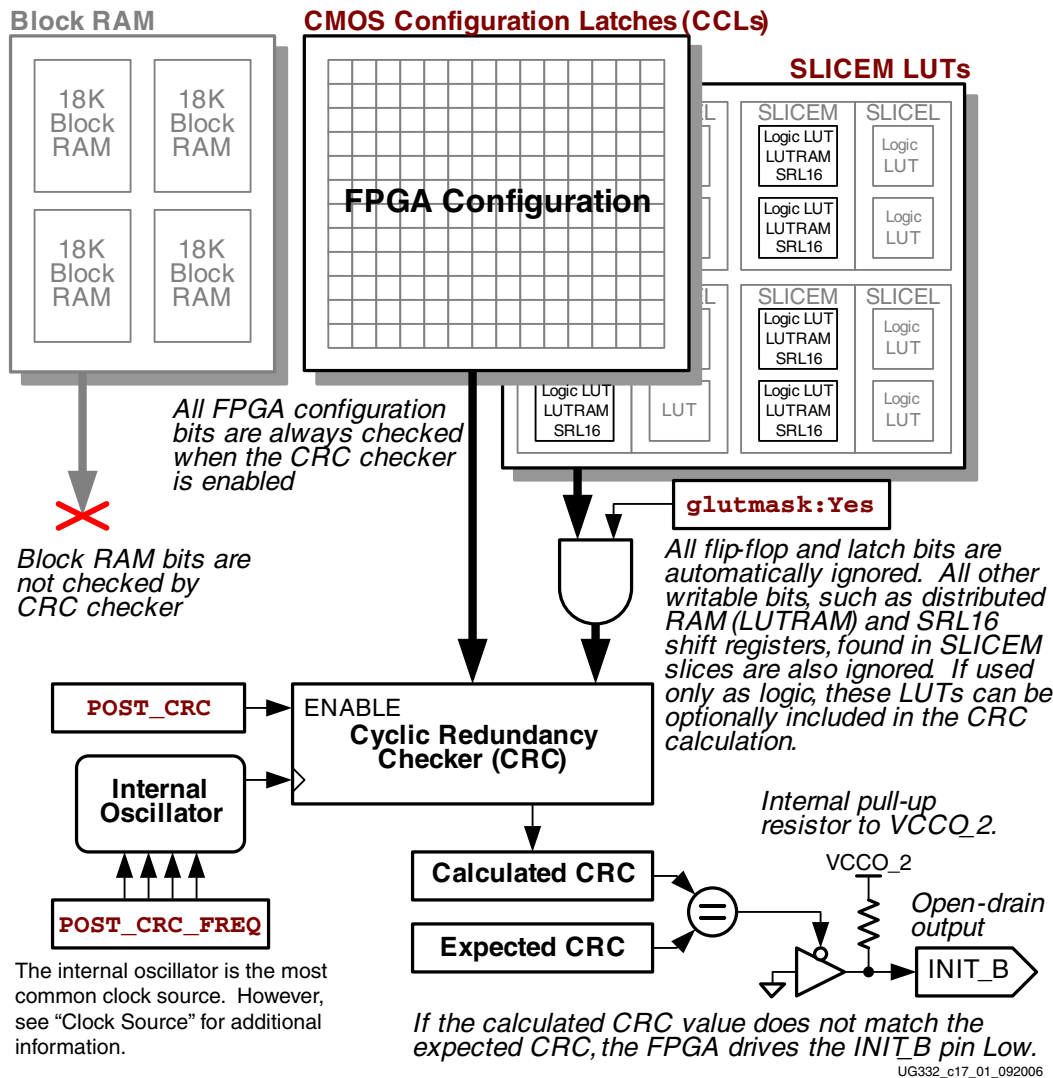


Figure 16-1: Conceptual Overview of Post-Configuration CRC Calculator

If the **POST_CRC=ENABLE** configuration constraint is set, then the CRC checker circuit continuously scans the FPGA bitstream, calculates a resulting CRC value, then compares this value against a previously calculated, expected CRC value. If there is a difference between the two CRC check values, then the CRC checker flags the error by driving the FPGA's **INIT_B** pin Low.

The calculated CRC value changes if any unmasked bit, in any location, changes for any reason. Obviously, the FPGA application will modify some locations during the course of normal operation. Consequently, all writable bits, such as flip-flops, latches, and block RAM are automatically excluded from the CRC calculation. Any write operations to these locations would otherwise result in a different calculated CRC value and a subsequent CRC "error." Similarly, the look-up tables (LUTs) within the SLICEM logic slices in each Configurable Logic Block (CLB) also potentially contain writable functions, such as distributed RAM or SRL16 shift registers.

Consequently, by default, all LUTs in all SLICEM logic slices are excluded from the calculated CRC value. However, if all the LUTs in the FPGA application are only used for

logic, that is, there is no distributed RAM or SRL16 shift registers in the design, then the SLICEM LUTs can be included in the calculation by setting the *glutmask:No* bitstream option. See the ISE® software project summary report to determine if the design uses any distributed RAM or SRL16 shift registers.

Block RAM is excluded from the CRC calculation because a change in the RAM content would have a subsequent change in the CRC result. Byte-, half-word-, or word-level changes are easily detected using the available parity bits provided as part of the block RAM function. See “[Techniques to Check Distributed and Block RAM Contents](#),” page 315 for more information.

Continuous CRC Checking Until Configuration, JTAG or Suspend Event

The CRC checker continues until one of the following events occurs. Upon any one of these events, the CRC checker stops operating.

- The configuration controller receives a valid synchronization word, which can occur if the FPGA is being reconfigured or from a MultiBoot operation.
- There is an active configuration operation via the JTAG port, or the Mode pins are set to JTAG mode.
- The FPGA enters the power-saving Suspend mode.

If enabled in the bitstream, the CRC checker will reset and restart at the end of the configuration event or when the FPGA awakens from Suspend mode.

Clock Source

If enabled, the post-configuration CRC checker is clocked by one of three sources, depending on the specific FPGA application, ordered from least-likely to most-likely.

- If the bitstream option *Persist:Yes* is selected and the FPGA is configured using one of the Slave configuration modes, then the post-configuration CRC checker is clocked using the FPGA's CCLK input pin.
- If the Internal Configuration Access Port (ICAP) feature is enabled, the post-configuration CRC checker is clocked by the CLK input on the ICAP design primitive.
- Otherwise, the post-configuration CRC checker is clocked by the FPGA's internal oscillator. Set the frequency of the internal oscillator using the `POST_CRC_FREQ` configuration constraint. See [Table 16-1](#) for available options.

CRC Checking Time

The time required for each CRC calculation is similar to the serial configuration time, and depends on the density and clock rate. The CRC engine is a one-bit-wide shift register as are the internal registers of the device. So, for each clock period one bit will be shifted into the CRC engine. The total time then to run one CRC check will be the (total number of configuration bits) X (clock period). For example, the XC3S50A has 437,312 bits; running at 12 MHz, the CRC check will take 0.0364 seconds. The XC3S1400A has 4,755,296 bits; running at 12 MHz, the CRC check will take 0.39627 seconds.

Behavior when CRC Error Occurs

As described earlier, the FPGA flags a post-configuration CRC error by driving the open-drain INIT_B pin Low. The INIT_B pin will stay Low until the device is re-configured. This is identical to the way that the FPGA flags a CRC error during configuration. When the

post-configuration CRC feature is enabled, the INIT_B pin is reserved as an open-drain output with an internal, dedicated pull-up resistor to the VCCO_2 supply input. The INIT_B pin cannot be used as a user-I/O when the CRC feature is enabled.

The `POST_CRC_ACTION` configuration constraint defines how the post-configuration CRC checker behaves should it detect an error. If `POST_CRC_ACTION=HALT`, then the CRC circuit stops calculating a new CRC value if an error occurs. This allows an external device to check the CRC signature using Readback. If `POST_CRC_ACTION=CONTINUE`, then the CRC circuit continues to check for additional post-configuration CRC errors, even after detecting an error. The INIT_B pin stays Low after the first error, while additional CRC changes would indicate additional errors.

The FPGA-based system separately determines what action to take when a CRC error occurs. Most applications will simply decide to reconfigure the FPGA.

Verifying CRC Error Behavior

To verify the post-configuration CRC checking function, the user can force a change using the SRL16 logic. Instantiate at least one SRL16 and set the `glutmask:No` bitstream option. Write to the SRL16 to change its state and the post-configuration CRC feature should flag the CRC error.

Preparing an Application to Use the Post-Configuration CRC Feature

- Enable the post-configuration CRC logic using the `POST_CRC=ENABLE` configuration constraint.
- The post-configuration CRC checker is clocked by one of three possible clock sources as described in “Clock Source”. Be sure that the application or system is providing the required clock input. In most applications, the CRC checker uses the FPGA’s internal oscillator as the clock source. Set the oscillator frequency using the `POST_CRC_FREQ` configuration constraint. By default, the oscillator is set at 1, which roughly equates to a 1 MHz clock. See Table 16-2 for available options.
- Using the `POST_CRC_ACTION` configuration constraint, define whether the CRC checker will continue to check for additional CRC errors or will halt checking.
- If any look-up tables (LUTs) in the FPGA application are used as distributed RAM or SRL16 shift registers, then leave the `glutmask` bitstream generator option at its default value.

Example User Constraints File (UCF)

Figure 16-2 provides an example user constraints file to enable the post-configuration CRC checker.

```
# Enable the post-configuration CRC checker
CONFIG POST_CRC = ENABLE ;

# Set clock frequency for CRC checker circuitry
CONFIG POST_CRC_FREQ = 1 ;

# Define if the CRC checker continues or halts after detecting an error
CONFIG POST_CRC_ACTION = CONTINUE ;
```

Figure 16-2: UCF Constraints for Post-Configuration CRC

CONFIG Constraints

Table 16-1 lists the available CONFIG constraints that control the post-configuration CRC feature.

Table 16-1: Post-Configuration CRC CONFIG Constraints

CONFIG Constraint	Setting	Description
POST_CRC	DISABLE	Default. Disable the post-configuration CRC checker. INIT_B pin is available as a user-I/O pin.
	ENABLE	Enable the post-configuration CRC checker. INIT_B pin is reserved to flag CRC errors and not available as a user-I/O pin.
POST_CRC_FREQ	1, 3, 6, 7, 8, 10, 12, 13, 17, 22, 25, 27, 33, 44, 50, 100	Default value is 1. Sets the clock frequency used for the post-configuration CRC checker.
POST_CRC_ACTION	CONTINUE	Default. If a CRC mismatch is detected, continue reading back the bitstream, computing the comparison CRC, and making the comparison against the precomputed CRC.
	HALT	If a CRC mismatch is detected, cease CRC check.

Bitstream Generator Options

Table 16-2 lists the bitstream generator (BitGen) options associated with the post-configuration CRC feature. The shaded fields are hidden because the CONFIG constraints are the preferred control mechanism, as described in “Preparing an Application to Use the Post-Configuration CRC Feature,” page 313. The *glutmask* option has no associated CONFIG constraint.

Table 16-2: Post-Configuration CRC Bitstream Generator Options

BitGen Option	Setting (default)	Description
post_crc_en	<i>No</i>	Default. Disable the post-configuration CRC checker.
	Yes	Enable the post-configuration CRC checker.
post_crc_freq	1, 3, 6, 7, 8, 10, 12, 13, 17, 22, 25, 27, 33, 44, 50, 100	Sets the clock frequency used for the post-configuration CRC checker. The available options are the same as for the <i>ConfigRate</i> bitstream option.
post_crc_keep	<i>No</i>	Default. Stop checking when error detected. Allows CRC signature to be read back.
	Yes	Continue to check for CRC errors after an error was detected.

Table 16-2: Post-Configuration CRC Bitstream Generator Options (Cont'd)

BitGen Option	Setting (default)	Description
glutmask	<u>Yes</u>	Default. Mask out the Look-Up Table (LUT) bits from the SLICEM logic slices. SLICEMs support writable functions such as distributed RAM and SRL16 shift registers, which generate CRC errors when bit locations are modified.
	No	Include the Look-Up Table (LUT) bits from SLICEM logic slices. Use this option only if the application does not include any distributed RAM or SRL16 shift registers.

Design Considerations

While all flip-flop and latch values are automatically ignored, the initial values for each flip-flop and latch are included in the CRC calculation. Consequently, do not issue a Readback CAPTURE operation when the post-configuration CRC feature is enabled. The CAPTURE operation captures the current flip-flop and latch values and writes them back to the memory cells that originally contained the initial values.

Techniques to Check Distributed and Block RAM Contents

As described earlier, block RAM, LUT RAM, and SRL16 shift registers are not included as part of the CRC calculation. Any RAM errors, should they occur, are not flagged on the INIT_B pin. However, it is possible to check RAM contents during operation using simple parity, as shown in Figure 16-3.

Each block RAM has additional bit locations specifically to store parity values. The parity generator is a simple XOR gate, implemented using FPGA logic. Parity is generated for any data written to the block RAM. The parity checker is also a simple XOR gate, effectively with an additional input. Parity is generated for any data value read from block RAM. The generated parity value is compared against the parity bit also read from the RAM. If the values are different, then an odd number of bits changed within the RAM location between the time the value was written to the time it was read and checked.

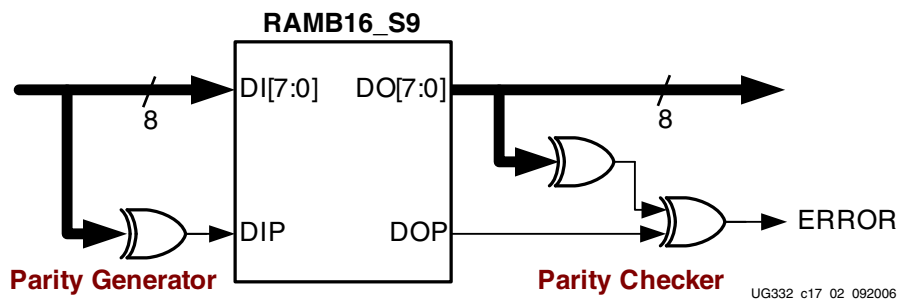


Figure 16-3: Checking Block RAM Contents Using Simple Parity

Although Figure 16-3 shows a block RAM example, the same technique applies for distributed RAM.

Similarly, both block RAM and distributed RAM support dual-port read operations. The parity checker function can be moved to the second read port so that it can continuously

monitor the RAM contents without affecting normal operation. Similarly, if the block RAM contents are static, if used to store PicoBlaze™ processor code as an example, then FPGA logic can use the second block RAM port and continuously calculate a CRC signature for the block RAM contents. If the signature changes between subsequent checking operations, then the circuit flags an error. This is similar to the method used to continuously check the FPGA configuration memory cells.

Configuration Details

This chapter provides more extensive details on the configuration logic in the Extended Spartan®-3A family, including the Spartan-3A, Spartan-3AN, and Spartan-3A DSP platforms. The Spartan-3E family is similar, and details on the Spartan-3 family can be found in [XAPP452](#).

All user-programmable features inside Extended Spartan-3A devices are controlled by volatile memory cells that must be configured at power-up. These memory cells are collectively known as *configuration memory*. They 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. The bitstream is automatically generated by the Xilinx ISE® design tools and is delivered to the device through one of the configuration interfaces.

The composition of the bitstream is largely independent of the configuration method. A bitstream for the Slave Parallel (SelectMAP) interface can look exactly the same as a bitstream for the Master Serial interface. Certain configuration operations, however, such as readback, can only be performed through the SelectMAP and JTAG interfaces.

The user generally does not need to know the details of the configuration format and commands. However, this detail can be useful for debugging purposes. After initial configuration, the user can send configuration commands to the device through the permanent JTAG interface, through the SelectMAP port if Persist is selected, or through the Internal Configuration Access Port if ICAP_SPARTAN3A is included in the design. The easiest method for sending configuration commands using the JTAG interface is with an SVF file (see [XAPP503 “SVF and XSVF File Formats for Xilinx Devices”](#)).

Configuration Memory Frames

Extended Spartan-3A family configuration memory is arranged in *frames* that are tiled about the device. These frames are the smallest addressable segments of the Extended Spartan-3A configuration memory space, and all operations must therefore act upon whole configuration frames.

The FPGA configuration memory can be visualized as a rectangular array of bits. The bits are grouped into vertical frames that are one-bit wide and extend from the top of the array to the bottom. A frame is the atomic unit of configuration. It is the smallest portion of the configuration memory that can be written to or read from. Frames do not directly map to any single piece of hardware. For instance, a single frame does not configure a single CLB or IOB, but actually configures a part of several logical resources, as well as some routing. Frames are grouped into larger units called columns.

Extended Spartan-3A frame counts and configuration sizes are shown in [Table 17-1](#). Depending on bitstream generator (BitGen) options, additional overhead exists in the configuration bitstream. The exact bitstream length is available in the .rpt (rawbits) file created by using the "-b" option with BitGen or selecting "Create ASCII Configuration File" in the Generate Programming File options popup in the ISE software. Bitstream length (words) are roughly equal to the configuration array size (words) plus configuration overhead (words). Bitstream length (bits) are roughly equal to the bitstream length in words times 16.

Table 17-1: Extended Spartan-3A Frame Count and Length

Device	Device Frames	Frame Length (words) ⁽¹⁾	Configuration Array Size (words) ⁽²⁾	Configuration Overhead (words) ⁽³⁾
XC3S50A/AN	367	74	147,600	174
XC3S200A/AN	540	138	243,048	238
XC3S400A/AN	692	170	381,792	270
XC3S700A/AN	844	202	552,352	302
XC3S1400A/AN	996	298	726,520	398
XC3SD1800A	1414	362	958,416	462
XC3SD3400A	1718	426	1,259,520	526

Notes:

1. All Extended Spartan-3A family configuration frames consist of 16-bit words.
2. *Configuration array size* equals the number of configuration frames times the number of words per frame.
3. *Configuration overhead* consists of commands in the bitstream that are needed to perform configuration, but do not themselves program any memory cells. Configuration overhead contributes to the overall bitstream size.

Configuration Control Logic

The Extended Spartan-3A family 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 (Parallel, JTAG, or Serial) to the appropriate register. The registers control all other aspects of configuration.

Packet Types

The FPGA bitstream consists of two packet types: Type 1 and Type 2. These packet types and their usage are described below.

Type 1 Packet

The Type 1 packet is used for register writes and reads. Only 6 register address bits are used in Extended Spartan-3A FPGAs. The header section is always a 16-bit word.

Following the Type 1 packet header is the Type 1 Data section, which contains the number of 16-bit words specified by the word count portion of the header. See [Table 17-2](#), [Table 17-3](#), and [Table 17-4](#).

Table 17-2: Type 1 Packet Header Format

Header Type	Opcode	Register Address	Word Count
[15:13]	[12:11]	[10:5]	[4:0]
001	xx	xxxxxx	xxxxx

Table 17-3: Type 1 Packet Data Format

Data	[15:0]
Word 1	xxxxxxxxxxxxxxxxxx
...	xxxxxxxxxxxxxxxxxx
Word [Word Count]	xxxxxxxxxxxxxxxxxx

Table 17-4: Opcode Format

Opcode	Function
00	NOOP
01	Read
10	Write
11	Reserved

Type 2 Packet

The Type 2 packet, which must follow a Type 1 packet, is used to write long blocks. The header section is always a 16-bit word. See [Table 17-5](#).

Following the Type 2 packet header is the Type 2 Word count section, which contains two 16-bit words with the MSB in the first word. See [Table 17-6](#) and [Table 17-7](#).

Following the Type 2 Word count section is the Type 2 Data section, which contains the number of 16-bit words specified by the Word count section. See [Table 17-8](#).

Table 17-5: Type 2 Packet Header

Header Type	Opcode	Register Address	Reserved
[15:13]	[12:11]	[10:5]	[4:0]
010	xx	xxxxx	RRRRR

Notes:

1. "R" means the bit is not used and is reserved for future use.

Table 17-6: Type 2 Packet Word Count 1

Word Count 1
[31:16]
0000xxxxxxxxxxxx

Table 17-7: Type 2 Packet Word Count 2

Word Count 2
[15:0]
xxxxxxxxxxxxxxxxxx

Table 17-8: Type 2 Packet Data

Data	[15:0]
Word 1	xxxxxxxxxxxxxxxxxx
...	xxxxxxxxxxxxxxxxxx
Word [Word Count]	xxxxxxxxxxxxxxxxxx

Configuration Registers

All bitstream commands are executed by reading or writing to the configuration registers. [Table 17-9](#) summarizes these registers. A detailed explanation of selected registers follows.

Table 17-9: Configuration Registers

Reg. Name	Read/Write	Address	Description
CRC	Read/Write	00 0000	CRC Register (uses a 22-bit CRC checksum algorithm to verify bitstream integrity during configuration)
FAR_MAJ	Write	00 0001	See Frame Address Registers (FAR_MAJ and FAR_MIN) , Block and Major
FAR_MIN	Write	00 0010	See Frame Address Registers (FAR_MAJ and FAR_MIN) , Minor
FDRI	Write	00 0011	Frame Data Register, Input (write configuration data by loading WCFG command and loading FDRI)
FDRO	Read	00 0100	Frame Data Register, Output register (read configuration data and optionally captured flip-flop data by loading RCFG command and then addressing the FDRO)
CMD	Read/Write	00 0101	See Command Register (CMD)
CTL	Read/Write	00 0110	See Control Register (CTL)
MASK	Read/Write	00 0111	Masking Register for CTL (1 allows bit to be written - default is all 0s)
STAT	Read	00 1000	See Status Register (STAT)
LOUT	Write	00 1001	Legacy Output Register (DOOUT for serial daisy chain)
COR1	Read/Write	00 1010	See Configuration Options Registers (COR1 and COR2)
COR2	Read/Write	00 1011	See Configuration Options Registers (COR1 and COR2)
PWRDN_REG	Read/Write	00 1100	See Suspend Options Register (PWRDN_REG)
FLR	Write	00 1101	Frame Length Register (number of 16-bit words in the length of a frame; 16 bits with first 6 reserved at 0)
IDCODE	Read/Write	00 1110	Device ID Register; includes 16 most significant bits of IDCODE (family and array codes) - see Table 12-4

Table 17-9: Configuration Registers (Cont'd)

Reg. Name	Read/Write	Address	Description
SNOWFLOW	Write	00 1111	Not supported; default is all 0s
HC_OPT_REG	Read/Write	01 0000	See Housecleaning Options Register (HC_OPT_REG)
reserved	-	01 0001	-
CSBO	Write	01 0010	Asserts CSO_B Output for Parallel Daisy Chaining
GENERAL1	Read/Write	01 0011	16 LSBs of MultiBoot Address - see “ MultiBoot Registers ” in Chapter 14
GENERAL2	Read/Write	01 0100	16 MSBs of MultiBoot Address - see “ MultiBoot Registers ” in Chapter 14
MODE_REG	Read/Write	01 0101	MultiBoot Mode - see “ Configuration Mode Register (MODE_REG) ” in Chapter 14
PU_GWE	Write	01 0110	GWE Cycle Exiting Suspend Mode (10 bits)
PU_GTS	Write	01 0111	GTS Cycle Exiting Suspend Mode (10 bits)
MFWR	Write	01 1000	See Multiple Frame Write Register (MFWR)
CCLK_FREQ	Write	01 1001	CCLK Frequency Select for Master Modes (16 bits); 6 MSBs reserved and default to 000001; 10 LSBs default to 0110111110
SEU_OPT	Write	01 1010	Post-Configuration CRC (POST_CRC) Options
EXP_SIGN	Read/Write	01 1011	32-bit Expected CRC Signature for POST_CRC
RDBK_SIGN	Read	01 1100	32-bit Calculated CRC Signature for POST_CRC

Command Register (CMD)

Configuration commands control the operation of the configuration state machine. Each command consists of five bits, and the MSB is always zero. The command code is written as part of a 16-bit value. Each 16-bit command is written as two bytes to the SelectMAP or ICAP interface, with the high byte presented first, followed by the low byte. Note that D0 is the most-significant bit (MSB) for the ICAP interface, which is the opposite direction from most processors.

The Command Register is used to instruct the configuration control logic to strobe global signals and perform other configuration functions. The command present in the CMD register is executed each time the Frame Address Registers are loaded with new values. [Table 17-10](#) gives the Command Register commands and codes.

Table 17-10: Command Register Codes

Command	Code	Description
NULL	00000	Null Command
WCFG	00001	Write Configuration Data: used prior to writing configuration data to the FDRI.
MFWR	00010	Multiple Frame Write: used to perform a write of a single frame of data to multiple frame addresses.
LFRM	00011	Last Frame: Deasserts the GHIGH_B signal, activating all interconnect. The GHIGH_B signal is asserted with the AGHIGH command.
RCFG	00100	Read Configuration Data: used prior to reading configuration data from the FDRO.

Table 17-10: Command Register Codes (Cont'd)

Command	Code	Description
START	00101	Begin Startup Sequence: initiates the startup sequence. The startup sequence begins after a successful CRC check and a DESYNC command are performed.
RCAP	00110	Reset Capture: resets the CAPTURE signal after performing readback-capture in single-shot mode (see “Readback Capture” in Chapter 18).
RCRC	00111	Reset CRC: resets the CRC register
AGHIGH	01000	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 in shutdown reconfiguration. Interconnect is reactivated with the LFRM command.
reserved	01001	-
GRESTORE	01010	Pulse the GRESTORE Signal: sets/resets (depending on user configuration) IOB and CLB flip-flops.
SHUTDOWN	01011	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	01100	Pulse GCAPTURE: loads the readback capture cells with the current register states (see “Readback Capture” in Chapter 18).
DESYNC	01101	Reset DALIGN Signal: used at the end of configuration to desynchronize the device. After de-synchronization, all values on the configuration data pins are ignored.
REBOOT	01110	MultiBoot operation (see “Extended Spartan-3A Family MultiBoot” in Chapter 14).
reserved	01111	-

Control Register (CTL)

The Control Register is used to set the configuration security level, the persist setting, and the Global Three-State signal. Writes to the CTL register are masked by the value in the MASK register (this allows the GTS_USR_B signal to be toggled without re-specifying the Security and Persist bits). The MASK register is cleared after each write to the CTL register, which prevents inadvertent changes to the Control Register.

The fields are defined in Table 17-11.

Table 17-11: Control Register Description

Name	Bit Index	Description
EN_MBOOT	7	Enable MultiBoot mode. 0: MultiBoot Disabled (default) 1: MultiBoot Enabled
Reserved	6	Reserved CTL register bit. Always leave this bit set to 0.
SBITS	5:4	Security Level. 00: Read/Write OK (default) 01: Readback disabled except through ICAP 10: All Readback disabled, writing disabled except CRC register

Table 17-11: Control Register Description (Cont'd)

Name	Bit Index	Description
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
ICAP	2	ICAP Port Enable. 0: ICAP Port Disabled (default) 1: ICAP Port Enabled
Reserved	1	Reserved CTL register bit. Always leave this bit set to 0.
GTS_USER_B	0	Active Low high-Z state for I/Os. 0: I/Os placed in high-Z state 1: I/Os active (default)

Frame Address Registers (FAR_MAJ and FAR_MIN)

The Frame Address Registers set the starting block and column address for the next configuration data input. The typical bitstream starts at address 0 and increments to the final count. The command in the CMD register is executed each time the FAR registers are updated.

There are three types of write to the Frame Address Registers:

1. Write one word to FAR_MAJ (only update the FAR_MAJ)
2. Write one word to FAR_MIN (only update the FAR_MIN)
3. Write two words to FAR_MAJ (update FAR_MAJ with first data and FAR_MIN with second data)

Table 17-12: Frame Address Register Description

Address Type	Bit Index	Description
FAR_MAJ Register		
Reserved	15:13	Reserved FAR register bits. Always leave these bits set to 0.
Block Type	12:10	Block types include CLBs & I/O, block RAM, etc.
Reserved	9:8	Reserved FAR register bits. Always leave these bits set to 0.
Major Address	7:0	Selects a major column. Column addresses start at 0 on the left and increase to the right.
FAR_MIN Register		
Reserved	15:8	Reserved FAR register bits. Always leave these bits set to 0.
Minor Address	7:0	Selects a memory-cell address line within a major column.

Status Register (STAT)

The Status Register indicates the value of numerous global signals. The register can be read through the SelectMAP or JTAG interfaces. A detailed explanation of each bit position is given in [Table 17-13](#).

Table 17-13: Status Register Description

Name	Bit Index	Description
SYNC_TIMEOUT	15	Configuration Watchdog Timer expired. 0: Default 1: Failed to find SYNC word before counter time out
SEU_ERR	14	Post-configuration CRC check error. 0: Default 1: POST_CRC error
DONE	13	Value on DONE pin.
INIT	12	Value on INIT_B pin.
MODE	11:9	Status of the MODE pins (M2:M0).
VSEL	8:6	Value on SPI Variant Select pins (VS2:VS0).
GHIGH_B	5	Status of GHIGH_B, asserted during configuration to disable interconnect and prevent contention. 0: GHIGH_B asserted 1: GHIGH_B deasserted
GWE	4	Status of Global Write Enable. 0: Flip-flops and block RAM are write disabled 1: Flip-flops and block RAM are write enabled
GTS_CFG_B	3	Status of Global Three-State. 0: All I/Os are placed in high-Z state 1: All I/Os behave as configured
DCM_LOCK	2	0: DCMs not locked 1: DCMs are locked This is a logical AND function of all DCM LOCKED signals. Unused DCM LOCKED signals = 1.
ID_ERROR	1	Attempt to write to FDRI without successful DEVICE_ID check. 0: No ID_ERROR 1: ID_ERROR
CRC_ERROR	0	0: No CRC error 1: CRC error

Configuration Options Registers (COR1 and COR2)

The Configuration Options Registers are used to set certain configuration options for the device. The name of each bit position in COR1 and COR2 is described in [Table 17-14](#).

Table 17-14: Configuration Options Register Description

Name	Bit Index	Description	Default
COR1			
DRIVE_AWAKE	15	0: Do not drive AWAKE pin (drive Low or disable) 1: Actively drive AWAKE pin (drive Low or High)	0
Reserved	14:5	Reserved COR register bits. Always leave these bits set to 0111111000.	0111111000

Table 17-14: Configuration Options Register Description (Cont'd)

Name	Bit Index	Description	Default
CRC_BYPASS	4	0: CRC enabled. 1: CRC disabled.	0
DONE_PIPE	3	0: No pipeline stage for DONEIN 1: Add pipeline stage for DONEIN The FPGA waits on DONE that is delayed by one StartupClk cycle. Use this option when StartupClk is running at high speeds.	0
DRIVE_DONE	2	0: DONE pin is open drain 1: DONE is actively driven High	0
SSCLKSRC	1:0	Startup-sequence clock source. 00: CCLK 01: UserClk (per connection on the STARTUP_SPARTAN3A block) 1x: JTAGClk	00
COR2			
RESET_ON_ERR	15	0: Do not fall back 1: On CRC error, fall back and retry first configuration file	0
Reserved	14	Reserved COR register bit. Always leave this bit set to 0.	0
BPI_DIV8	13	0: Update BPI address at every bus_clk cycle 1: Update BPI address every 8th bus_clk cycle (for serial daisy chains)	0
SINGLE	12	0: Readback is not single-shot New captured values are loaded on each successive CAP assertion on the CAPTURE_SPARTAN3A primitive. Capture can also be performed with the GCAPTURE instruction in the CMD register. 1: Readback is single-shot. The RCAP instruction must be loaded into the CMD register between successive readbacks.	0
DONE_CYCLE	11:9	Startup cycle to release the DONE pin. 001: Startup cycle 1 010: Startup cycle 2 011: Startup cycle 3 100: Startup cycle 4 101: Startup cycle 5 110: Startup cycle 6	100

Table 17-14: Configuration Options Register Description (Cont'd)

Name	Bit Index	Description	Default
LOCK_CYCLE	8:6	Startup cycle to stall in until DCMs lock. 000: Startup cycle 0 001: Startup cycle 1 010: Startup cycle 2 011: Startup cycle 3 100: Startup cycle 4 101: Startup cycle 5 110: Startup cycle 6 111: No Wait	111
GTS_CYCLE	5:3	Startup cycle to deassert the Global Three-State (GTS) signal. 001: Startup cycle 1 010: Startup cycle 2 011: Startup cycle 3 100: Startup cycle 4 101: Startup cycle 5 110: Startup cycle 6	101
GWE_CYCLE	2:0	Startup phase to deassert the Global Write Enable (GWE) signal. 001: Startup cycle 1 010: Startup cycle 2 011: Startup cycle 3 100: Startup cycle 4 101: Startup cycle 5 110: Startup cycle 6	110

Suspend Options Register (PWRDN_REG)

The Suspend Options Register (PWRDN_REG) is used to set certain options for the device Suspend and Awake features. The name of each bit position in PWRDN_REG is described in [Table 17-15](#).

Table 17-15: Suspend Options Register Description

Name	Bit Index	Description	Default
WAKE_DELAY2	13:9	Wakeup cycle delay 2.	00100
WAKE_DELAY1	8:6	Wakeup cycle delay 1.	010
FILTER_B	5	0: SUSPEND filter on. 1: SUSPEND filter off.	0
EN_PGSR	4	0: No GSR pulse during wakeup. 1: Generate GSR pulse during wakeup.	0
Unused	3		0
EN_PWRDN	2	0: Suspend disabled. 1: Suspend enabled.	0

Table 17-15: Suspend Options Register Description (Cont'd)

Name	Bit Index	Description	Default
EN_PORB	1	Allow power supply rails to be lowered. 0: Enable Power-On-Reset. 1: Disable Power-On-Reset. (not recommended)	0
KEEP_SCLK	0	Use Configuration Startup clock for Suspend Startup. 0: Use Master CCLK for Startup sequence. 1: Use SSCLKSRC for Startup sequence.	1

Housecleaning Options Register (HC_OPT_REG)

The Housecleaning Options Register (HC_OPT_REG) is used to set certain options for device housecleaning or reset of the configuration memory during initialization. The name of each bit position in HC_OPT_REG is described in Table 17-16.

Table 17-16: Housecleaning Options Register Description

Name	Bit Index	Description	Default
BRAM_SKIP	5	0: Reset block RAM. 1: Skip reset of block RAM.	0
TWO_ROUND	4	0: 1 round of housecleaning. 1: 2 rounds of housecleaning.	1
HC_CYCLE	3:0	Number of housecleaning cycles.	1111

Multiple Frame Write Register (MFWR)

The Multiple Frame Write (MFWR) Register is used with the BitGen -g Compress option. If more than one frame has identical data, it is possible to load that frame into the configuration logic, and instruct the logic to load the frame into multiple address locations. Depending on the utilization of the device, this may decrease the size of the bitstream considerably. This feature is only supported upon initial configuration. Therefore, to reconfigure the device with this feature, the part must be power-cycled or reset with PROG_B. To write multiple frames with the same data, the following steps need to be performed:

1. Write the WCFG command to the CMD register.
2. Write a desired frame to the FDRI register.
3. Write the FAR register with the first desired address.
4. Write the MFWR command to the CMD register.
5. Write two dummy words to the MFWR register.
6. Write the FAR register with the second desired address.
7. Write two dummy words to the MFWR register.
8. Repeat steps 6 and 7 until the last desired address is reached.

Bitstream Composition

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 serial/parallel configuration modes or by issuing the JPROGRAM_B instruction for JTAG configuration mode. Configuration proceeds as shown in [Table 17-17](#):

Table 17-17: Configuration Sequence

Configuration Data (hex)	Explanation
FFFF	Dummy word
AA99	Sync word
30A1	Type 1 write 1 word to CMD
0007	RCRC command
3321	Type 1 write 1 word CCLK_FREQ
05BE	CCLK frequency
31A1	Type 1 write 1 word FLR
0XXX	Frame length
3141	Type 1 write 1 word to COR1
3F00	Configuration Option Register 1
3161	Type 1 write 1 word to COR2
09EE	Configuration Option Register 2
31C2	Type 1 write 2 words to IDCODE
XXXX	MSB Device_ID
XXXX	LSB Device_ID
30E1	Type 1 write 1 word to MASK
0000	Data word 0
30C1	Type 1 write 1 word to CTL
0001	Data word 0
3181	Type 1 write 1 word to PWRDN_REG
0881	Data word 0
3201	Type 1 write 1 word to HC_OPT_REG
001F	Data word 0
32C1	Type 1 write 1 word to PU_GWE
0006	Data word 0
32E1	Type 1 write 1 word to PU_GTS
0005	Data word 0
32A1	Type 1 write 1 word to MODE_REG

Table 17-17: Configuration Sequence (Cont'd)

Configuration Data (hex)	Explanation
000E	Data word 0
3261	Type 1 write 1 word to GENERAL1_REG
0000	Data word 0
3281	Type 1 write 1 word to GENERAL2_REG
0000	Data word 0
3301	Type 1 write 1 word to SEU_OPT_REG
3FF0	Data word 0
3362	Type 1 write 2 words to EXP_SIGN_REG
0000	Data word 0
0000	Data word 1
3022	Type 1 write 2 words to FAR_MAJ
0XXX	FAR_MAJ
00XX	FAR_MIN
30A1	Type 1 write 1 word to CMD
0001	WCFG command
5062	Type 2 write 0 words to FDRI
0XXX	Word count 1 (MSB)
XXXX	Word count 2 (LSB)
XXXX	Data 1
...	...
XXXX	Data (word count)
3001	Type 1 write 1 word to CRC
XXXX	Data word 0
30A1	Type 1 write 1 word to CMD
000A	GRESTORE command
30A1	Type 1 write 1 word to CMD
0003	LFRM command
2000	Type 1 NOOP
...	Type 1 NOOPs
2000	Type 1 NOOP
30A1	Type 1 write 1 word to CMD
0005	START command
30C1	Type 1 write 1 word to CTL
0001	Data word 0
3001	Type 1 write 1 word to CRC

Table 17-17: Configuration Sequence (Cont'd)

Configuration Data (hex)	Explanation
XXXX	Data word 0
30A1	Type 1 write 1 word to CMD
000D	DESYNC command
2000	Type 1 NOOP
2000	
2000	
2000	
2000	
2000	
2000	
2000	
2000	

Readback

This chapter provides more extensive details on the readback logic in the Extended Spartan®-3A family, including the Spartan-3A, Spartan-3AN, and Spartan-3A DSP platforms. The Spartan-3E family is similar, and details on the Spartan-3 family can be found in [XAPP452](#).

Spartan®-3, Spartan-3E, and Extended Spartan-3A devices allow users to read back the configuration memory through the SelectMAP (slave parallel) or JTAG (IEEE 1149.1 boundary scan) interfaces. 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 (distributed LUT RAM, SRL16 shift registers, and block RAM). 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, and is useful for design debugging.

To read configuration memory, users must send a sequence of commands to the device to initiate the readback procedure; once initiated the device dumps the contents of its configuration memory to the SelectMAP or JTAG interface. The configuration memory read procedure sections for SelectMAP, IEEE 1149.1 JTAG, and IEEE 1532 JTAG describe the steps for reading configuration memory.

Users can send the readback command sequence from a custom 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® software), can perform all readback and comparison functions and report to the user whether there were any configuration errors. iMPACT cannot perform capture operations, although Readback Capture is seldom used for design debugging because the ChipScope™ Pro tool's ILA core, sold separately through the Xilinx website, provides superior design debugging functionality in a user-friendly interface.

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. The “[Verifying Readback Data](#)” section explains how this is done.

Preparing a Design for Readback

To prepare a design for readback, the Bitstream Generator (BitGen) security setting must not prohibit readback (`-g security:none`). 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 the security requirement, no special considerations are necessary to enable readback through the Boundary-Scan port.

If capture functionality is needed, the CAPTURE_SPARTAN3A primitive can be instantiated in the user design (see “[Readback Capture \(CAPTURE\)](#)” in [Chapter 13](#)). 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. Alternatively, writing the GCAPTURE command to the CMD register can be used (see [Readback Capture](#)). The register values are later read out of the device along with all other configuration memory.

Readback Command Sequences

Configuration memory is read from the FDRO (Frame Data Register - Output) configuration register and can be accessed from the JTAG and SelectMAP interfaces. Readback is possible while the FPGA design is active or in a shutdown state, although block RAMs cannot be accessed by the user design while they are being accessed by the configuration logic.

Accessing Configuration Registers through the SelectMAP Interface

To read configuration memory through the SelectMAP interface, the user must set the interface for write control to send commands to the FPGA, and then switch the interface to read control to read data from the device. 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 CSI_B signal (CS_B in the Spartan-3 family) must be deasserted (CSI_B = 1) before toggling the RDWR_B signal, otherwise the user causes an abort (refer to “[SelectMAP ABORT](#)” in [Chapter 7](#) for details).

The procedure for changing the SelectMAP interface from Write to Read control, or vice versa, is:

1. Deassert CSI_B.
2. Toggle RDWR_B.
RDWR_B = 0: Write control
RDWR_B = 1: Read control
3. Assert CSI_B.
4. This procedure is illustrated in [Figure 18-1](#).

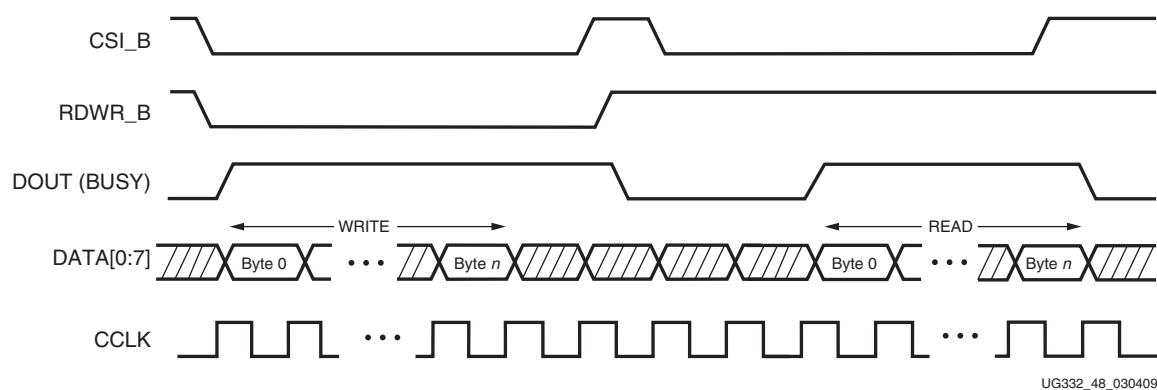


Figure 18-1: Changing the SelectMAP Port from Write to Read Control

The Extended Spartan-3A family has a DOUT pin which operates like the BUSY pin in earlier families during readback. The BUSY signal goes Low to indicate when the DATA bus contains valid readback data. However, it is not necessary to monitor BUSY since the readback data is always valid three CCLK cycles after CSI_B is asserted.

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. The procedure for reading the STAT register through the SelectMAP interface in the Extended Spartan-3A family follows (for the Spartan-3 and Spartan-3E families, refer to [XAPP452](#)):

1. Write the Synchronization word to the device.
2. Write the *read STAT register* packet header to the device.
3. Write two dummy words to the device to flush the packet buffer.
4. Read two bytes using SelectMAP; this is the Status register value.
5. Write the DESYNC command to the device
6. Write two dummy words to the device to flush the packet buffer.

Table 18-1: Status Register Readback Command Sequence (SelectMAP)

Step	SelectMAP Port Direction	Configuration Data	Explanation
1	Write	AA99	Sync Word
2	Write	2901	Read 1 word from STAT register
3	Write	2000	NOOP
		2000	NOOP
4	Read	SSSS	Device writes 1 word from the STAT register to the configuration interface
5	Write	30A1	Type 1 write 1 word to CMD
		000D	Desync command
6	Write	2000	NOOP
		2000	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 18-2](#).

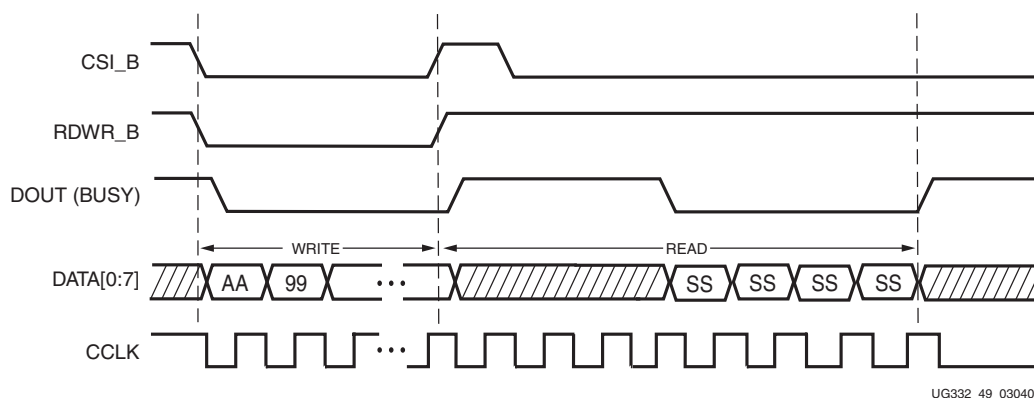


Figure 18-2: SelectMAP Status Register Read

To read registers other than STAT, the address specified in the Type 1 packet header in Step 2 of Table 18-1 should be modified and the word count changed if necessary. Reading from the FDRO register is a special case that is described in the next section.

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. Additional steps are needed to accommodate the configuration logic. Configuration data coming from the FDRO register passes through the frame buffer. The first frame of readback data should be discarded.

1. Write the Synchronization word to the device.
2. Write 1 NOOP command.
3. Write the RCRC command to the CMD register.
4. Write 2 NOOP commands.
5. Write the Shutdown command.
6. Write four NOOP instructions to ensure the shutdown sequence has completed. DONE goes Low during the shutdown sequence.
7. Write the RCFG command to the CMD register.
8. Write the Starting Frame Address to the FAR (typically 0x00000000)
9. Write the *read FDRO register* packet header to the device. The FDRO read length is given by:

$$\text{FDRO Read Length} = (\text{words per frame}) \times (\text{frames to read} + 1) + 1$$

One extra frame is read to account for the frame buffer. The frame buffer produces one dummy frame at the beginning of the read and one at the end. Also, one extra word is read in SelectMAP mode.

10. Write two dummy words to the device to flush the packet buffer.
11. Read the FDRO register from the SelectMAP interface. The FDRO read length is the same as in step 9 above.
12. Write one NOOP instruction.
13. Write the START command.
14. Write the RCRC command.
15. Write the DESYNC command.

16. Write at least 32 bits of NOOP commands to flush the packet buffer. Continue sending CCLK pulses until DONE goes High.

Table 18-2 shows the readback command sequence.

Table 18-2: Shutdown Readback Command Sequence (SelectMAP)

Step	SelectMAP Port Direction	Configuration Data	Explanation
1	Write	AA99	Sync word
2	Write	2000	Type 1 NOOP word 0
3	Write	30A1	Type 1 write 1 word to CMD
		0007	RCRC command
4	Write	2000	Type 1 NOOP word 0
		2000	Type 1 NOOP word 1
5	Write	30A1	Type 1 write 1 word to CMD
		000B	SHUTDOWN command
6	Write	2000	Type 1 NOOP word 0
		2000	Type 1 NOOP word 1
		2000	Type 1 NOOP word 2
		2000	Type 1 NOOP word 3
7	Write	30A1	Type 1 write 1 word to CMD
		0004	RCFG command
8	Write	3022	Type 1 write 2 words to FAR_MAJ
		0000	FAR_MAJ Address = 0000
		0000	FAR_MIN Address = 0000
9	Write	2880	Type 1 read 0 words from FDRO
		XXXX	Type 2 read n words from FDRO
10	Write	2000	Type 1 NOOP word 0
		2000	Type 1 NOOP word 1
11	Read	0000	Packet data read FDRO word 0
		...	
		0000	Packet data read FDRO word n
12	Write	2000	Type 1 NOOP word 0
13	Write	30A1	Type 1 write 1 word to CMD
		0005	START command
14	Write	30A1	Type 1 write 1 word to CMD
		0007	RCRC command

Table 18-2: Shutdown Readback Command Sequence (SelectMAP) (Cont'd)

Step	SelectMAP Port Direction	Configuration Data	Explanation
15	Write	30A1	Type 1 write 1 word to CMD
		000D	DESYNC command
16	Write	2000	Type 1 NOOP word 0
		2000	Type 1 NOOP word 1

Accessing Configuration Registers through the JTAG Interface

JTAG access to the 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 into the configuration logic by going through the CFG_IN register; configuration memory is read out through the CFG_OUT register. The JTAG state transitions for accessing the CFG_IN and CFG_OUT registers are described in [Table 18-3](#).

Table 18-3: Shifting in the JTAG CFG_IN and CFG_OUT Instructions

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
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)	0	5
		00100 (CFG_OUT)		
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	1	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 1s 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. 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 one NOOP instruction to the device.
 - c. Write the *read STAT register* packet header to the device.
 - d. Write two dummy 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.

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 16 bits out of the Status register through the Shift-DR state.
6. Reset the TAP controller.

Table 18-4: 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 SHIFT-IR.	0	1	1
	Move into the SELECT-DR state.	X	1	2
	Move into the SHIFT-DR state.	X	0	2

Table 18-4: Status Register Readback Command Sequence (JTAG) (Cont'd)

Step	Description	Set and Hold		# of Clocks (TCK)
		TDI	TMS	
3	Shift configuration packets into the CFG_IN data register, MSB first.	a: 0xAA99 b: 0x2000 c: 0x2901 d: 0x2000 0x2000	0	79
	Shift the LSB of the last configuration packet while exiting SHIFT-DR.	0	1	1
	Move into the SELECT-IR state.	X	1	3
	Move into the SHIFT-IR state.	X	0	2
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 Shift-IR.	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.	0xSSSS	0	31
	Shift the last bit of the STAT register out of the CFG_OUT data register while exiting SHIFT-DR.	S	1	1
	Move into the Select-IR state.	X	1	3
	Move into the Shift-IR State.	X	0	2
6	Reset the TAP Controller.	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 *dummy data* and should be discarded. The 1149.1 JTAG readback flow is recommended for most users.

1. Reset the TAP controller.
2. 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.
3. Shift packet write commands into the CFG_IN register through the Shift-DR state:
 - a. Write a dummy word to the device.
 - b. Write the Synchronization word to the device.
 - c. Write a NOOP instruction to the device.

- d. Write the RCRC command to the device.
- e. Write two dummy words to flush the packet buffer.
- 4. Shift the JSHUTDOWN instruction into the JTAG Instruction Register.
- 5. Move into the RTI state; remain there for 12 TCK cycles to complete the Shutdown sequence. The DONE pin goes Low during the Shutdown sequence.
- 6. Shift the CFG_IN instruction into the JTAG Instruction Register.
- 7. Move to the Shift-DR state and shift packet write commands into the CFG_IN register:
 - a. Write a dummy word to the device.
 - b. Write the Synchronization word to the device.
 - c. Write a NOOP instruction to the device.
 - d. Write the *write CMD register* header.
 - e. Write the RCFG command to the device.
 - f. Write the *write FAR register* header.
 - g. Write the starting frame address to the FAR register (typically 0x00000000).
 - h. Write the *read FDRO register* Type-1 packet header to the device.
 - i. Write a Type-2 packet header to indicate the number of words to read from the device.
 - j. Write two dummy 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.

- 8. Shift the CFG_OUT instruction into the JTAG Instruction Register through the Shift-DR 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.
- 9. Shift frame data from the FDRO register through the Shift-DR state.
- 10. Reset the TAP controller.

Table 18-5: 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 CFG_IN instruction, LSB first.	00101	0	5
	Shift the MSB of the CFG_IN instruction while exiting Shift-IR.	1	1	1
	Move into the SELECT-DR state.	X	1	2
	Move into the SHIFT-DR state.	X	0	2

Table 18-5: Shutdown Readback Command Sequence (JTAG) (Cont'd)

Step	Description	Set and Hold		# of Clocks (TCK)
		TDI	TMS	
3	Shift configuration packets into the CFG_IN data register, MSB first.	a: 0xFFFF b: 0xAA99 c: 0x2000 d: 0x30A1 0x0007 e: 0x2000 0x2000	0	111
	Shift the LSB of the last configuration packet while exiting SHIFT-DR.	0	1	1
	Move into the SELECT-IR State.	X	1	3
	Move into the SHIFT-IR State.	X	0	2
4	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 to RTI.	X	1	1
			0	1
5	Remain in RTI 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
6	Shift the first 5 bits of the CFG_IN instruction, LSB first.	00101	0	5
	Shift the MSB of the CFG_IN instruction while exiting SHIFT-IR.	1	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: 0xFFFF b: 0xAA99 c: 0x2000 d: 0x30A1 e: 0x0004 f: 0x3022 g: 0x0000 0x0000 h: 0x2880 i: 0xFFFF j: 0x2000 0x2000	0	191
	Shift the LSB of the last configuration packet while exiting SHIFT-DR.	0	1	1
	Move into the SELECT-IR State.	X	1	3
	Move into the SHIFT-IR State.	X	1	3
			0	2

Table 18-5: Shutdown Readback Command Sequence (JTAG) (Cont'd)

Step	Description	Set and Hold		# of Clocks (TCK)
		TDI	TMS	
8	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 Shift-IR.	0	1	1
	Move into the SELECT-DR state.	X	1	2
	Move into the SHIFT-DR state.	X	0	2
9	Shift the contents of the FDRO register out of the CFG_OUT data register.	...	0	number of readback bits – 1
	Shift the last bit of the FDRO register out of the CFG_OUT data register while exiting SHIFT-DR.	X	1	1
	Move into the Select-IR state.	X	1	3
	Move into the Shift-IR State.	X	0	2
10	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 through the CFG_IN JTAG register, rather the ISC_READ JTAG register is used to read configuration memory directly.

At the end of 1532 JTAG readback, CRC Error status must be cleared by issuing the Reset CRC command or writing the correct CRC value to the CRC register. The 1532 JTAG readback procedure is illustrated in [Figure 18-3](#).

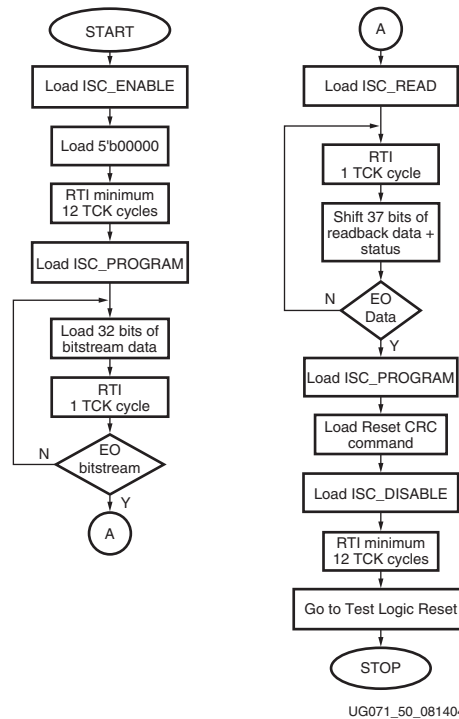


Figure 18-3: IEEE 1532 JTAG Readback Flow

Verifying Readback Data

Table 18-6 lists the readback files.

Table 18-6: Readback Files

File Extension	File Type	BitGen Setting	Description
.rba	ASCII	-b and -g Readback	An ASCII file that contains readback commands, rather than configuration commands, and expected readback data where the configuration data normally is. 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.

Table 18-6: Readback Files (Cont'd)

File Extension	File Type	BitGen Setting	Description
.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.
.11	ASCII	-1	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 and the .rbd file contains only expected readback data. Systems that use an .rbd file for readback must store readback commands elsewhere. The actual readback data must be masked against an .msk or .msd mask file, as certain bits in 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.

The readback data stream contains configuration frame data that are preceded by one frame of pad data, as described in [Configuration Memory Read Procedure \(SelectMAP\)](#). 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 18-4](#).

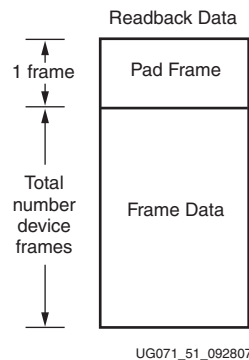


Figure 18-4: Readback Data Stream

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, because these can 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 by 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. See [Figure 18-5](#).

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 can convert the `.rbd` and `.msd` files to a binary format using a script or text editor, to simplify the verify procedure for some systems and to reduce the size of the files by a factor of eight.

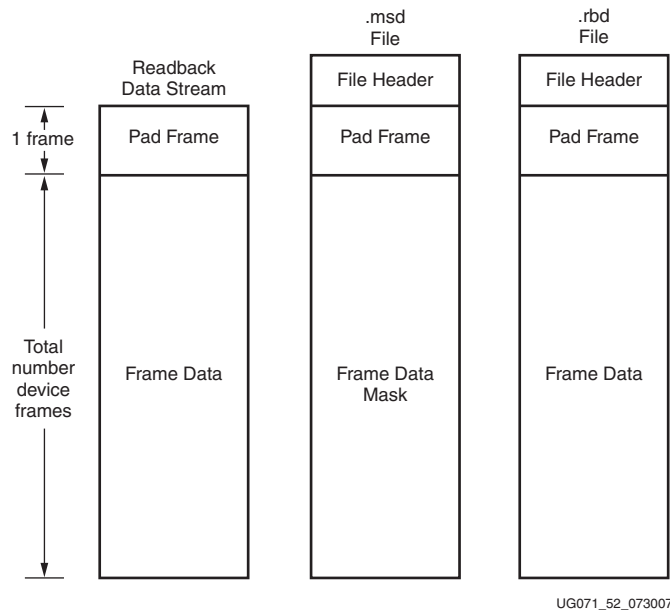


Figure 18-5: Comparing Readback Data Using the `.msd` and `.rbd` Files

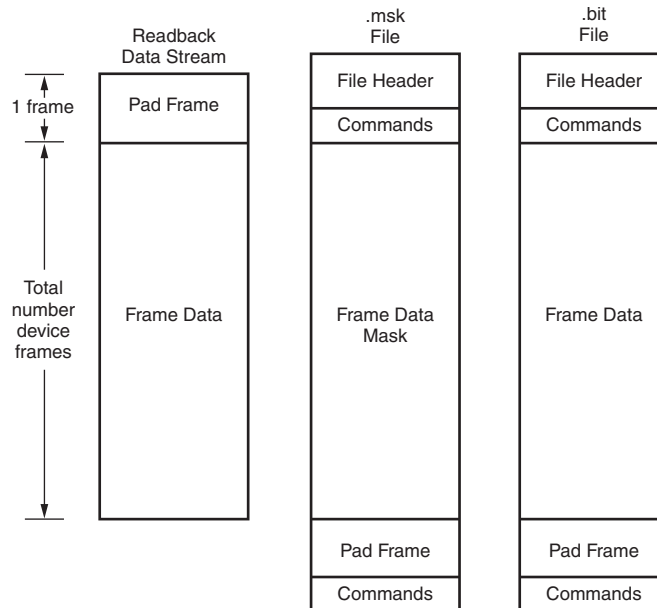
The drawback to this approach is that in addition to storing the initial configuration bitstream and the `.msd` file, the golden `.rbd` file must be stored somewhere, increasing the overall storage requirement.

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. See [Figure 18-6](#).

This approach requires the least in-system storage space, because only the `.bit`, `.msk`, and readback commands must be stored.



UG071 53 073007

Figure 18-6: 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.

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_SPARTAN3A primitive in their design (Figure 18-7) and asserting the CAP input on that primitive while the design is operating. On the next rising clock edge on the CAPTURE_SPARTAN3A 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. Register values are stored in the same memory cell that programs the register's initial state configuration. Therefore, sending the GRESTORE command to the configuration logic after the Capture sequence can cause registers to return to an unintended state.

Alternatively, the GRDBK signal can be asserted by writing the GCAPTURE command to the CMD register. This command asserts the GRDBK signal for two CCLK or TCK cycles, depending on the Startup clock setting.

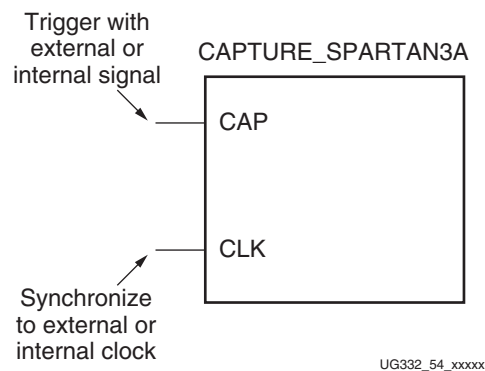


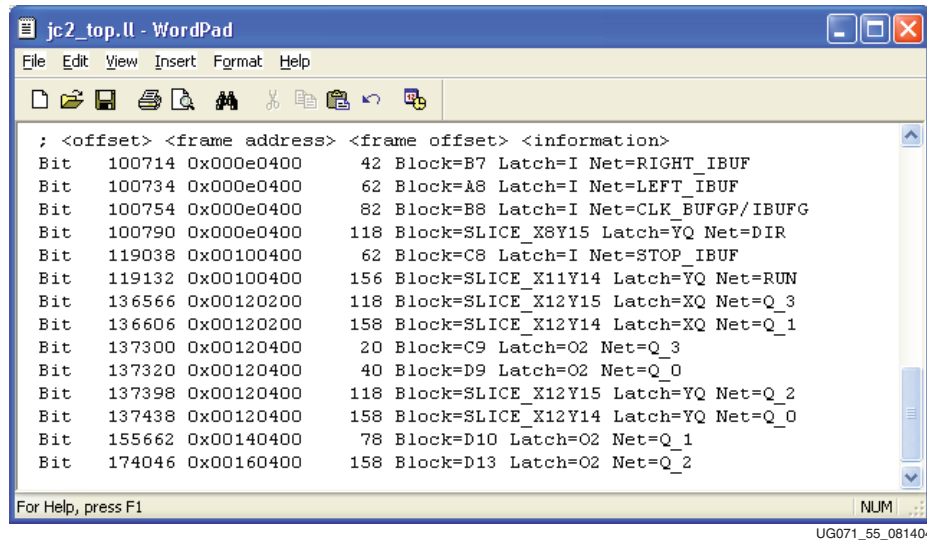
Figure 18-7: Extended Spartan-3A Library Primitive

Table 18-7: Capture Signals

Signal	Description	Access
GCAPTURE	Captures the state of all slice and IOB registers. Complement of GRESTORE.	GCAPTURE command through the CMD register or CAP input on capture block, user controlled.
GRESTORE	Initializes all registers as configured.	CMD register and STARTUP_SPARTAN3A block.

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_SPARTAN3A 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 in Figure 18-8.



```

; <offset> <frame address> <frame offset> <information>
Bit 100714 0x000e0400 42 Block=B7 Latch=I Net=RIGHT_IBUF
Bit 100734 0x000e0400 62 Block=A8 Latch=I Net=LEFT_IBUF
Bit 100754 0x000e0400 82 Block=B8 Latch=I Net=CLK_BUFGP/IBUFG
Bit 100790 0x000e0400 118 Block=SLICE_X8Y15 Latch=YQ Net=DIR
Bit 119038 0x00100400 62 Block=C8 Latch=I Net=STOP_IBUF
Bit 119132 0x00100400 156 Block=SLICE_X11Y14 Latch=YQ Net=RUN
Bit 136566 0x00120200 118 Block=SLICE_X12Y15 Latch=XQ Net=Q_3
Bit 136606 0x00120200 158 Block=SLICE_X12Y14 Latch=XQ Net=Q_1
Bit 137300 0x00120400 20 Block=C9 Latch=O2 Net=Q_3
Bit 137320 0x00120400 40 Block=D9 Latch=O2 Net=Q_0
Bit 137398 0x00120400 118 Block=SLICE_X12Y15 Latch=YQ Net=Q_2
Bit 137438 0x00120400 158 Block=SLICE_X12Y14 Latch=YQ Net=Q_0
Bit 155662 0x00140400 78 Block=D10 Latch=O2 Net=Q_1
Bit 174046 0x00160400 158 Block=D13 Latch=O2 Net=Q_2

```

For Help, press F1 NUM

UG071_55_081404

Figure 18-8: Logic Allocation File Format

Figure 18-8 shows a snippet from a logic allocation file for an 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 readback frame data. 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 18-8) that is located in Slice X8Y15 is located at bit offset 100790.

Note that captured flip-flop values, along with distributed RAM and SRL16 values, are stored in their inverted sense.

