Configuration Handbook, Volume 2

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com

CF5V2-2.2

Contents

Section IV. Board Layout Tips & Debugging Techniques

Chapter Revision Dates

The chapters in this book, *Configuration Handbook, Vol. 2*, were revised on the following dates. Where chapters or groups of chapters are available separately, part numbers are listed.

Chapter 1. Altera Configuration Devices
Revised: *August 2005*
Part number: *CF52001-2.2*

Chapter 2. Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet
Revised: *August 2005*
Part number: *CF52002-2.1*

Chapter 3. Altera Enhanced Configuration Devices
Revised: *August 2005*
Part number: *S52014-2.1*

Chapter 4. Serial Configuration Devices (EPCS1, EPCS4, EPCS16, & EPCS64) Features
Revised: *August 2005*
Part number: *C51014-1.5*

Chapter 5. Configuration Devices for SRAM-Based LUT Devices Data Sheet
Revised: *August 2005*
Part number: *CF52005-2.1*

Chapter 6. Device Configuration Options
Revised: *August 2005*
Part number: *CF52006-2.1*

Chapter 7. Configuration File Formats
Revised: *August 2005*
Part number: *CF52007-2.1*

Chapter 8. Configuring Mixed Altera FPGA Chains
Revised: *August 2005*
Part number: *CF52008-2.1*

Chapter 9. Combining Different Configuration Schemes
Revised: *August 2005*
Part number: *CF52009-2.1*

Chapter 10.	Using Flash Memory to Configure FPGAs	
	Revised:	*August 2005*
	Part number:	*CF52010-2.1*
Chapter 11.	Debugging Configuration Problems	
	Revised:	*August 2005*
	Part number:	*CF52011-2.1*

About this Handbook

This handbook provides comprehensive information about configuring Altera® FPGAs and configuration devices.

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at **www.altera.com.** For technical support on this product, go to **www.altera.com/mysupport**. For additional information about Altera products, consult the sources shown below.

Information Type	USA & Canada	All Other Locations
Technical support	**www.altera.com/mysupport/**	**www.altera.com/mysupport/**
	(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	+1 408-544-8767 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
Product literature	**www.altera.com**	**www.altera.com**
Altera literature services	literature@altera.com	literature@altera.com
Non-technical customer service	(800) 767-3753	+ 1 408-544-7000 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
FTP site	**ftp.altera.com**	**ftp.altera.com**

Typographic Conventions

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, web addresses, and software utility names are shown in bold type. Examples: **f_{MAX}**, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*

Visual Cue	Meaning
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>*.**pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: "Typographic Conventions."
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input`. Active-low signals are denoted by suffix `n`, e.g., `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ● •	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
⚠ CAUTION	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.
⚠	The warning indicates information that should be read prior to starting or continuing the procedure or processes
↵	The angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.

Section I. FPGA Configuration Devices

This section provides information on Altera® configuration devices. The following chapters contain information about how to use these devices, feature descriptions, device pin tables, and package diagrams.

This section includes the following chapters:

- Chapter 1. Altera Configuration Devices
- Chapter 2, Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet
- Chapter 3, Altera Enhanced Configuration Devices
- Chapter 4, Serial Configuration Devices (EPCS1, EPCS4, EPCS16, & EPCS64) Features
- Chapter 5, Configuration Devices for SRAM-Based LUT Devices Data Sheet

Revision History

The table below shows the revision history for Chapters 1 through 5.

Chapter	Date/Version	Changes Made
1	August 2005, v2.2	● Removed active cross references refering to document outside Chapter 1.
	February 2005, v2.1	● Document format updated. ● Updated Table 1–2.
	July 2004, v2.0	● Added Stratix II and Cyclone II device information throughout chapter. ● Added EPCS16 and EPCS64 device information throughout chapter. ● Updated data size for EP1C4 and all APEX II devices.
	September 2003, v1.0	Initial Release.

Chapter	Date/Version	Changes Made
2	August 2005, v2.1	● Removed active cross references refering to document outside Chapter 2.
	July 2004, v2.0	● Added Stratix II and Cyclone II device information throughout chapter. ● Updated VCCW connection in Figures 2–2, 2–3, and 2–4. ● Updated Note (1) of Figures 2–2, 2–3, and 2–4. ● Updated Note (4) of Table 2–13. ● Updated unit of ICC0 in Table 2–17. ● Added ICCW to Table 2–17.
	September 2003, v1.0	Initial Release.
3	August 2005, v21	Changed active cross references to text.
	July 2004, v1.1	● Added text regarding pointing to an incorrect page after Figure 3–8. ● Renamed .**hexpof** to .**hexout** throughout chapter.
	September 2003, v1.0	Initial Release.
4	August 2005, v1.5	● Changed active cross references to text.
	January 2005, v1.4	● Added EPCS16 and EPCS64 device information throughout chapter. ● Added information about reading and writing Raw Programming Data files (**.rpd**).
	July 2004, v1.2	● Added Stratix II and Cyclone II device information throughout chapter. ● Added EPCS16 and EPCS64 device information throughout chapter. ● Added USB Blaster information throughout chapter. ● Added 16-pin SOIC package pin information throughout chapter.
	October 2003, v1.1	● Added Serial Configuration Device Memory Access section. ● Updated timing information in Tables 4–12 and 4–15.
	September 2003, v1.0	Initial Release.
5	August 2005, v2.1	Removed active cross references refering to document outside Chapter 5.
	July 2004, v2.0	Added Stratix II and Cyclone II device information throughout chapter.
	September 2003, v1.0	Initial Release.

1. Altera Configuration Devices

CF52001-2.2

Introduction

During device operation, Altera® FPGAs store configuration data in SRAM cells. Because SRAM memory is volatile, the SRAM cells must be loaded with configuration data each time the device powers up. You can configure Stratix® series, Cyclone™ series, APEX™ II, APEX 20K, Mercury™, ACEX® 1K, FLEX® 10K, and FLEX 6000 devices using data stored in an Altera configuration device. Altera configuration devices are offered in different densities and provide a variety of features.

The Altera enhanced configuration devices (EPC16, EPC8, and EPC4) support a single-device configuration solution for high-density FPGAs, including Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10K devices. The enhanced configuration devices are ISP-capable through its Joint Test Action Group (JTAG) interface. Enhanced configuration devices cannot be cascaded nor can the flash interface pins be shared between multiple enhanced configuration devices for implementing a shared bus interface.

The enhanced configuration devices are divided into two major blocks, the controller and the flash memory. Because of its large flash memory size and decompression feature, enhanced configuration devices hold configuration data for one or multiple Altera FPGAs. In addition, unused portions of the flash can be used as memory storage for a programmable logic device (PLD) or processor (e.g., Nios® processor). After configuration, access to the flash memory is through the external flash interface of the enhanced configuration devices. Fast passive parallel (FPP) configuration, where configuration data is sent byte-wide on the `DATA[7..0]` pins every clock cycle, is also supported for fast configuration times. FPP configuration is supported in Stratix series, and APEX II devices.

For information on enhanced configuration devices, refer to *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* and *Using Altera Enhanced Configuration Device*.

The Altera serial configuration devices (EPCS4, EPCS1, EPCS16, and EPCS64) support a single-device configuration solution for Stratix II FPGAs and the Cyclone series. Serial configuration devices offer a low cost, low pin count configuration solution for Stratix II FPGAs and the Cyclone series. The serial configuration devices cannot be cascaded. Unused portions of the flash can be accessed using the Nios processor.

For information on serial configuration devices, refer to *Serial Configuration Devices (EPCS1, EPCS4, EPCS16, & EPCS64) Data Sheet*.

The EPC2, EPC1 and EPC1441 configuration devices provide configuration support for Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10K devices. The EPC2 device is ISP-capable through its JTAG interface. This enables you to program them on board for a quick and efficient prototyping environment. The EPC2 and EPC1 can be cascaded to hold large configuration files.

For more information on EPC2, EPC1, and EPC1441 configuration devices, refer to *Configuration Devices for SRAM-Based LUT Devices Data Sheet*.

Choosing a Configuration Device

Table 1–1 summarizes the features of the Altera configuration devices and the amount of configuration space they hold. Note that not every configuration device can be used to configure every Altera device. For example, EPCS devices can only be used to configure Stratix II or Cyclone series devices.

Table 1–1. Altera Configuration Devices

Device	Memory Size (bits)	On-Chip Decompression Support	ISP Support	Daisy Chain Support	Reprogrammable	Operating Voltage (V)
EPC16	16,777,216	Yes	Yes	No	Yes	3.3
EPC8	8,388,608	Yes	Yes	No	Yes	3.3
EPC4	4,194,304	Yes	Yes	No	Yes	3.3
EPCS64	67,108,864 *(1)*	No	No *(2)*	No	Yes	3.3
EPCS16	16,777,216 *(1)*	No	No *(2)*	No	Yes	3.3
EPCS4	4,194,304	No	No *(2)*	No	Yes	3.3
EPCS1	1,048,576	No	No *(2)*	No	Yes	3.3
EPC2	1,695,680	No	Yes	Yes	Yes	5.0 or 3.3
EPC1	1,046,496	No	No	Yes	No	5.0 or 3.3
EPC1441	440,800	No	No	No	No	5.0 or 3.3

Notes to Table 1–1:
(1) This information is preliminary.
(2) The EPCS device can be re-programmed in system by an external microprocessor using SRunner. For more information about SRunner refer to the *SRunner: in Embedded Solution for EPCS Programming White Paper* on the Altera web site at **www.altera.com**.

To choose the appropriate configuration device, you need to determine the total configuration space required for your target FPGA or chain of FPGAs. These numbers are listed in each device family section. If you are configuring a chain of FPGAs, you need to add the configuration file size for each FPGA to determine the total configuration space needed. For example, if you are configuring an EP20K200E and an EP20K60E device in a daisy chain, your total configuration space requirement would be 1.964 Mbits + .641 Mbits = 2.605 Mbits. Next, use Table 1–1 to determine which configuration device fulfills your configuration space requirements.

Using the example above, to configure an EP20K400E and an EP20K60E device in a chain, 2.605 Mbits would fit in three EPC1 devices, two EPC2, or one EPC4 device. Only EPC2 and EPC1 devices can be cascaded. Using the configuration file size tables in each device family section along with Table 1–1, you can determine the number of configuration devices required to configure your target FPGA or chain of FPGAs.

Table 1–2 shows which configuration devices and how many are needed to configure each Altera FPGA.

Table 1–2. Configuration Devices Required (Part 1 of 4)

Family	Device	Data Size (Bits) (1)	EPC1064/ 1064V	EPC1213	EPC1441	EPC1	EPC2	EPC4 (2)	EPC8 (2)	EPC16 (2)	EPCS1	EPCS4	EPCS16	EPCS64
Stratix II (1.2 V) (5)	EP2S15	5,000,000					3	1	1	1			1	1
	EP2S30	10,100,000					7		1	1			1	1
	EP2S60	17,100,000					11			1			1 (4)	1
	EP2S90	27,500,000					17							1
	EP2S130	39,600,000					24							1
	EP2S180	52,400,000					31							1
Stratix (1.5 V)	EP1S10	3,534,640					3 (3)	1	1	1				
	EP1S20	5,904,832					4	1	1	1				
	EP1S25	7,894,144					5		1	1				
	EP1S30	10,379,368					7		1	1				
	EP1S40	12,389,632					8		1	1				
	EP1S60	17,543,968					11			1				
	EP1S80	23,834,032					15			1				
Stratix GX (1.5 V)	EP1SGX10	3,534,640					3	1	1	1				
	EP1SGX25	7,894,144					5		1	1				
	EP1SGX40	12,389,632					8		1	1				
Cyclone II (1.2 V) (5)	EP2C5	1,223,980					1	1	1	1	1 (4)	1	1	1
	EP2C8	1,983,792					2	1	1	1		1	1	1
	EP2C20	3,930,986					3	1	1	1		1	1	1
	EP2C35	7,071,234					5		1	1			1	1
	EP2C50	9,122,148					6		1	1			1	1
	EP2C70	10,249,694					7		1	1			1	1

Table 1–2. Configuration Devices Required (Part 2 of 4)

Family	Device	Data Size (Bits) *(1)*	EPC1064/ 1064V	EPC1213	EPC1441	EPC1	EPC2	EPC4 *(2)*	EPC8 *(2)*	EPC16 *(2)*	EPCS1	EPCS4	EPCS16	EPCS64
Cyclone (1.5 V)	EP1C3	627,376				1	1	1	1	1	1	1	1	
	EP1C4	924,512				1	1	1	1	1	1	1	1	
	EP1C6	1,167,216				1 *(4)*	1	1	1	1	1 *(4)*	1	1	
	EP1C12	2,326,528					1 *(4)*	1	1	1		1	1	
	EP1C20	3,559,608					2 *(4)*	1	1	1		1	1	
APEX II (1.5 V)	EP2A15	4,358,512					3	1	1	1				
	EP2A25	6,275,200					4	1	1	1				
	EP2A40	9,640,528					6		1	1				
	EP2A70	17,417,088					11			1				
Mercury (1.8 V)	EP1M120	1,303,120					1	1	1	1				
	EP1M350	4,394,032					3	1	1	1				
APEX 20KC (1.8 V)	EP20K200C	196,8016					2	1	1	1				
	EP20K400C	390,9776					3	1	1	1				
	EP20K600C	567,3936					4	1	1	1				
	EP20K1000C	8,960,016					6		1	1				
APEX 20KE (1.8 V)	EP20K30E	354,832			1	1	1	1	1	1				
	EP20K60E	648,016				1	1	1	1	1				
	EP20K100E	1,008,016				1	1	1	1	1				
	EP20K160E	1,524,016					1	1	1	1				
	EP20K200E	1,968,016					2	1	1	1				
	EP20K300E	2,741,616					2	1	1	1				
	EP20K400E	3,909,776					3	1	1	1				
	EP20K600E	5,673,936					4	1	1	1				
	EP20K1000E	8,960,016					6		1	1				
	EP20K1500E	12,042,256					8		1	1				

Table 1–2. Configuration Devices Required (Part 3 of 4)

Family	Device	Data Size (Bits) (1)	EPC1064/ 1064V	EPC1213	EPC1441	EPC1	EPC2	EPC4 (2)	EPC8 (2)	EPC16 (2)	EPCS1	EPCS4	EPCS16	EPCS64
APEX 20K (2.5 V)	EP20K100	993,360				1	1	1	1	1				
	EP20K200	1,950,800					2	1	1	1				
	EP20K400	3,880,720					3	1	1	1				
ACEX 1K (2.5 V)	EP1K10	159,160			1	1	1	1	1	1				
	EP1K30	473,720				1	1	1	1	1				
	EP1K50	224,480				1	1	1	1	1				
	EP1K100	367,392					1	1	1	1				
FLEX 10KE (2.5 V)	EPF10K30E	473,720				1	1	1	1	1				
	EPF10K50E	784,184				1	1	1	1	1				
	EPF10K50S	784,184				1	1	1	1	1				
	EPF10K100B	1,200,000					1	1	1	1				
	EPF10K100E	1,335,720					1	1	1	1				
	EPF10K130E	1,838,360					2	1	1	1				
	EPF10K200E	2,756,296					2	1	1	1				
	EPF10K200S	2,756,296					2	1	1	1				
FLEX 10KA (3.3 V)	EPF10K10A	120,000			1	1	1	1	1	1				
	EPF10K30A	406,000			1	1	1	1	1	1				
	EPF10K50V	621,000				1	1	1	1	1				
	EPF10K100A	1,200,000					1	1	1	1				
	EPF10K130V	1,600,000					1	1	1	1				
	EPF10K250A	3,300,000					2	1	1	1				

Table 1–2. Configuration Devices Required (Part 4 of 4)

Family	Device	Data Size (Bits) (1)	EPC1064/ 1064V	EPC1213	EPC1441	EPC1	EPC2	EPC4 (2)	EPC8 (2)	EPC16 (2)	EPCS1	EPCS4	EPCS16	EPCS64
FLEX 10K (5.0 V)	EPF10K10	118,000			1	1	1	1	1	1				
	EPF10K20	231,000			1	1	1	1	1	1				
	EPF10K30	376,000			1	1	1	1	1	1				
	EPF10K40	498,000				1	1	1	1	1				
	EPF10K50	621,000				1	1	1	1	1				
	EPF10K70	892,000				1	1	1	1	1				
	EPF10K100	1,200,000					1	1	1	1				
FLEX 6000/A (3.3 V)	EPF6010A	260,000			1	1								
	EPF6016 (5.0 V) / EPF6016A	260,000			1	1								
	EPF6024A	398,000			1	1								
FLEX 8000A (5.0 V)	EPF8282A / EPF8282AV (3.3 V)	40,000	1	1	1	1								
	EPF8452A	64,000	1	1	1	1								
	EPF8636A	96,000		1	1	1								
	EPF8820A	128,000		1	1	1								
	EPF81188A	192,000		1	1	1								
	EPF81500A	250,000			1	1								

Notes to Table 1–2:

(1) Raw Binary Files (**.rbf**) were used to determine these sizes.
(2) These values with the enhanced configuration device compression feature enabled.
(3) EP1S10 ES device requires four EPC2 devices.
(4) This is with the Stratix II or Cyclone series compression feature enabled.
(5) This information is preliminary.

2. Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet

CF52002-2.1

Features

- Enhanced configuration devices include EPC4, EPC8, and EPC16 devices
- Single-chip configuration solution for Stratix® series, Cyclone™ series, APEX™ II, APEX 20K (including APEX 20K, APEX 20KC, and APEX 20KE), Mercury™, ACEX® 1K, and FLEX® 10K (FLEX 10KE and FLEX 10KA) devices
- Contains 4-, 8-, and 16-Mbit flash memories for configuration data storage
  - On-chip decompression feature almost doubles the effective configuration density
- Standard flash die and a controller die combined into single stacked chip package
- External flash interface supports parallel programming of flash and external processor access to unused portions of memory
  - Flash memory block/sector protection capability via external flash interface
  - Supported in EPC16 and EPC4 devices
- Page mode support for remote and local reconfiguration with up to eight configurations for the entire system
  - Compatible with Stratix series Remote System Configuration feature
- Supports byte-wide configuration mode fast passive parallel (FPP); 8-bit data output per `DCLK` cycle
- Supports true n-bit concurrent configuration (n = 1, 2, 4, and 8) of Altera FPGAs
- Pin-selectable 2-ms or 100-ms power-on reset (POR) time
- Configuration clock supports programmable input source and frequency synthesis
  - Multiple configuration clock sources supported (internal oscillator and external clock input pin)
  - External clock source with frequencies up to 133 MHz
  - Internal oscillator defaults to 10 MHz; Programmable for higher frequencies of 33, 50, and 66 MHz
  - Clock synthesis supported via user programmable divide counter
- Available in the 100-pin plastic quad flat pack (PQFP) and the 88-pin Ultra FineLine BGA® packages
  - Vertical migration between all devices supported in the 100-pin PQFP package
- Supply voltage of 3.3 V (core and I/O)

- Hardware compliant with IEEE Std. 1532 in-system programmability (ISP) specification
- Supports ISP via Jam Standard Test and Programming Language (STAPL)
- Supports Joint Test Action Group (JTAG) boundary scan
- `nINIT_CONF` pin allows private JTAG instruction to initiate FPGA configuration
- Internal pull-up resistor on `nINIT_CONF` always enabled
- User programmable weak internal pull-up resistors on `nCS` and `OE` pins
- Internal weak pull-up resistors on external flash interface address and control lines, bus hold on data lines
- Standby mode with reduced power consumption

For more information on FPGA configuration schemes and advanced features, refer to the appropriate FPGA family chapter in the *Configuration Handbook*.

Functional Description

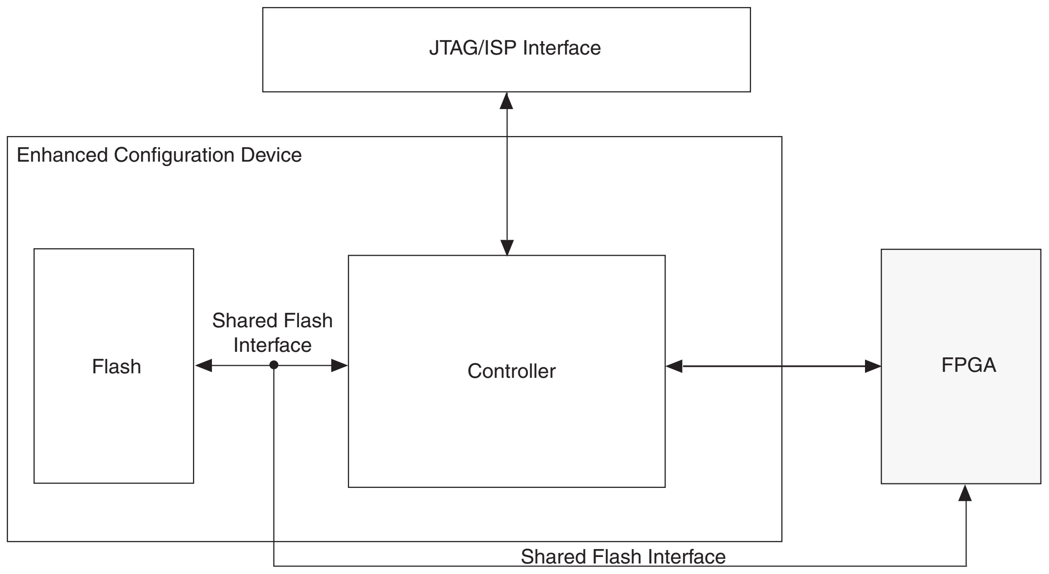
The Altera enhanced configuration device is a single-device, high-speed, advanced configuration solution for very high-density FPGAs. The core of an enhanced configuration device is divided into two major blocks, a configuration controller and a flash memory. The flash memory is used to store configuration data for systems made up of one or more Altera FPGAs. Unused portions of the flash memory can be used to store processor code or data that can be accessed via the external flash interface after FPGA configuration is complete.

The external flash interface is currently supported in the EPC16 and EPC4 devices. For information on using this feature in the EPC8 device, contact Altera Applications.

The enhanced configuration device has a 3.3-V core and I/O interface. The controller chip is a synchronous system that implements the various interfaces and features. Figure 2–1 shows a block diagram of the enhanced configuration device. The controller chip features three separate interfaces:

- A configuration interface between the controller and the Altera FPGA(s)
- A JTAG interface on the controller that enables in-system programmability (ISP) of the flash memory
- An external flash interface that the controller shares with an external processor, or FPGA implementing a Nios® embedded processor (interface available after ISP and configuration)

Figure 2–1. Enhanced Configuration Device Block Diagram



The enhanced configuration device features multiple configuration schemes. In addition to supporting the traditional passive serial (PS) configuration scheme for a single device or a serial device chain, the enhanced configuration device features concurrent configuration and parallel configuration. With the concurrent configuration scheme, up to eight PS device chains can be configured simultaneously. In the FPP configuration scheme, 8-bits of data are clocked into the FPGA each cycle. These schemes offer significantly reduced configuration times over traditional schemes.

Furthermore, the enhanced configuration device features a dynamic configuration or page mode feature. This feature allows you to dynamically reconfigure all the FPGAs in your system with new images stored in the configuration memory. Up to eight different system configurations or pages can be stored in memory and selected using the `PGM[2..0]` pins. Your system can be dynamically reconfigured by selecting one of the eight pages and initiating a reconfiguration cycle.

This page mode feature combined with the external flash interface allows remote and local updates of system configuration data. The enhanced configuration devices are compatible with the Stratix Remote System Configuration feature.

☞ For more information on Stratix Remote System Configuration, refer to the *Using Remote System Configuration with Stratix & Stratix GX Devices chapter of the Stratix Device Handbook*.

Other user programmable features include:

- Real-time decompression of configuration data
- Programmable configuration clock (`DCLK`)
- Flash ISP
- Programmable power-on-reset delay (`PORSEL`)

FPGA Configuration

FPGA configuration is managed by the configuration controller chip. This process includes reading configuration data from the flash memory, decompressing it if necessary, transmitting configuration data via the appropriate `DATA[]` pins, and handling errors conditions.

After POR, the controller determines the user-defined configuration options by reading its option bits from the flash memory. These options include the configuration scheme, configuration clock speed, decompression, and configuration page settings. The option bits are stored at flash address location 0x8000 (word address) and occupy 512-bits or 32-words of memory. These options bits are read using the internal flash interface and the default 10 MHz internal oscillator.

After obtaining the configuration settings, it checks if the FPGA is ready to accept configuration data by monitoring the `nSTATUS` and `CONF_DONE` lines. When the FPGA is ready (`nSTATUS` is high and `CONF_DONE` is low), the controller begins data transfer using the `DCLK` and `DATA[]` output pins. The controller selects the configuration page to be transmitted to the FPGA(s) by sampling its `PGM[2..0]` pins after POR or reset.

The function of the configuration unit is to transmit decompressed data to the FPGA, depending on the configuration scheme. The enhanced configuration device supports four concurrent configuration modes, with n = 1, 2, 4, or 8 (where n is the number of bits that are sent per `DCLK` cycle on the `DATA[n]` lines). The value n=1 corresponds to the traditional PS configuration scheme. The values n=2, 4, and 8 correspond to concurrent configuration of 2, 4, or 8 different PS configuration chains, respectively. Additionally, the FPGA can be configured in FPP mode, where eight bits of `DATA` are clocked into the FPGA per `DCLK` cycle. Depending on the configuration bus width (n), the circuit shifts uncompressed configuration data to the valid `DATA[n]` pins. Unused `DATA[]` pins drive low.

In addition to transmitting configuration data to the FPGAs, the configuration circuit is also responsible for pausing configuration whenever there is insufficient data available for transmission. This occurs when the flash read bandwidth is lower than the configuration write bandwidth. Configuration is paused by stopping the `DCLK` to the FPGA, when waiting for data to be read from the flash or for data to be decompressed. This technique is called "Pausing `DCLK`."

The enhanced configuration device flash memories feature a 90-ns access time (approximately 10 MHz). Hence, the flash read bandwidth is limited to about 160 megabits per second (Mbps) (16-bit flash data bus, `DQ[]`, at 10 MHz). However, the configuration speeds supported by Altera FPGAs are much higher and translate to high configuration write bandwidths. For instance, 100-MHz Stratix FPP configuration requires data at the rate of 800 Mbps (8-bit `DATA[]` bus at 100 MHz). This is much higher than the 160 Mbps the flash memory can support, and is the limiting factor for configuration time. Compression increases the effective flash read bandwidth since the same amount of configuration data takes up less space in the flash memory after compression. Since Stratix configuration data compression ratios are approximately two, the effective read bandwidth doubles to about 320 Mbps.

Finally, the configuration controller also manages errors during configuration. A `CONF_DONE` error occurs when the FPGA does not de-assert its `CONF_DONE` signal within 64 DCLK cycles after the last bit of configuration data is transmitted. When a `CONF_DONE` error is detected, the controller pulses the `OE` line low, which pulls `nSTATUS` low and triggers another configuration cycle.

A cyclic redundancy check (CRC) error occurs when the FPGA detects corruption in the configuration data. This corruption could be a result of noise coupling on the board such as poor signal integrity on the configuration signals. When this error is signaled by the FPGA (by driving the `nSTATUS` line low), the controller stops configuration. If the **Auto-Restart Configuration After Error** option is enabled in the FPGA, it releases its `nSTATUS` signal after a reset time-out period and the controller attempts to reconfigure the FPGA.

After the FPGA configuration process is complete, the controller drives `DCLK` low and the `DATA[]` pins high. Additionally, the controller tri-states its internal interface to the flash memory, enables the weak internal pull-ups on the flash address and control lines, and enables bus-keep circuits on flash data lines.

The following sections briefly describe the different configuration schemes supported by the enhanced configuration device: FPP, PS, and concurrent configuration.

For detailed information on using these schemes to configure your Altera FPGA, refer to the appropriate FPGA family chapter in the *Configuration Handbook*.

Configuration Signals

Table 2–3 lists the configuration signal connections between the enhanced configuration device and Altera FPGAs.

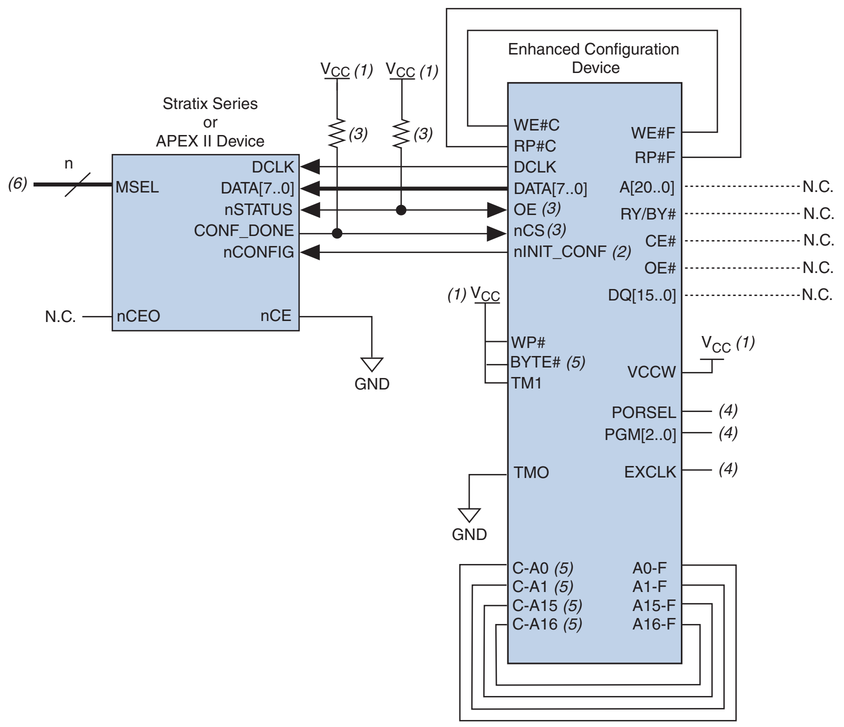
Table 2–3. Configuration Signals

Enhanced Configuration Device Pin	Altera FPGA Pin	Description
`DATA[]`	`DATA[]`	Configuration data transmitted from the configuration device to the FPGA, which is latched on the rising edge of `DCLK`.
`DCLK`	`DCLK`	Configuration device generated clock used by the FPGA to latch configuration data provided on the `DATA[]` pins.
`nINIT_CONF`	`nCONFIG`	Open-drain output from the configuration device that is used to initiate FPGA reconfiguration using the initiate configuration (`INIT_CONF`) JTAG instruction. This connection is not needed if the `INIT_CONF` JTAG instruction is not needed. If `nINIT_CONF` is not connected to `nCONFIG`, `nCONFIG` must be tied to V_{CC} either directly or through a pull-up resistor.
`OE`	`nSTATUS`	Open-drain bidirectional configuration status signal, which is driven low by either device during POR and to signal an error during configuration. Low pulse on `OE` resets the enhanced configuration device controller.
`nCS`	`CONF_DONE`	Configuration done output signal driven by the FPGA.

Fast Passive Parallel Configuration

Stratix series and APEX II devices can be configured using the enhanced configuration device in FPP mode. In this mode, the enhanced configuration device sends a byte of data on the `DATA[7..0]` pins, which connect to the `DATA[7..0]` input pins of the FPGA, per `DCLK` cycle. Stratix series and APEX II FPGAs receive byte-wide configuration data per `DCLK` cycle. Figure 2–2 shows the enhanced configuration device in FPP configuration mode. In this figure, the external flash interface is not used and hence most flash pins are left unconnected (with the few noted exceptions). For specific details on configuration interface connections including pull-up resistor values, supply voltages, and `MSEL` pin settings, refer to the appropriate FPGA family chapter in the Configuration Handbook.

Figure 2–2. FPP Configuration



Notes to Figure 2–2:

(1) The V_{CC} should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin is available on enhanced configuration devices and has an internal pull-up resistor that is always active. This means an external pull-up resistor is not required on the `nINIT_CONF`/`nCONFIG` line. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used, `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' `OE` and `nCS` pins have internal programmable pull-up resistors. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus® II software. To turn off the internal pull-up resistors, check the **Disable nCS and OE pull-ups on configuration device** option when generating programming files.

(4) For `PORSEL`, `PGM[]`, and `EXCLK` pin connections, refer to Table 2–9.

(5) In the 100-pin PQFP package, you must externally connect the following pins: `C-A0` to `F-A0`, `C-A1` to `F-A1`, `C-A15` to `F-A15`, `C-A16` to `F-A16`, and `BYTE#` to V_{CC}. Additionally, you must make the following pin connections in both 100-pin PQFP and 88-pin Ultra FineLine BGA packages: `C-RP#` to `F-RP#`, `C-WE#` to `F-WE#`, `TM1` to V_{CC}, `TM0` to GND, and `WP#` to V_{CC}.

(6) Connect the FPGA `MSEL[]` input pins to select the FPP configuration mode. For details, refer to the appropriate FPGA family chapter in the Configuration Handbook.

Multiple FPGAs can be configured using a single enhanced configuration device in FPP mode. In this mode, multiple Stratix series and/or APEX II FPGAs are cascaded together in a daisy chain.

After the first FPGA completes configuration, its nCEO pin asserts to activate the second FPGA's nCE pin, which prompts the second device to start capturing configuration data. In this setup, the FPGAs CONF_DONE pins are tied together, and hence all devices initialize and enter user mode simultaneously. If the enhanced configuration device or one of the FPGAs detects an error, configuration stops (and simultaneously restarts) for the whole chain because the nSTATUS pins are tied together.

While Altera FPGAs can be cascaded in a configuration chain, the enhanced configuration devices cannot be cascaded to configure larger devices/chains.

For configuration schematics and more information on multi-device FPP configuration, refer to the appropriate FPGA family chapter in the *Configuration Handbook*.

Passive Serial Configuration

Stratix series, Cyclone series, APEX II, APEX 20KC, APEX 20KE, APEX 20K, and FLEX 10K devices can be configured using enhanced configuration devices in the PS mode. This mode is similar to the FPP mode, with the exception that only one bit of data (DATA[0]) is transmitted to the FPGA per DCLK cycle. The remaining DATA[7..1] output pins are unused in this mode and drive low.

The configuration schematic for PS configuration of a single FPGA or single serial chain is identical to the FPP schematic (with the exception that only DATA[0] output from the enhanced configuration device connects to the FPGA DATA0 input pin; remaining DATA[7..1] pins are left floating).

For configuration schematics and more information on multi-device PS configuration, refer to the appropriate FPGA family chapter in the *Configuration Handbook*.

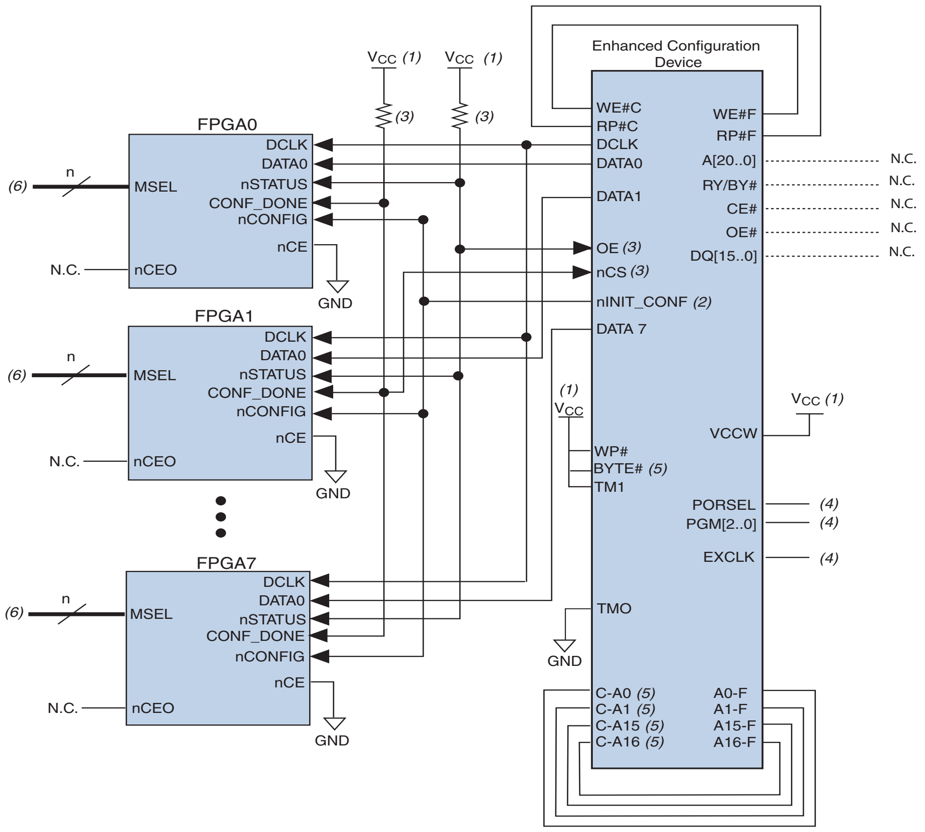
Concurrent Configuration

The enhanced configuration device supports concurrent configuration of multiple FPGAs (or FPGA chains) in PS mode. Concurrent configuration is when the enhanced configuration device simultaneously outputs n bits of configuration data on the DATA[n-1..0] pins (n = 1, 2, 4, or 8), and each DATA[] line serially configures a different FPGA (chain). The number of concurrent serial chains is user-defined via the Quartus II software and can be any number between 1 and 8. For example, three concurrent chains you can select the 4-bit PS mode, and connect the least significant DATA bits to the FPGAs or FPGA chains. Leave the most significant DATA bit (DATA[3]) unconnected. Similarly, for 5-, 6- or 7-bit concurrent chains you can select the 8-bit PS mode.

Figure 2–3 shows the schematic for configuring multiple FPGAs concurrently in the PS mode using an enhanced configuration device.

For specific details on configuration interface connections including pull-up resistor values, supply voltages, and MSEL pin settings, refer to the appropriate FPGA family chapter in the *Configuration Handbook*.

Figure 2–3. Concurrent Configuration of Multiple FPGAs in PS Mode (n = 8)



Notes to Figure 2–3:

(1) Connect V_{CC} to the same supply voltage as the configuration device.

(2) The nINIT_CONF pin is available on enhanced configuration devices and has an internal pull-up resistor that is always active. This means an external pull-up resistor is not required on the nINIT_CONF/nCONFIG line. The nINIT_CONF pin does not need to be connected if its functionality is not used. If nINIT_CONF is not used, nCONFIG must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' OE and nCS pins have internal programmable pull-up resistors. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the **Disable nCS and OE pull-ups on configuration device** option when generating programming files.

(4) For PORSEL, PGM[], and EXCLK pin connections, refer to Table 2–9.

(5) In the 100-pin PQFP package, you must externally connect the following pins: C-A0 to F-A0, C-A1 to F-A1, C-A15 to F-A15, C-A16 to F-A16, and BYTE# to V_{CC}. Additionally, you must make the following pin connections in both 100-pin PQFP and 88-pin Ultra FineLine BGA packages: C-RP# to F-RP#, C-WE# to F-WE#, TM1 to V_{CC}, TM0 to GND, and WP# to V_{CC}.

(6) Connect the FPGA MSEL[] input pins to select the PS configuration mode. For details, refer to the appropriate FPGA family chapter in the Configuration Handbook.

Table 2–4 summarizes the concurrent PS configuration modes supported in the enhanced configuration device.

Table 2–4. Enhanced Configuration Devices in PS Mode

Mode Name	Mode (n =) (1)	Used Outputs	Unused Outputs
Passive serial mode	1	`DATA0`	`DATA[7..1]` drive low
Multi-device passive serial mode	2	`DATA[1..0]`	`DATA[7..2]` drive low
Multi-device passive serial mode	4	`DATA[3..0]`	`DATA[7..4]` drive low
Multi-device passive serial mode	8	`DATA[7..0]`	-

Note to Table 2–4:
(1) This is the number of valid `DATA` outputs for each configuration mode.

For configuration schematics and more information on concurrent configuration, refer to *Using Altera Enhanced Configuration Devices*, chapter 3 in volume 2 of the *Configuration Handbook*. or the appropriate FPGA family chapter in the *Configuration Handbook*.

External Flash Interface

The enhanced configuration devices support external FPGA or processor access to its flash memory. The unused portions of the flash memory can be used by the external device to store code or data. This interface can also be used in systems that implement remote configuration capabilities. Configuration data within a particular configuration page can be updated via the external flash interface and the system could be reconfigured with the new FPGA image. This interface is also useful to store Nios boot code and/or application code.

For more information on the Stratix remote configuration feature, refer to the *Using Remote System Configuration with Stratix & Stratix GX Devices* chapter of the *Stratix Device Handbook*.

The address, data, and control ports of the flash memory are internally connected to the enhanced configuration device controller and to external device pins. An external source can drive these external device pins to access the flash memory when the flash interface is available.

This external flash interface is a shared bus interface with the configuration controller chip. The configuration controller is the primary bus master. Since there is no bus arbitration support, the external device can only access the flash interface when the controller has tri-stated its internal interface to the flash. Simultaneous access by the controller and the external device will cause contention, and result in configuration and programming failures.

Since the internal flash interface is directly connected to the external flash interface pins, controller flash access cycles will toggle the external flash interface pins. The external device must be able to tri-state its flash interface during these times and ignore transitions on the flash interface pins.

☞ The external flash interface signals cannot be shared between multiple enhanced configuration devices because this causes contention during in-system programming and configuration. During these times, the controller chips inside the enhanced configuration devices are actively accessing flash memory. Therefore, enhanced configuration devices do not support shared flash bus interfaces.

The enhanced configuration device controller chip accesses flash memory during:

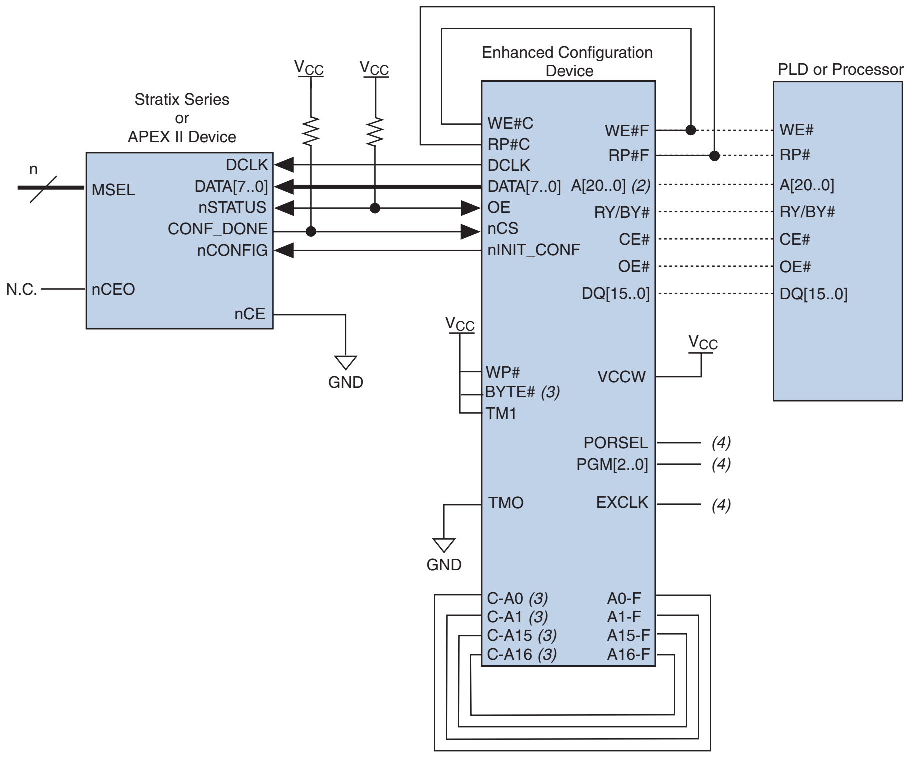
- FPGA configuration—reading configuration data from flash
- JTAG-based flash programming—storing configuration data in flash
- At POR—reading option bits from flash

During these times, the external FPGA/processor must tri-state its interface to the flash memory. After configuration and programming, the enhanced configuration device's controller tri-states the internal interface and goes into an idle mode. To interrupt a configuration cycle in order to access the flash via the external flash interface, the external device can hold the FPGA's `nCONFIG` input low. This keeps the configuration device in reset by holding the `nSTATUS-OE` line low, allowing external flash access.

For further details on the software support for the external flash interface feature, refer to *Using Altera Enhanced Configuration Devices*, chapter 3 in volume 2 of the *Configuration Handbook*. For details on flash commands, timing, memory organization, and write protection features, refer to the appropriate flash data sheet (Sharp LHF16306 for EPC16 devices and Micron MT28F400B3 for EPC4 devices) on the Altera web site at **www.altera.com**.

Figure 2–4 shows a FPP configuration schematic with the external flash interface being used.

Figure 2–4. FPP Configuration with External Flash Interface *Note (1)*



Notes to Figure 2–4:

(1) For external flash interface support in EPC8 enhanced configuration device, contact Altera Applications.
(2) Pin `A20` in EPC16 devices, pins `A20` and `A19` in EPC8 devices, and pins `A20`, `A19`, and `A18` in EPC4 devices should be left floating. These pins should not be connected to any signal, i.e., they are no-connect pins.
(3) In the 100-pin PQFP package, you must externally connect the following pins: `C-A0` to `F-A0`, `C-A1` to `F-A1`, `C-A15` to `F-A15`, `C-A16` to `F-A16`, and `BYTE #` to V_{CC}. Additionally, you must make the following pin connections in both 100-pin PQFP and 88-pin Ultra FineLine BGA packages: `C-RP#` to `F-RP#`, `C-WE#` to `F-WE#`, `TM1` to V_{CC}, `TM0` to GND, and `WP#` to V_{CC}.
(4) For `PORSEL`, `PGM[]`, and `EXCLK` pin connections, refer to Table 2–9.

Dynamic Configuration (Page Mode)

The dynamic configuration or page mode feature allows the enhanced configuration device to store up to eight different sets of designs for all the FPGAs in your system. You can then choose which page (set of configuration files) the enhanced configuration device should use for FPGA configuration.

Dynamic configuration or the page mode feature enables you to store a minimum of two pages: a factory default or fail-safe configuration, and an application configuration. The fail-safe configuration page could be programmed during system production, while the application configuration page could support remote or local updates. These remote updates could add or enhance system features and performance. However, with remote update capabilities comes the risk of possible corruption of configuration data. In the event of such a corruption, the system could automatically switch to the fail-safe configuration and avoid system downtime.

The enhanced configuration device page mode feature works with the Stratix Remote System Configuration feature, to enable intelligent remote updates to your systems.

For more information on remotely updating Stratix FPGAs, refer to *Using Remote System Configuration with Stratix & Stratix GX Devices* in the *Stratix Device Handbook*.

The three `PGM[2..0]` input pins control which page is used for configuration, and these pins are sampled at the start of each configuration cycle when `OE` goes high. The page mode selection allows you to dynamically reconfigure the functionality of your FPGA(s) by switching the `PGM[2..0]` pins and asserting `nCONFIG`. Page 0 is defined as the default page and the `PGM[2]` pin is the most significant bit (MSB).

The `PGM[2..0]` input pins must not be left floating on your board, regardless of whether this feature is used or not. When this feature is not used, connect the `PGM[2..0]` pins to GND to select the default page 000.

The enhanced configuration device pages are dynamically sized regions in memory. The start address and length of each page is programmed into the option bit space of the flash memory during initial programming. All subsequent configuration cycles will sample the `PGM[]` pins and use the option bit information to jump to the start of the corresponding configuration page. Each page must have configuration files for all FPGAs in your system that are connected to that enhanced configuration device.

For example, if your system requires three configuration pages and includes two FPGAs, each page will store two SRAM Object Files (**.sof**) for a total of six SOFs in the configuration device.

Furthermore, all enhanced configuration device configuration schemes (PS, FPP, and concurrent PS) are supported with the page mode feature. The number of pages and/or devices that can be configured using a single enhanced configuration device is only limited by the size of the flash memory.

For detailed information on the page mode feature implementation and programming file generation steps using Quartus II software, refer to *Using Altera Enhanced Configuration Devices*, chapter 3 in volume 2 of the *Configuration Handbook*.

Real-Time Decompression

Enhanced configuration devices support on-chip real time decompression of configuration data. FPGA configuration data is compressed by the Quartus II software and stored in the enhanced configuration device. During configuration, the decompression engine inside the enhanced configuration device will decompress or expand configuration data. This feature increases the effective configuration density of the enhanced configuration device up to 7, 15, or 30 Mbits in the EPC4, EPC8, and EPC16, respectively.

The enhanced configuration device also supports a parallel 8-bit data bus to the FPGA to reduce configuration time. However, in some cases, the FPGA data transfer time is limited by the flash read bandwidth. For example, when configuring an APEX II device in FPP (byte-wide data per cycle) mode at a configuration speed of 66 MHz, the FPGA write bandwidth is equal to 8 bits × 66 MHz = 528 Mbps. The flash read interface, however, is limited to approximately 10 MHz (since the flash access time is ~90 ns). This translates to a flash read bandwidth of 16 bits × 10 MHz = 160 Mbps. Hence, the configuration time is limited by the flash read time.

When configuration data is compressed, the amount of data that needs to be read out of the flash is reduced by about 50%. If 16 bits of compressed data yields 30 bits of uncompressed data, the flash read bandwidth increases to 30 bits × 10 MHz = 300 Mbps, reducing overall configuration time.

You can enable the controller's decompression feature in the Quartus II software, **Configuration Device Options** window by turning on **Compression Mode**.

☞ The decompression feature supported in the enhanced configuration devices is different from the decompression feature supported by the Stratix II FPGAs and the Cyclone series. When configuring Stratix II FPGAs or the Cyclone series using enhanced configuration devices, Altera recommends enabling decompression in Stratix II FPGAS or the Cyclone series only for faster configuration.

The compression algorithm used in Altera devices is optimized for FPGA configuration bitstreams. Since FPGAs have several layers of routing structures (for high performance and easy routability), large amounts of resources go unused. These unused routing and logic resources as well as un-initialized memory structures result in a large number of configuration RAM bits in the disabled state. Altera's proprietary compression algorithm takes advantage of such bitstream qualities.

The general guideline for effectiveness of compression is the higher the device logic/routing utilization, the lower the compression ratio (where compression ratio is defined as original bitstream size divided by the compressed bit-stream size).

For Stratix designs, based on a suite of designs with varying amounts of logic utilization, the minimum compression ratio was observed to be 1.9 or a ~47% size reduction for these designs. Table 2–5 shows sample compression ratios from a suite of Stratix designs. These numbers serve as a guideline (not a specification) to help you allocate sufficient configuration memory to store compressed bitstreams.

Table 2–5. Stratix Compression Ratios Note (1)

	Minimum	Average
Logic Utilization	98%	64%
Compression Ratio	1.9	2.3
% Size Reduction	47%	57%

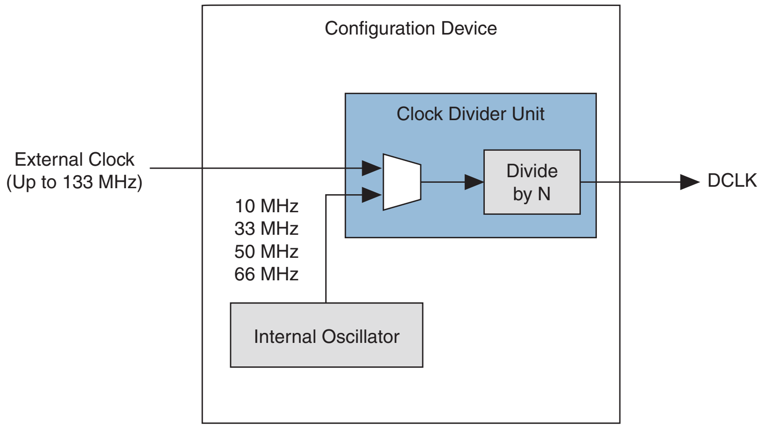
Note to Table 2–5:

(1) These numbers are preliminary. They are intended to serve as a guideline, not a specification.

Programmable Configuration Clock

The configuration clock (DCLK) speed is user programmable. One of two clock sources can be used to synthesize the configuration clock; a programmable oscillator or an external clock input pin (EXCLK). The configuration clock frequency can be further synthesized using the clock divider circuitry. This clock can be divided by the N counter to generate your DCLK output. The N divider supports all integer dividers between 1 and 16, as well as a 1.5 divider and a 2.5 divider. The duty cycle for all clock divisions other than non-integer divisions is 50% (for the non-integer dividers, the duty cycle will not be 50%). See Figure 2–5 for a block diagram of the clock divider unit.

Figure 2–5. Clock Divider Unit



The DCLK frequency is limited by the maximum DCLK frequency the FPGA supports.

The maximum DCLK input frequency supported by the FPGA is specified in the appropriate FPGA family chapter in the *Configuration Handbook*.

The controller chip features a programmable oscillator that can output four different frequencies. The various settings generate clock outputs at frequencies as high as 10, 33, 50, and 66 MHz, as shown in Table 2–6.

Table 2–6. Internal Oscillator Frequencies

Frequency Setting	Min (MHz)	Typ (MHz)	Max (MHz)
10	6.4	8.0	10.0
33	21.0	26.5	33.0
50	32.0	40.0	50.0
66	42.0	53.0	66.0

Clock source, oscillator frequency, and clock divider (N) settings can be made in the Quartus II software, by accessing the **Configuration Device Options** inside the **Device Settings** window or the **Convert Programming Files** window. The same window can be used to select between the internal oscillator and the external clock `(EXCLK)` input pin as your configuration clock source. The default setting selects the internal oscillator at the 10 MHz setting as the clock source, with a divide factor of 1.

For more information on making the configuration clock source, frequency, and divider settings, refer to *Using Altera Enhanced Configuration Devices*, chapter 3 in volume 2 of the *Configuration Handbook*.

Flash In-System Programming (ISP)

The flash memory inside enhanced configuration devices can be programmed in-system via the JTAG interface and the external flash interface. JTAG-based programming is facilitated by the configuration controller in the enhanced configuration device. External flash interface programming requires an external processor or FPGA to control the flash.

The enhanced configuration device flash memory supports 100,000 erase cycles.

JTAG-based Programming

The IEEE Std. 1149.1 JTAG Boundary Scan is implemented in enhanced configuration devices to facilitate the testing of its interconnection and functionality. Enhanced configuration devices also support the ISP mode. The enhanced configuration device is compliant with the IEEE Std. 1532 draft 2.0 specification.

The JTAG unit of the configuration controller communicates directly with the flash memory. The controller processes the ISP instructions and performs the necessary flash operations. The enhanced configuration devices support a maximum JTAG `TCK` frequency of 10 MHz.

During JTAG-based ISP, the external flash interface is not available. Before the JTAG interface programs the flash memory, an optional JTAG instruction (PENDCFG) can be used to assert the FPGA's `nCONFIG` pin (via the `nINIT_CONF` pin). This will keep the FPGA in reset and terminate any internal flash access. This function prevents contention on the flash pins when both JTAG ISP and an external FPGA/processor try to access the flash simultaneously. The `nINIT_CONF` pin is released when the Initiate Configuration (`nINIT_CONF`) JTAG instruction is updated. As a result, the FPGA is configured with the new configuration data stored in flash.

This function can be added to your programming file in the Quartus II software by enabling the **Initiate configuration after programming** option in the **Programmer options** window (Options menu).

Programming via External Flash Interface

This method allows parallel programming of the flash memory (using the 16-bit data bus). An external processor or FPGA acts as the flash controller and has access to programming data (via a communication link such as UART, Ethernet, and PCI). In addition to the program, erase, and verify operations, the external flash interface supports block/sector protection instructions.

For information on protection commands, areas, and lock bits, refer to the appropriate flash memory data sheet (Sharp LHF16506 for EPC16 devices and Micron MT28F400B3 for EPC4 devices) on the Altera web site at **www.altera.com**.

External flash interface programming is only allowed when the configuration controller has relinquished flash access (by tri-stating its internal interface). If the controller has not relinquished flash access (during configuration or JTAG-based ISP), you must hold the controller in reset before initiating external programming. The controller can be reset by holding the FPGA `nCONFIG` line at a logic low level. This keeps the controller in reset by holding the `nSTATUS-OE` line low, allowing external flash access.

If initial programming of the enhanced configuration device is done in-system via the external flash interface, the controller must be kept in reset by driving the FPGA `nCONFIG` line low to prevent contention on the flash interface.

Pin Description

Tables 2–7 through 2–9 describe the enhanced configuration device pins. These tables include configuration interface pins, external flash interface pins, JTAG interface pins, and other pins.

Table 2–7. Configuration Interface Pins

Pin Name	Pin Type	Description
DATA[7..0]	Output	This is the configuration data output bus. DATA changes on each falling edge of DCLK. DATA is latched into the FPGA on the rising edge of DCLK.
DCLK	Output	The DCLK output pin from the enhanced configuration device serves as the FPGA configuration clock. DATA is latched by the FPGA on the rising edge of DCLK.
nCS	Input	The nCS pin is an input to the enhanced configuration device and is connected to the FPGA's CONF_DONE signal for error detection after all configuration data is transmitted to the FPGA. The FPGA will always drive nCS and OE low when nCONFIG is asserted. This pin contains a programmable internal weak pull-up resistor that can be disabled/enabled in the Quartus II software through the **Disable nCS and OE pull-ups on configuration device** option.
nINIT_CONF	Open-Drain Output	The nINIT_CONF pin can be connected to the nCONFIG pin on the FPGA to initiate configuration from the enhanced configuration device via a private JTAG instruction. This pin contains an internal weak pull-up resistor that is always active. The INIT_CONF pin does not need to be connected if its functionality is not used. If nINIT_CONF is not used, nCONFIG must be pulled to V_{CC} either directly or through a pull-up resistor.
OE	Open-Drain Bidirectional	This pin is driven low when POR is not complete. A user-selectable 2-ms or 100-ms counter holds off the release of OE during initial power up to permit voltage levels to stabilize. POR time can be extended by externally holding OE low. OE is connected to the FPGA nSTATUS signal. After the enhanced configuration device controller releases OE, it waits for the nSTATUS-OE line to go high before starting the FPGA configuration process. This pin contains a programmable internal weak pull-up resistor that can be disabled/enabled in the Quartus II software through the **Disable nCS and OE pull-ups on configuration device** option.

Table 2–8. External Flash Interface Pins (Part 1 of 2)

Pin Name	Pin Type	Description
A[20..0]	Input	These pins are the address input to the flash memory for read and write operations. The addresses are internally latched during a write cycle. When the external flash interface is not used, leave these pins floating (with the few exceptions noted below). These flash address, data, and control pins are internally connected to the configuration controller. In the 100-pin PQFP package, four address pins (A0, A1, A15, A16) are not internally connected to the controller. These loop back connections must be made on the board between the C-A[] and F-A[] pins even when not using the external flash interface. All other address pins are connected internal to the package. All address pins are connected internally in the 88-pin Ultra FineLine BGA package. Pin A20 in EPC16 devices, pins A20 and A19 in EPC8 devices, and pins A20, A19, and A18 in EPC4 devices are no-connects. These pins should be left floating on the board.
DQ[15..0]	Bidirectional	This is the flash data bus interface between the flash memory and the controller. The controller or an external source drives DQ[15..0] during the flash command and the data write bus cycles. During the data read cycle, the flash memory drives the DQ[15..0] to the controller or external device. Leave these pins floating on the board when the external flash interface is not used.
CE#	Input	Active low flash input pin that activates the flash memory when asserted. When it is high, it deselects the device and reduces power consumption to standby levels. This flash input pin is internally connected to the controller. Leave this pin floating on the board when the external flash interface is not used.
RP# *(1)*	Input	Active low flash input pin that resets the flash when asserted. When high, it enables normal operation. When low, it inhibits write operation to the flash memory, which provides data protection during power transitions. This flash input is not internally connected to the controller. Hence, an external loop back connection between C-RP# and F-RP# must be made on the board even when you are not using the external flash interface. When using the external flash interface, connect the external device to the RP# pin with the loop back.

Table 2–8. External Flash Interface Pins (Part 2 of 2)

Pin Name	Pin Type	Description
`OE#`	Input	Active low flash control input that is asserted by the controller or external device during flash read cycles. When asserted, it enables the drivers of the flash output pins. Leave this pin floating on the board when the external flash interface is not used.
`WE#` *(1)*	Input	Active low flash write strobe asserted by the controller or external device during flash write cycles. When asserted, it controls writes to the flash memory. In the flash memory, addresses and data are latched on the rising edge of the `WE#` pulse. This flash input is not internally connected to the controller. Hence, an external loop back connection between `C-WE#` and `F-WE#` must be made on the board even when you are not using the external flash interface. When using the external flash interface, connect the external device to the `WE#` pin with the loop back.
`WP#`	Input	This pin is usually tied to V_{CC} or ground on the board. The controller does not drive this pin because it could cause contention. Connection to V_{CC} is recommended for faster block erase/programming times and to allow programming of the flash bottom boot block, which is required when programming the device using the Quartus II software. This pin should be connected to V_{CC} even when the external flash interface is not used.
`VCCW`	Supply	Block erase, full chip erase, word write, or lock bit configuration power supply. Connect this pin to the 3.3-V V_{CC} supply, even when you are not using the external flash interface.
`RY/BY#`	Output	Flash asserts this pin when a write or erase operation is complete. This pin is not connected to the controller. Leave this pin floating when the external flash interface is not used.
`BYTE#`	Input	This is flash byte enable pin and is only available for enhanced configuration devices in the 100-pin PQFP package. This pin must be connected to V_{CC} on the board even when you are not using the external flash interface (the controller uses the flash in 16-bit mode).

Note to Table 2–8:

(1) These pins can be driven to 12 V during production testing of the flash memory. Since the controller cannot tolerate the 12-V level, connections from the controller to these pins are not made internal to the package. Instead they are available as two separate pins. You must connect the two pins at the board level (for example, on the printed circuit board (PCB), connect the `C-WE#` pin from controller to `F-WE#` pin from the flash memory).

Table 2–9. JTAG Interface Pins and Other Required Controller Pins

Pin Name	Pin Type	Description
`TDI`	Input	This is the JTAG data input pin. Connect this pin to V_{CC} if the JTAG circuitry is not used.
`TDO`	Output	This is the JTAG data output pin. Do not connect this pin if the JTAG circuitry is not used (leave floating).
`TCK`	Input	This is the JTAG clock pin. Connect this pin to GND if the JTAG circuitry is not used.
`TMS`	Input	This is the JTAG mode select pin. Connect this pin to V_{CC} if the JTAG circuitry is not used.
`PGM[2..0]`	Input	These three input pins select one of the eight pages of configuration data to configure the FPGA(s) in the system. Connect these pins on the board to select the page specified in the Quartus II software when generating the enhanced configuration device POF. `PGM[2]` is the MSB. Default selection is page 0; `PGM[2..0]=000`. These pins must not be left floating.
`EXCLK`	Input	Optional external clock input pin that can be used to generate the configuration clock (`DCLK`). When an external clock source is not used, connect this pin to a valid logic level (high or low) to prevent a floating input buffer.
`PORSEL`	Input	This pin selects a 2-ms or 100-ms POR counter delay during power up. When `PORSEL` is low, POR time is 100-ms. When `PORSEL` is high, POR time is 2 ms. This pin must be connected to a valid logic level.
`TM0`	Input	For normal operation, this test pin must be connected to GND.
`TM1`	Input	For normal operating, this test pin must be connected to V_{CC}.

Power-On Reset (POR)

The POR circuit keeps the system in reset until power supply voltage levels have stabilized. The POR time consists of the V_{CC} ramp time and a user programmable POR delay counter. When the supply is stable and the POR counter expires, the POR circuit releases the `OE` pin. The POR time can be further extended by an external device by driving the `OE` pin low.

☞ Do not execute JTAG or ISP instructions until POR is complete.

The enhanced configuration device supports a programmable POR delay setting. You can set the POR delay to the default 100-ms setting or reduce the POR delay to 2 ms for systems that require fast power-up. The `PORSEL` input pin controls this POR delay; a logic high level selects the 2-ms delay, while a logic low level selects the 100-ms delay.

The enhanced configuration device can enter reset under the following conditions:

- The POR reset starts at initial power-up during V_{CC} ramp-up or if V_{CC} drops below the minimum operating condition anytime after V_{CC} has stabilized
- The FPGA initiates reconfiguration by driving `nSTATUS` low, which occurs if the FPGA detects a CRC error or if the FPGA's `nCONFIG` input pin is asserted
- The controller detects a configuration error and asserts `OE` to initiate re-configuration of the Altera FPGA (for example when `CONF_DONE` stays low after all configuration data has been transmitted)

Power Sequencing

Altera requires that you power-up the FPGA's V_{CCINT} supply before the enhanced configuration device's POR expires.

Power up needs to be controlled so that the enhanced configuration device's `OE` signal goes high after the `CONF_DONE` signal is pulled low. If the EEPC device exits POR before the FPGA is powered up, the `CONF_DONE` signal will be high since the pull-up resistor is holding this signal high. When the enhanced configuration device exits POR, `OE` is released and pulled high by a pull-up resistor. Since the enhanced configuration device samples the `nCS` signal on the rising edge of `OE`, it detects a high level on `CONF_DONE` and enters an idle mode. `DATA` and `DCLK` outputs will not toggle in this state and configuration will not begin. The enhanced configuration device will only exit this mode if it is powered down and then powered up correctly.

☞ To ensure the enhanced configuration device enters configuration mode properly, you need to ensure that the FPGA completes power-up before the enhanced configuration device exits POR.

The pin-selectable POR time feature is useful for ensuring this power-up sequence. The enhanced configuration device has two POR settings, 2 ms when `PORSEL` is set to a high level and 100 ms when `PORSEL` is set to a low level. For more margin, the 100-ms setting can be selected to allow the FPGA to power-up before configuration is attempted.

Alternatively, a power monitoring circuit or a power good signal can be used to keep the FPGA's nCONFIG pin asserted low until both supplies have stabilized. This ensures the correct power up sequence for successful configuration.

Programming & Configuration File Support

The Quartus II development software provides programming support for the enhanced configuration device and automatically generates the POF files for the EPC4, EPC8, and EPC16 devices. In a multi-device project, the software can combine the SOF files for multiple Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10K FPGAs into one programming file for the enhanced configuration device.

Refer to Using *Altera Enhanced Configuration Devices*, chapter 3 in volume 2 of the *Configuration Handbook* or the *Software Settings* section in the *Configuration Handbook* for details on generating programming files.

Enhanced configuration devices can be programmed in-system through its industry-standard 4-pin JTAG interface. The ISP feature in the enhanced configuration device provides ease in prototyping and updating FPGA functionality.

After programming an enhanced configuration device in-system, FPGA configuration can be initiated by including the enhanced configuration device's JTAG INIT_CONF instruction (Table 2–10).

The ISP circuitry in the enhanced configuration device is compliant with the IEEE Std. 1532 specification. The IEEE Std. 1532 is a standard that allows concurrent ISP between devices from multiple vendors.

Table 2–10. Enhanced Configuration Device JTAG Instructions (Part 1 of 2) *Note (1)*

JTAG Instruction	OPCODE	Description
SAMPLE/PRELOAD	00 0101 0101	Allows a snapshot of the state of the enhanced configuration device pins to be captured and examined during normal device operation and permits an initial data pattern output at the device pins.
EXTEST	00 0000 0000	Allows the external circuitry and board-level interconnections to be tested by forcing a test pattern at the output pins and capturing results at the input pins.
BYPASS	11 1111 1111	Places the 1-bit bypass register between the TDI and the TDO pins, which allow the BST data to pass synchronously through a selected device to adjacent devices during normal device operation.

Table 2–10. Enhanced Configuration Device JTAG Instructions (Part 2 of 2) Note (1)

JTAG Instruction	OPCODE	Description
IDCODE	00 0101 1001	Selects the device IDCODE register and places it between `TDI` and `TDO`, allowing the device IDCODE to be serially shifted out to `TDO`. The device IDCODE for all enhanced configuration devices is the same and shown below: `0100A0DDh`
USERCODE	00 0111 1001	Selects the USERCODE register and places it between `TDI` and `TDO`, allowing the USERCODE to be serially shifted out the `TDO`. The 32-bit USERCODE is a programmable user-defined pattern.
INIT_CONF	00 0110 0001	This function initiates the FPGA re-configuration process by pulsing the `nINIT_CONF` pin low, which is connected to the FPGA(s) `nCONFIG` pin(s). After this instruction is updated, the `nINIT_CONF` pin is pulsed low when the JTAG state machine enters `Run-Test/Idle` state. The `nINIT_CONF` pin is then released and nCONFIG is pulled high by the resistor after the JTAG state machine goes out of `Run-Test/Idle` state. The FPGA configuration starts after `nCONFIG` goes high. As a result, the FPGA is configured with the new configuration data stored in flash via ISP. This function can be added to your programming file (POF, JAM, JBC) in the Quartus II software by enabling the **Initiate configuration after programming** option in the **Programmer options** window (Options menu).
PENDCFG	00 0110 0101	This optional function can be used to hold the `nINIT_CONF` pin low during JTAG-based ISP of the enhanced configuration device. This feature is useful when the external flash interface is controlled by an external FPGA/processor. This function prevents contention on the flash pins when both the controller and external device try to access the flash simultaneously. Before the enhanced configuration device's controller can access the flash memory, the external FPGA/processor needs to tri-state its interface to flash.This can be ensured by resetting the FPGA using the `nINIT_CONF`, which drives the `nCONFIG` pin and keeps the external FPGA/processor in the "reset" state. The `nINIT_CONF` pin is released when the Initiate Configuration (`INIT_CONF`) JTAG instruction is issued.

Note to Table 2–10:

(1) Enhanced configuration device instruction register length is 10 and boundary scan length is 174.

For more information on the enhanced configuration device JTAG support, refer to the BSDL files provided at the Altera web site.

Enhanced configuration devices can also be programmed by third-party flash programmers or on-board processors using the external flash interface. Programming files (POF) can be converted to an Intel HEX format file (**.hexout**) using the Quartus II **Convert Programming Files** utility, for use with the programmers or processors.

You can also program the enhanced configuration devices using the Quartus II software, the Altera Programming Unit (APU), and the appropriate configuration device programming adapter. Table 2–11 shows which programming adapter to use with each enhanced configuration device.

Table 2–11. Table 10. Programming Adapters

Device	Package	Adapter
EPC16	88-pin Ultra FineLine BGA	PLMUEPC-88
	100-pin PQFP	PLMQEPC-100
EPC8	100-pin PQFP	PLMQEPC-100
EPC4	100-pin PQFP	PLMQEPC-100

IEEE Std. 1149.1 (JTAG) Boundary-Scan

The enhanced configuration device provides JTAG BST circuitry that complies with the IEEE Std. 1149.1-1990 specification. JTAG boundary-scan testing can be performed before or after configuration, but not during configuration.

Figure 2–6 shows the timing requirements for the JTAG signals.

Figure 2–6. JTAG Timing Waveforms

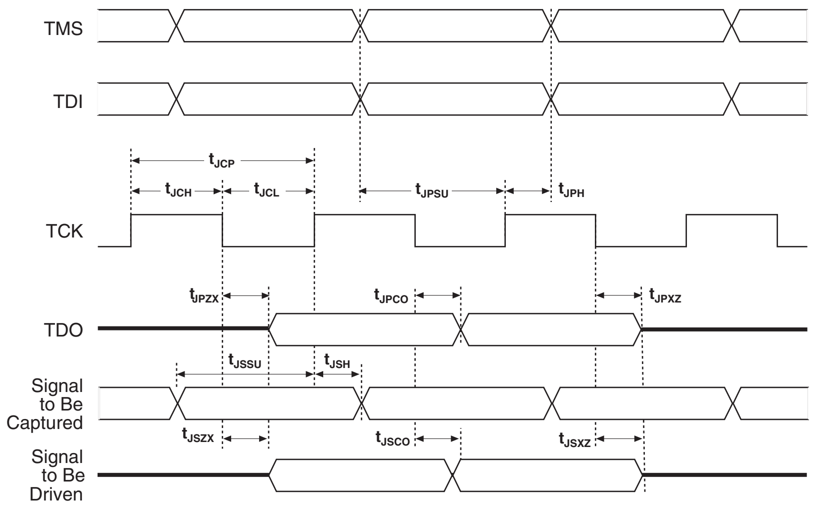


Table 2–12 shows the timing parameters and values for the enhanced configuration device.

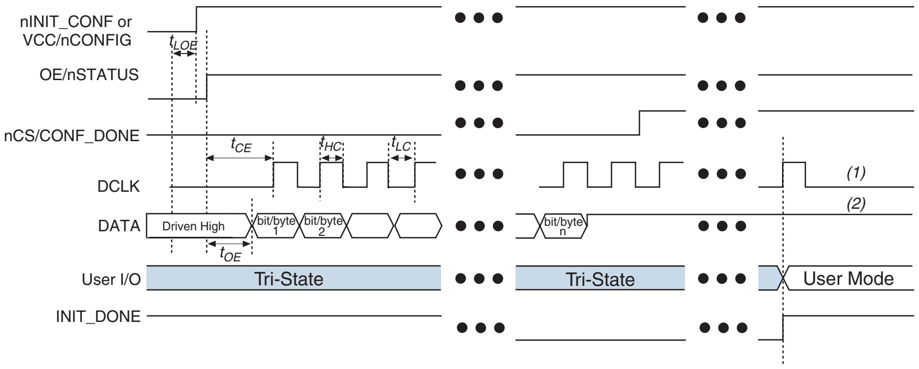
Table 2–12. JTAG Timing Parameters & Values

Symbol	Parameter	Min	Max	Unit
t_{JCP}	`TCK` clock period	100		ns
t_{JCH}	`TCK` clock high time	50		ns
t_{JCL}	`TCK` clock low time	50		ns
t_{JPSU}	JTAG port setup time	20		ns
t_{JPH}	JTAG port hold time	45		ns
t_{JPCO}	JTAG port clock output		25	ns
t_{JPZX}	JTAG port high impedance to valid output		25	ns
t_{JPXZ}	JTAG port valid output to high impedance		25	ns
t_{JSSU}	Capture register setup time	20		ns
t_{JSH}	Capture register hold time	45		ns
t_{JSCO}	Update register clock to output		25	ns
t_{JSZX}	Update register high-impedance to valid output		25	ns
t_{JSXZ}	Update register valid output to high impedance		25	ns

Timing Information

Figure 2–7 shows the configuration timing waveform when using an enhanced configuration device.

Figure 2–7. Configuration Timing Waveform Using an Enhanced Configuration Device



Notes to Figure 2–7:

(1) The enhanced configuration device will drive `DCLK` low after configuration.

(2) The enhanced configuration device will `DATA[]` high after configuration.

Table 2–13 defines the timing parameters when using the enhanced configuration devices.

For flash memory (external flash interface) timing information, please refer to the corresponding flash data sheet on the Altera web site (Sharp LHF16J06 for EPC16 devices and Micron MT28F400B3 for EPC4 devices).

Table 2–13. Enhanced Configuration Device Configuration Parameters (Part 1 of 2)

Symbol	Parameter	Condition	Min	Typ	Max	Unit
f_{DCLK}	`DCLK` frequency	40% duty cycle			66.7	MHz
t_{DCLK}	`DCLK` period		15			ns
t_{HC}	`DCLK` duty cycle high time	40% duty cycle	6			ns
t_{LC}	`DCLK` duty cycle low time	40% duty cycle	6			ns
t_{CE}	`OE` to first `DCLK` delay		40			ns
t_{OE}	`OE` to first `DATA` available		40			ns
t_{OH}	`DCLK` rising edge to `DATA` change		*(1)*			ns
t_{CF} *(2)*	`OE` assert to `DCLK` disable delay		277			ns
t_{DF} *(2)*	`OE` assert to `DATA` disable delay		277			ns

Table 2–13. Enhanced Configuration Device Configuration Parameters (Part 2 of 2)

Symbol	Parameter	Condition	Min	Typ	Max	Unit
t_{RE} *(3)*	`DCLK` rising edge to `OE`		60			ns
t_{LOE}	`OE` assert time to assure reset		60			ns
f_{ECLK}	`EXCLK` input frequency	40% duty cycle			133	MHz
t_{ECLK}	`EXCLK` input period		7.5			ns
t_{ECLKH}	`EXCLK` input duty cycle high time	40% duty cycle	3.375			ns
t_{ECLKL}	`EXCLK` input duty cycle low time	40% duty cycle	3.375			ns
t_{ECLKR}	`EXCLK` input rise time	133 MHz			3	ns
t_{ECLKF}	`EXCLK` input fall time	133 MHz			3	ns
t_{POR} *(4)*	POR time	2 ms	1	2	3	ms
		100 ms	70	100	120	ms

Notes to Table 2–13:

(1) To calculate t_{OH}, use the following equation: t_{OH} = 0.5 (`DCLK` period) - 2.5 ns.

(2) This parameter is used for CRC error detection by the FPGA.

(3) This parameter is used for `CONF_DONE` error detection by the enhanced configuration device.

(4) The FPGA V_{CCINT} ramp time should be less than 1-ms for 2-ms POR, and it should be less than 70 ms for 100-ms POR.

Operating Conditions

Tables 2–14 through 2–18 provide information on absolute maximum ratings, recommended operating conditions, DC operating conditions, supply current values, and pin capacitance data for the enhanced configuration devices.

Table 2–14. Enhanced Configuration Device Absolute Maximum Rating

Symbol	Parameter	Condition	Min	Max	Unit
V_{CC}	Supply voltage	With respect to ground	-0.5	4.6	V
V_I	DC input voltage	With respect to ground	-0.5	3.6	V
I_{MAX}	DC V_{CC} or ground current			100	mA
I_{OUT}	DC output current, per pin		-25	25	mA
P_D	Power dissipation			360	mW
T_{STG}	Storage temperature	No bias	-65	150	C
T_{AMB}	Ambient temperature	Under bias	-65	135	C
T_J	Junction temperature	Under bias		135	C

Table 2–15. Enhanced Configuration Device Recommended Operating Conditions

Symbol	Parameter	Condition	Min	Max	Unit
V_{CC}	Supplies voltage for 3.3-V operation		3.0	3.6	V
V_I	Input voltage	With respect to ground	–0.3	V_{CC} + 0.3	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	70	C
		For industrial use	–40	85	C
T_R	Input rise time			20	ns
T_F	Input fall time			20	ns

Table 2–16. Enhanced Configuration Device DC Operating Conditions

Symbol	Parameter	Condition	Min	Typ	Max	Unit
V_{CC}	Supplies voltage to core		3.0	3.3	3.6	V
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage				0.8	V
V_{OH}	3.3-V mode high-level TTL output voltage	I_{OH} = –4 mA	2.4			V
	3.3-V mode high-level CMOS output voltage	I_{OH} = –0.1 mA	V_{CC} – 0.2			V
V_{OL}	Low-level output voltage TTL	I_{OL} = –4 mA DC			0.45	V
	Low-level output voltage CMOS	I_{OL} = –0.1 mA DC			0.2	V
I_I	Input leakage current	V_I = V_{CC} or ground	–10		10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or ground	–10		10	µA
R_{CONF}	Configuration pins	Internal pull up (`OE`, `nCS`, `nINIT`, `CONF`)		6		kΩ

Table 2–17. Enhanced Configuration Device I_{CC} Supply Current Values

Symbol	Parameter	Condition	Min	Typ	Max	Unit
I_{CC0}	Current (standby)			50	100	µA
I_{CC1}	V_{CC} supply current (during configuration)			60mA	90mA	µA
I_{CCW}	V_{CCW} supply current			*(1)*	*(1)*	

Note to Table 2–17:

(1) For V_{CCW} supply current information, refer to the appropriate flash memory data sheet at **www.altera.com**.

Table 2–18. Enhanced Configuration Device Capacitance

Symbol	Parameter	Condition	Min	Max	Unit
CIN	Input pin capacitance			10	pF
COUT	Output pin capacitance			10	pF

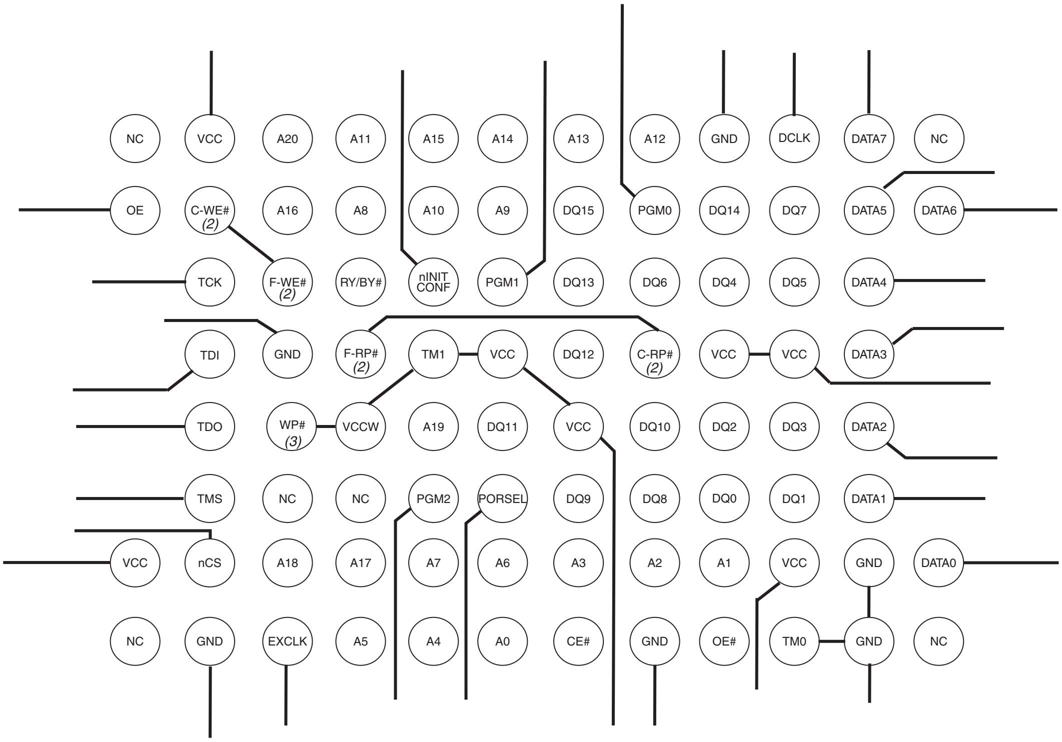
Package

The EPC16 enhanced configuration device is available in both the 88-pin Ultra FineLine BGA package and the 100-pin PQFP package. The Ultra FineLine BGA package, which is based on 0.8-mm ball pitch, maximizes board space efficiency. A board can be laid out for this package using a single PCB layer. The EPC8 and EPC4 devices are available in the 100-pin PQFP package.

Enhanced configuration devices support vertical migration in the 100-pin PQFP package.

Figure 2–8 shows the PCB routing for the 88-pin Ultra FineLine BGA package. The Gerber file for this layout is on the Altera web site.

Figure 2–8. PCB Routing for 88-Pin Ultra FineLine BGA Package Note (1)



Notes to Figure 2–8:

(1) If the external flash interface feature is not used, then the flash pins should be left unconnected since they are internally connected to controller unit. The only pins that need external connections are `WP#`, `WE#`, and `RP#`. If the flash is being used as an external memory source, then the flash pins should be connected as outlined in the pin descriptions section.

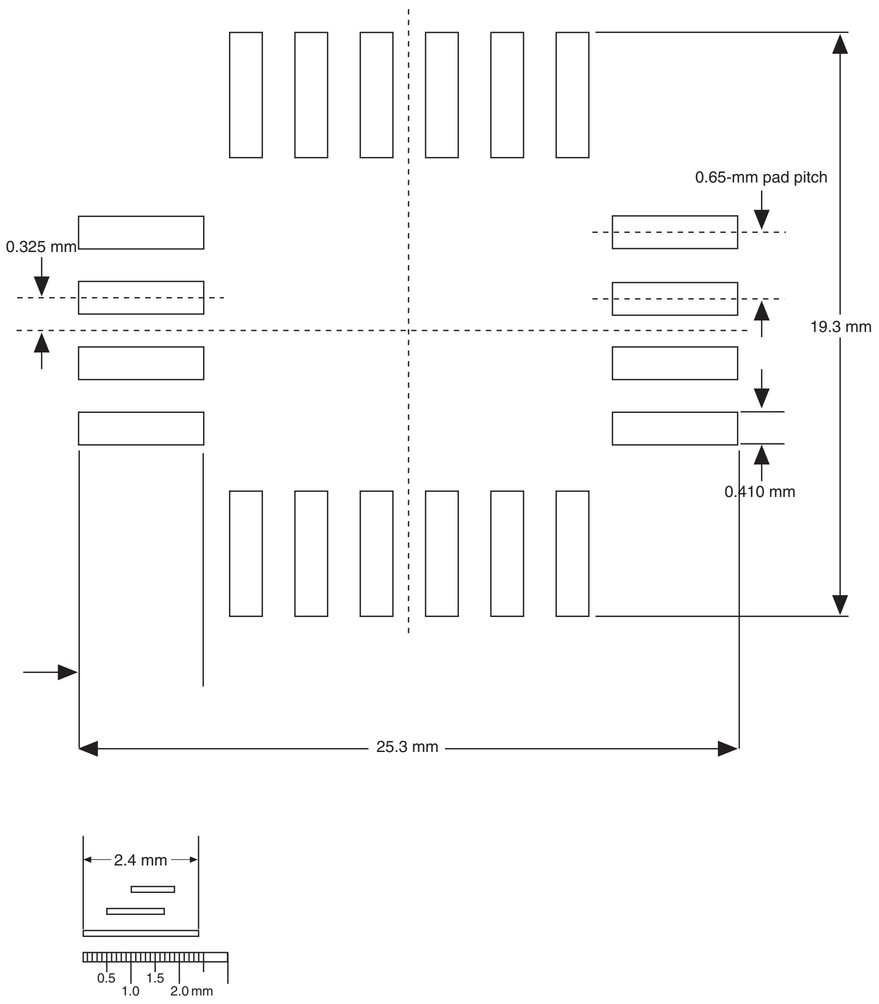
(2) `F-RP#` and `F-WE#` are pins on the flash die. `C-RP#` and `C-WE#` are pins on the controller die. `C-WE#` and `F-WE#` should be connected together on the PCB. `F-RP#` and `C-RP#` should also be connected together on the PCB.

(3) `WP#` (write protection pin) should be connected to a high level (3.3 V) to be able to program the flash bottom boot block, which is required when programming the device using the Quartus II software.

Package Layout Recommendation

EPC16 and EPC8 enhanced configuration devices in the 100-pin PQFP packages have different package dimensions than other Altera 100-pin PQFP devices (including EPC4). Figure 2–9 shows the 100-pin PQFP PCB footprint specifications for enhanced configuration devices that allows for vertical migration between all three devices. These footprint dimensions are based on vendor-supplied package outline diagrams.

Figure 2–9. Enhanced Configuration Device PCB Footprint Specifications for 100-Pin PQFP Packages *Notes (1), (2)*



***Notes to** Figure 2–9:*

(1) Used 0.5-mm increase for front and back of nominal foot length

(2) Used 0.3-mm increase to maximum foot width.

For package outline drawings, refer to the *Altera Device Package Information Data Sheet*.

Device Pin-Outs

For pin-out information, see the Altera web site at **www.altera.com**.

Ordering Codes

Table 2–19 shows the ordering codes for EPC4, EPC8, and EPC16 enhanced configuration devices.

Table 2–19. Enhanced Configuration Device Ordering Codes

Device	Package	Temperature	Ordering Code
EPC4	100-pin PQFP	Commercial	EPC4QC100
EPC4	100-pin PQFP	Industrial	EPC4QI100
EPC8	100-pin PQFP	Commercial	EPC8QC100
EPC8	100-pin PQFP	Industrial	EPC8QI100
EPC16	100-pin PQFP	Commercial	EPC16QC100
EPC16	100-pin PQFP	Industrial	EPC16QI100
EPC16	88-pin UBGA	Commercial	EPC16UC88

3. Altera Enhanced Configuration Devices

S52014-2.1

Introduction

Altera's latest enhanced configuration devices address the need for a high-density configuration solution by combining industry-standard flash memory with a feature-rich configuration controller. A single-chip configuration solution provides designers with several new and advanced features that significantly reduce configuration times. This application note discusses the hardware and software implementation of enhanced configuration device features such as concurrent and dynamic configuration, data compression, clock division, and an external flash memory interface. Enhanced configuration devices include EPC4, EPC8, and EPC16 devices.

Concurrent Configuration

Configuration data is transmitted from the enhanced configuration device to the SRAM-based device on the `DATA` lines. The `DATA` lines are outputs on the enhanced configuration devices, and inputs to the SRAM-based devices.

These `DATA` lines correspond to the `Bitn` lines in the **Convert Programming Files** window in the Altera® Quartus® II software. For example, if you specify a SRAM Object File (**.sof**) to use `Bit0` in the Quartus II software, that **.sof** will be transmitted on the `DATA[0]` line from the enhanced configuration device to the SRAM-based device.

Enhanced configuration devices can concurrently configure a number of devices with a variety of supported configuration schemes.

Supported Schemes & Guidelines

By using enhanced configuration devices, there are several different ways to configure Altera SRAM-based programmable logic devices (PLDs):

- 1-bit passive serial (PS)
- 2-bit PS
- 4-bit PS
- 8-bit PS
- Fast passive parallel (FPP)

Additionally, you can use these configuration schemes in conjunction with the dynamic configuration option (previously called page mode operation) for sophisticated configuration setups.

FPP configuration mode uses the eight `DATA [7..0]` lines from the enhanced configuration device, which can be used to configure Stratix® series, and APEX™ II devices. To decrease configuration time, FPP configuration provides eight bits of configuration data per clock cycle to the target device.

For more information on configuration schemes, refer to the *Enhanced Configuration Devices Data Sheet, Application Note 116: Configuring SRAM-Based LUT Devices*, or *Configuring Stratix & Stratix GX Devices*.

Concurrent Configuration Using *n*-Bit PS Modes

The *n*-bit (*n* = 1, 2, 4, and 8) PS configuration mode allows enhanced configuration devices to concurrently configure SRAM-based devices or device chains. In addition, these devices do not have to be the same device family or density; they can be any combination of Altera SRAM-based devices. An individual enhanced configuration device `DATA` line is available for each targeted device. Each `DATA` line can also feed a daisy chain of devices.

The Quartus II software only allows the selection of *n*-bit PS configuration modes. However, you can use these modes to configure any number of devices from 1 to 8. When configuring SRAM-based devices using *n*-bit PS modes, use Table 3–1 to select the appropriate configuration mode for the fastest configuration times.

Mode selection has an impact on the amount of memory used, as described in "Calculating the Size of Configuration Space" on page 3–17.

Table 3–1. Recommended Configuration Using n-Bit PS Modes

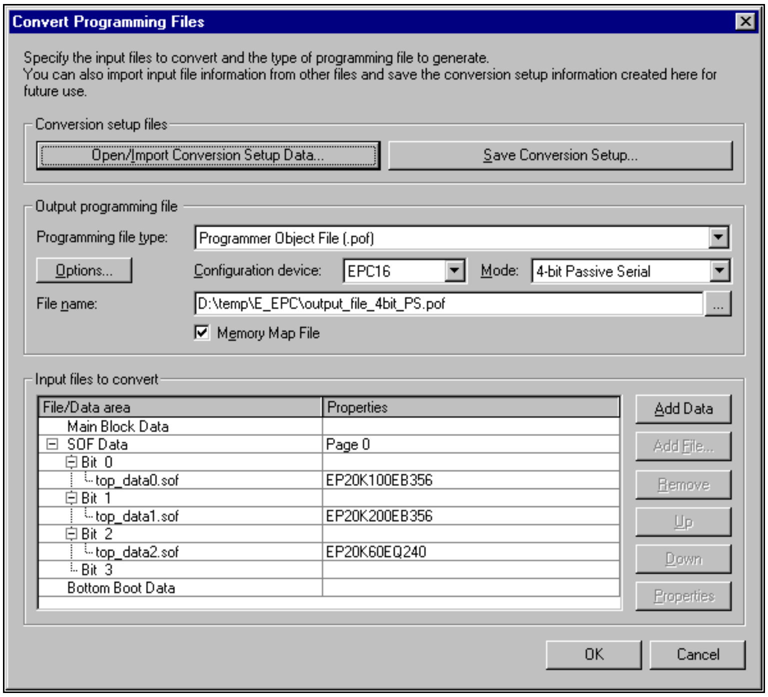
Number of Devices *(1)*	Recommended Configuration Mode
1	1-bit PS
2	2-bit PS
3	4-bit PS
4	4-bit PS
5	8-bit PS
6	8-bit PS
7	8-bit PS
8	8-bit PS

Note to Table 3–1:

(1) Assume that each `DATA` line is only configuring one device, not a daisy chain of devices.

For example, if you configure three SRAM-based devices, you would use the 4-bit PS mode. For the `DATA0`, `DATA1`, and `DATA2` lines, the corresponding **.sof** data will be transmitted from the configuration device to the SRAM-based PLD. For `DATA3`, you can leave the corresponding `Bit3` line blank in the Quartus II software. On the printed circuit board (PCB), leave the `DATA3` line from the enhanced configuration device unconnected. Figure 3–1 shows the **Quartus II Convert Programming Files** window (Tools menu) setup for this scheme.

Figure 3–1. Software Settings for Configuring Devices Using n-Bit PS Modes



Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Programming file type: Programmer Object File (.pof)

Configuration device: EPC16

Mode: 4-bit Passive Serial

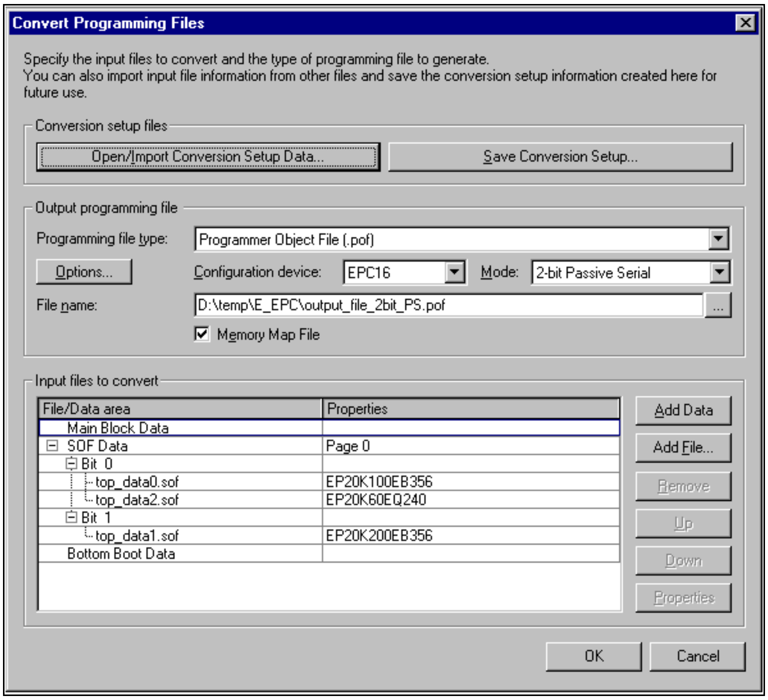
File name: D:\temp\E_EPC\output_file_4bit_PS.pof

☑ Memory Map File

File/Data area	Properties
Main Block Data	
SOF Data	Page 0
Bit 0	
top_data0.sof	EP20K100EB356
Bit 1	
top_data1.sof	EP20K200EB356
Bit 2	
top_data2.sof	EP20K60EQ240
Bit 3	
Bottom Boot Data	

Alternatively, you can daisy chain two SRAM-based devices to one `DATA` line while the other `DATA` lines drive one device each. For example, you could use the 2-bit PS mode to drive two SRAM-based devices with `DATA Bit0` (EP20K100E and EP20K60E devices) and the third device (the EP20K200E device) with `DATA Bit1`. This 2-bit PS configuration scheme requires less space in the configuration flash memory, but may increase the total system configuration time. See Figure 3–2.

Figure 3–2. Setup for Daisy Chaining Two SRAM-Based Devices to One DATA Line


Convert Programming Files

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Output programming file

Programming file type: Programmer Object File (.pof)

Configuration device: EPC16 Mode: 2-bit Passive Serial

File name: D:\temp\E_EPC\output_file_2bit_PS.pof

☑ Memory Map File

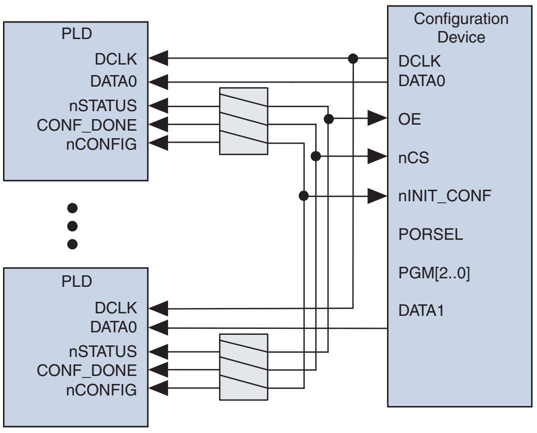
Input files to convert

File/Data area	Properties
Main Block Data	
SOF Data	Page 0
Bit 0	
top_data0.sof	EP20K100EB356
top_data2.sof	EP20K60EQ240
Bit 1	
top_data1.sof	EP20K200EB356
Bottom Boot Data	

Design Guidelines

For debugging, Altera recommends keeping the control lines such as `nSTATUS`, `nCONFIG`, and `CONF_DONE` between each PLD and the configuration device separate. You can keep control lines separate by using a switch to manage which control signals are fed back into the enhanced configuration device. Figure 3–3 shows an example of the connections between the enhanced configuration device and the targeted PLDs.

Figure 3–3. Example of Using Debugging Switches for Control Lines



Dynamic Configuration (Page Mode) Implementation Overview

Pages in enhanced configuration devices allow you to organize and store various configurations for entire systems that use one or more Altera PLDs. This dynamic configuration (or page mode) feature allows systems to dynamically reconfigure their PLDs with different configuration files.

You can use different pages to store configuration files that support different standards (e.g., I/O standards, memory). Alternatively, the different pages can place the system in different modes. For instance, page 0 could contain a configuration file (**.sof**) for the PLD that only processes data packets; page 1 could contain a configuration file for the same PLD that processes data and voice packets.

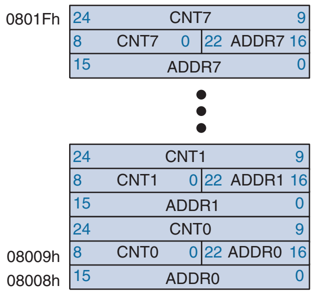
With the ability to dynamically switch pages, you can also configure Altera devices with various revisions for debugging without having to reprogram the configuration device. For example, you can configure a device that is on “stand-by” to perform another function and then reconfigure it back with the original configuration file.

A page is a section of the flash memory space that contains configuration data for all PLDs in the system. One page stores one system configuration regardless of the number of PLDs in the system. The size of each page is dynamic and can change each time the enhanced configuration device is reprogrammed. Enhanced configuration devices support a maximum of eight pages of configuration data, or eight system configurations. The number of pages is also limited to the density of the configuration device.

The number of pages required in a system is not dependent on the number of PLDs in the system, but depends on the number of unique system configurations.

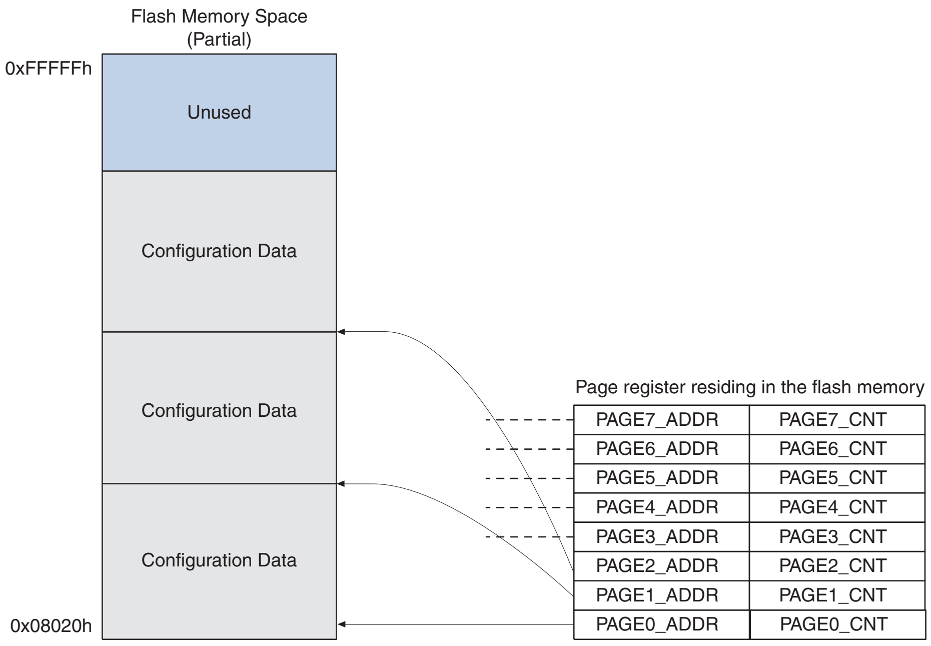
External page mode input pins `PGM[2..0]` determine which page to use during PLD configuration, and page pointers determine the data location. Each page pointer consists of a starting address register and a length count register. The word-addressable starting address register (23 bits) is used to determine where the page begins in the flash memory. The count register (25 bits) determines the length of the page counted in nibbles (group of 4 bits equaling half of a byte). Figure 3–4 shows a block diagram of the option-bit space and its address locations.

Figure 3–4. Option-Bit Memory Map



For instance, a page for the EPC16 configuration device must start between word addresses `0x08020h` and `0xFFFFFh` and cannot overlap with other pages. See Figure 3–5 for an EPC16 page mode example using three pages.

Figure 3–5. EPC16 Page Mode Implementation Example



During configuration, different pages are selected by the `PGM[2..0]` pins. These pins are used to select one out of eight pages (or eight system configurations). `PGM[2..0]` pins are sampled once before the configuration data is sent to the target PLDs.

Setting the `PGM[2..0]` pins to select an incorrect page (e.g., a page that is non-existent or a blank page) causes the enhanced configuration device to enter an erroneous state. The only way to recover from this state is to set the `PGM[2..0]` pins to select a valid page and then power cycle the board.

To ensure proper configuration, only set the `PGM[2..0]` pins to select valid pages.

Within each page, you can store as many configuration files as your system needs. There is no limitation to the length of a page except for the physical limitation determined by the size of the flash memory (e.g., `0xFFFFFFh` for EPC16 devices). However, all pages must be contiguous.

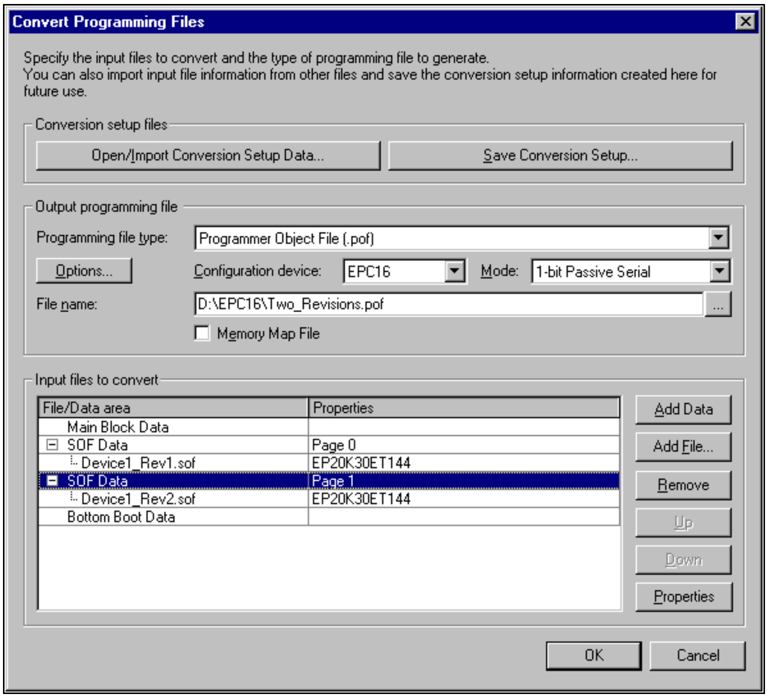
Software Implementation (Convert Programming Files)

The **Convert Programming Files** window (Tools menu) in the Quartus II software allows you to create enhanced configuration device programmer object files (**.pof**) and enable the dynamic configuration feature.

Passive parallel asynchronous (PPA) and passive parallel synchronous (PPS) configuration modes are not supported by enhanced configuration devices. If you choose one of these modes, the Quartus II software reports an error message when the enhanced configuration device's **.pof** is generated.

In the **Convert Programming Files** window, there are **SOF Data** entries (**.sof**), located in the **Input files to convert** dialog box. Each **SOF Data** entry refers to a unique system configuration. Figure 3–6 shows the setup for a system that has one APEX device and uses two pages, 0 and 1. Each of the two pages has a different version of the configuration file for the same APEX device.

Figure 3–6. Using Page Mode Example



Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Programming file type: Programmer Object File (.pof)

Configuration device: EPC16

Mode: 1-bit Passive Serial

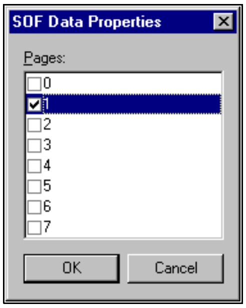
File name: D:\EPC16\Two_Revisions.pof

☐ Memory Map File

File/Data area	Properties
Main Block Data	
SOF Data	Page 0
Device1_Rev1.sof	EP20K30ET144
SOF Data	Page 1
Device1_Rev2.sof	EP20K30ET144
Bottom Boot Data	

To set which page pointer(s) will point to a particular page or **SOF Data** entry, select **SOF Data** and click **Properties**. Clicking **Properties** launches the **SOF Data Properties** window where you can select page pointers to point to the **SOF Data** chosen. If you do not use the **SOF Data Properties** window to make changes, the default page is 0. Each **SOF Data** entry for your configuration device must have a unique page number(s). Figure 3–7 shows page pointer 1 being assigned to the **SOF Data** section containing **Device1_Rev2.sof** (from Figure 3–6).

Figure 3–7. Software Setting for Selecting Pages

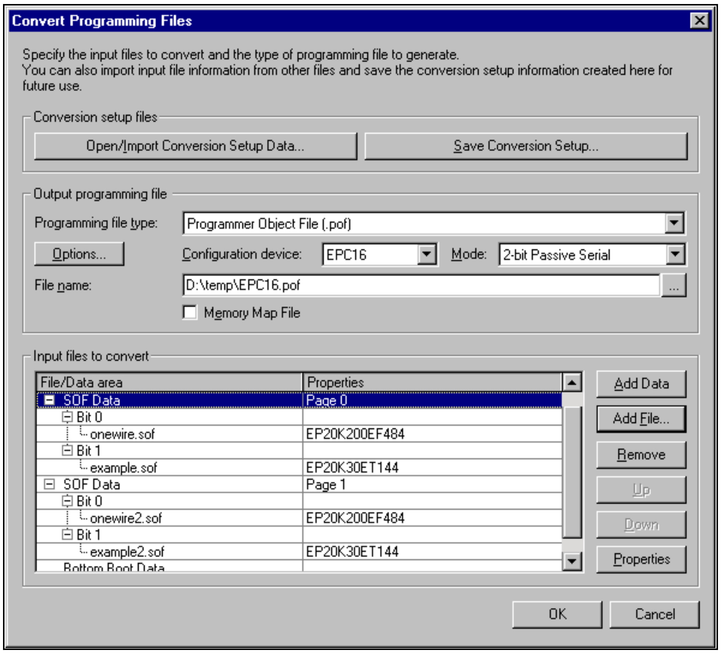

SOF Data Properties

Pages:

☐ 0
✓ 1
☐ 2
☐ 3
☐ 4
☐ 5
☐ 6
☐ 7

Figure 3–8 shows a more complex setup that uses the 2-bit PS configuration mode to concurrently configure two different APEX devices with multiple pages storing two revisions of each design. Two configurations for the entire system requires four configuration files (i.e., the number of devices multiplied by the number of unique system configurations).

Figure 3–8. Concurrent Configuration of Two Devices with Two System Configurations



Convert Programming Files

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Output programming file

Programming file type: Programmer Object File (.pof)

Configuration device: EPC16

Mode: 2-bit Passive Serial

File name: D:\temp\EPC16.pof

Memory Map File

Input files to convert

File/Data area	Properties
SOF Data	Page 0
Bit 0	
onewire.sof	EP20K200EF484
Bit 1	
example.sof	EP20K30ET144
SOF Data	Page 1
Bit 0	
onewire2.sof	EP20K200EF484
Bit 1	
example2.sof	EP20K30ET144
Bottom Boot Data	

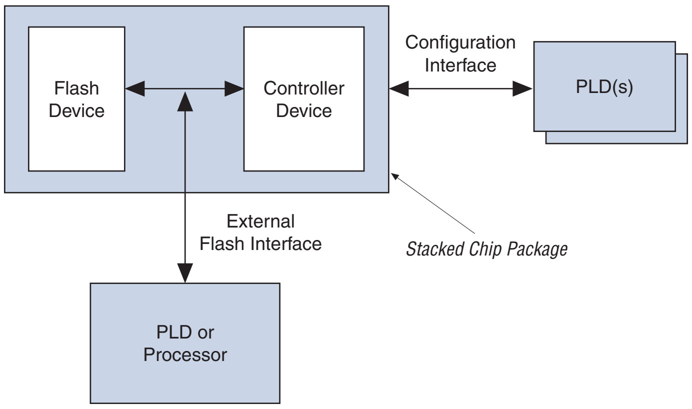
By selecting the *Memory Map File* option, the Quartus II Memory Map output file (**.map**) describing the flash memory address locations is generated. This information is typically useful when using the external flash interface feature.

External Flash Memory Interface

Enhanced configuration devices support an external flash interface that allows devices external to the controller access to the enhanced configuration device's flash memory. You can use the flash memory to store boot or application code for processors, or as general-purpose memory for processors and PLDs.

Figure 3–9 shows the interfaces available on the enhanced configuration device.

Figure 3–9. Enhanced Configuration Device Interfaces



Applications that require remote update capabilities for on-board programmable logic (Stratix series devices), and applications that use soft embedded processor cores (e.g., the Nios® embedded processor) typically use the external flash memory interface feature.

For soft core embedded processor applications, the controller configures the programmable logic by using configuration data stored in the flash memory. On successful configuration, the embedded processor uses the external flash interface to boot up and run code from the same flash memory, eliminating the need for a stand-alone flash memory device.

For applications requiring remote system configuration capabilities, a processor or PLD can use the external flash interface to store an updated configuration image into a new page in flash memory (the external flash interface coupled with dynamic configuration). You can obtain new configuration data from a local intelligent host or through the Internet. Reconfiguring the system with the new page updates the system configuration.

For more information on implementing remote and local system updates with enhanced configuration devices, refer to the *Using Remote System Configuration with Stratix & Stratix GX Devices* chapter in the *Stratix Handbook*, or the *Remote System Upgrades with Stratix II Devices* chapter in the *Stratix II Handbook*.

Currently, EPC4 and EPC16 configuration devices support the external flash interface. For support of this feature in other enhanced configuration devices, contact Altera Applications.

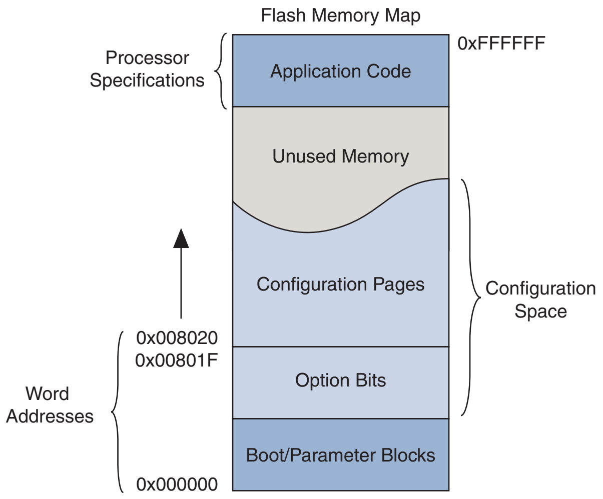
Flash Memory Map

You can divide an enhanced configuration device's flash memory into two categories: logical (configuration and processor space) and physical (flash data block boundaries). Configuration space consists of portions of memory used to store configuration option bits and configuration data. Processor space consists of portions of memory used to store boot and application code.

Logical Divisions

In all enhanced configuration devices, configuration option bits are stored ranging from word address `0x008000` to `0x00801F` (i.e., byte address `0x010000` to `0x01003F`). These bits are used to enable various controller features such as configuration mode selection, compression mode selection, and clock divider selection. In all enhanced configuration devices, configuration data is stored starting from word address location `0x008020` or byte address `0x010040`. The ending address of configuration space is not fixed and depends on the number and density of PLDs configured using the enhanced configuration device as well as the number of pages. All remaining address locations above the configuration space are available for processor application code. The boot space spans addresses `0x000000` to `0x007FFF`. Both boot and application code spaces are intended for use by an external processor or PLD. Figure 3–10 shows the flash memory map inside an EPC16 device.

Figure 3–10. EPC16 Flash Memory Map



Physical Divisions

Conversely, physical divisions are flash data blocks that can be individually written to and erased. For instance, the EPC16 device contains 16-Mbit Sharp flash memory that is divided into 2 boot blocks, 6 parameter blocks, and 31 main data blocks. These physical divisions vary from one flash memory or vendor to another and must be considered if the external flash interface is used to erase or write flash memory. These divisions are not significant if the interface is used as a read-only interface after initial programming.

For detailed information on enhanced configuration device flash memories, refer to the corresponding flash memory data sheet. The SHARP and Micron data sheets include flash command details, timing diagrams, and flash memory map information, and are available at **http://www.altera.com**.

Interface Availability & Connections

Flash memory ports are shared between the internal controller and the external device. A processor or PLD can use the external flash interface to access flash memory only when the controller is not using the interface. Therefore, the internal controller is the primary master of the bus, while the external device is the secondary master.

Flash memory ports (address, data, and control) are internally connected to the controller device. Additionally, these ports are connected to pins on the package providing the external interface. During in-system programming of the enhanced configuration device as well as configuration of the PLDs, the controller uses the internal interface to flash memory, rendering the external interface unavailable. External devices should tri-state all connections (address, data, and control) for the entire duration of in-system programming and configuration to prevent contention.

On completion of in-system programming and configuration, the internal controller tri-states its interface to the flash memory and enables weak internal pull-up resistors on address and control lines as well as bus-hold circuits on the data lines. The internal flash interface is now disabled and the external flash interface is available.

☞ If you do not use the external flash interface feature, most flash-related pins must be left unconnected on the board to avoid contention. There are a few exceptions to this guideline outlined in the data sheet and pin-out tables.

For detailed schematics, refer to the *Enhanced Configuration Device Data Sheet*.

Quartus II Software Support

You can use the **Convert Programming Files** window to generate flash memory programming files. You can program flash memory in-system using Joint Test Action Group (JTAG) or through the external flash interface. Select the **.pof** when programming the flash memory in-system. You can also convert this **.pof** to a Jam™ standard test and programming language (STAPL) file (**.jam**) or Jam Byte-Code file (**.jbc**) for in-system programming. When programming the flash memory through the external flash interface, you can create a **.hexout** from this window.

The **.hexout** used for programming enhanced configuration devices is different from the **.hexout** configuration file generated for SRAM PLDs.

Along with PLD configuration files, you can program processor boot and application code into flash memory through the **Convert Programming Files** window. You can add a **.hex** file containing boot code to the **Bottom Boot Data** section of the window. Similarly, you can add a **.hex** file containing application code to the **Main Block Data** section. You can store these files in the flash memory using relative or absolute addressing. For selecting the type of addressing, highlight the **Bottom Boot Data** or **Main Block Data** section and click **Properties** (**Convert Programming Files** window).

Relative addressing mode allows the Quartus II software to pick the location of the file in memory. For instance, the Quartus II software always stores boot code starting at address location `0x000000`. This data increases to higher addresses.

The maximum boot file size for the EPC16 configuration device is 32 K words or 64 Kbytes. The boot code is limited to the boot and flash memory parameter blocks.

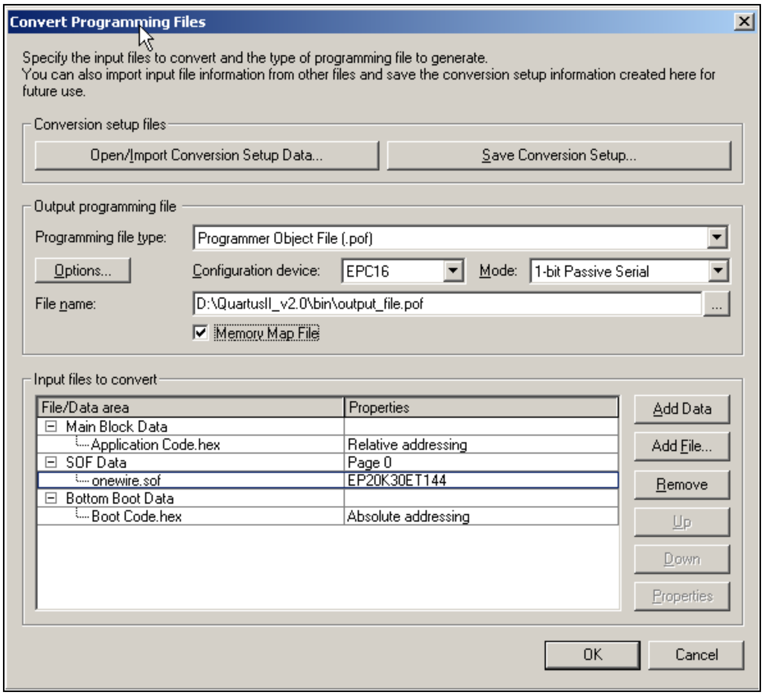
When you select relative addressing mode for **Main Block Data**, the Quartus II software aligns the last byte of information with the highest address (i.e., `0x1FFFFF`). Therefore, the starting address is dependent on the size of the **.hex** file. You can easily obtain the starting address of the application code by using the **.map** file discussed below.

Conversely, the absolute addressing mode forces the Quartus II software to store the boot or application **.hex** file data in address locations specified inside the **.hex** file itself. When this mode is selected, create **.hex** files with the correct offsets and ensure there is no overlap with addresses used for storing configuration data.

Figure 3–11 shows a screen shot of the **Convert Programming Files** window setup to create a **.pof** and **.map** file for an enhanced configuration device.

☞ Only one **.hex** file can be added to the **Bottom Boot Data** and **Main Block Data** sections of this window.

Figure 3–11. Storing Boot & Application Code in Flash Memory



Convert Programming Files

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Output programming file

Programming file type: Programmer Object File (.pof)

Configuration device: EPC16 Mode: 1-bit Passive Serial

File name: D:\QuartusII_v2.0\bin\output_file.pof

☑ Memory Map File

Input files to convert

File/Data area	Properties
⊟ Main Block Data	
Application Code.hex	Relative addressing
⊟ SOF Data	Page 0
onewire.sof	EP20K30ET144
⊟ Bottom Boot Data	
Boot Code.hex	Absolute addressing

You can use the **Quartus II Convert Programming Files** window to create two files specific to the external flash interface feature—the **.hexout** and the **.map** files. The **.hexout** contains an image of the flash memory and the **.map** file contains memory map information. The **.hexout** can be used by an external processor or PLD to program the flash memory via the external flash interface. The **.map** file contains starting and ending addresses for boot code, configuration page data, and application code.

You can use the **.hexpout** to program blank enhanced configuration devices and/or update portions of the flash memory (e.g., a new configuration page). This file uses the Intel hexadecimal file format and contains 16 Mbits or 2 Mbytes of data. The format of the **.map** file is shown in Table 3–2.

Table 3–2. File Format (.map) Note (1)

Block	Start Address	End Address
`BOTTOM BOOT`	`0x00000000`	`0x0000001F`
`OPTION BITS`	`0x00010000`	`0x0001003F`
`PAGE 0`	`0x00010040`	`0x0001AD7F`
`MAIN`	`0x001FFFE0`	`0x001FFFFF`

Note to Table 3–2:
(1) All the addresses in this file are byte addresses.

To perform partial flash memory updates, select the relevant portions of the **.hexout** using memory map information provided in the **.map** file.

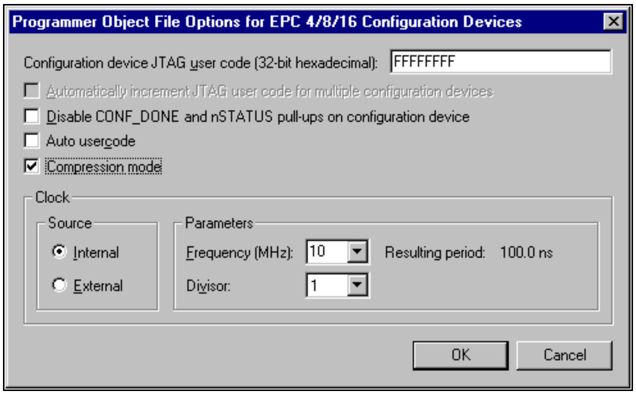
Configuration data and processor space data could exist within the same physical data block. In such cases, erasing the physical data block would affect both configuration and processor data, requiring you to update both. You can avoid this situation by storing application data starting from the next available whole data block.

Data Compression

Enhanced configuration devices support an efficient compression algorithm that compresses configuration data by 1.9× for typical designs, effectively doubling the size of the device. To select the right density for enhanced configuration devices, you should pre-calculate the total size of uncompressed configuration space.

By clicking **Options** (Convert Programming Files window), you can turn on the *Compression mode* option in the **Programming Object File Options** window with **pof** selected as the programming file type, as shown in Figure 3–12.

Figure 3–12. Selecting Compression Mode



Programmer Object File Options for EPC 4/8/16 Configuration Devices

Configuration device JTAG user code (32-bit hexadecimal): FFFFFFFF

☐ Automatically increment JTAG user code for multiple configuration devices

☐ Disable CONF_DONE and nSTATUS pull-ups on configuration device

☐ Auto usercode

☑ Compression mode

Clock

Source: [✓ Internal] External

Parameters: Frequency (MHz): 10 — Resulting period: 100.0 ns

Divisor: 1

Calculating the Size of Configuration Space

When using 1-bit PS configuration mode to serially configure multiple devices, all configuration data is transmitted through the same DATA line and the devices are daisy-chained together. Therefore, the total size of the uncompressed configuration data is equal to the sum of the SRAM-based device's configuration file size multiplied by the number of pages used.

When using *n*-bit PS configuration mode to concurrently configure multiple devices, each SRAM-based device has its own DATA line from the enhanced configuration devices. The total size of the uncompressed configuration space is equal to the size of the largest device's configuration file size multiplied by *n* (where *n* = 1, 2, 4, or 8), which is then multiplied by the number of pages used. For example, if three devices are concurrently configured using 4-bit PS configuration mode, the total size of the uncompressed configuration space is equal to the size of the largest device's configuration file multiplied by four.

When using FPP configuration mode, the total size of the uncompressed configuration space is equal to the sum of the SRAM-based device's configuration file size multiplied by the number of pages used

For configuration file sizes of SRAM-based devices, refer to *Application Note 116: Configuring SRAM-Based LUT Devices*.

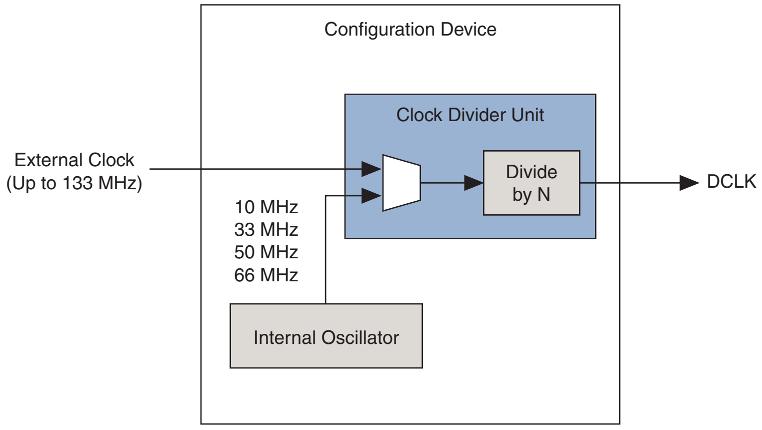
Clock Divider

The clock divider value specifies the clock frequency divisor, which is used to determine the DCLK frequency, or how fast the data is clocked into the SRAM-based device. You must consider the maximum DCLK input frequency of the targeted SRAM device family while selecting the clock input and divider settings. For DCLK timing specifications of SRAM-based devices, refer to *Application Note 116: Configuring SRAM-Based LUT Devices*.

Settings & Guidelines

Enhanced configuration devices can use either the internal oscillator or an external clock source to clock data into SRAM-based devices, as shown in Figure 3–13. The enhanced configuration device's internal oscillator runs at nominal speeds of 10, 33, 50, or 66 MHz. The minimum and maximum speeds are shown in the *Enhanced Configuration Device Data Sheet*. Additionally, the enhanced configuration device can accept an external clock source running at speeds of up to 133 MHz.

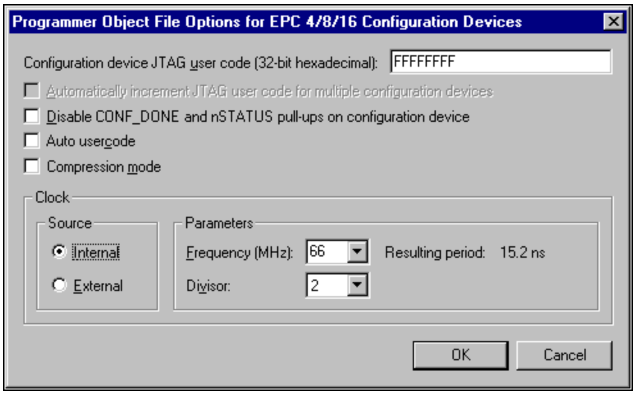
Figure 3–13. Clock Divider Unit in Enhanced Configuration Devices



Software Implementations

You can select the clock source and the clock speed in the **Programming Object File Options** window with **pof** selected as the programming file type (Convert Programming Files window), as shown in Figure 3–14. You can type the appropriate external clock frequency in the **Frequency (MHz)** drop-down menu, and select any value from the divisor list regardless of the clock source setting.

Figure 3–14. Software for Setting Clock Source & Clock Divisor


Programmer Object File Options for EPC 4/8/16 Configuration Devices
Configuration device JTAG user code (32-bit hexadecimal): FFFFFFFF
☐ Automatically increment JTAG user code for multiple configuration devices
☐ Disable CONF_DONE and nSTATUS pull-ups on configuration device
☐ Auto usercode
☐ Compression mode
Clock
Source
● Internal
○ External
Parameters
Frequency (MHz): 66
Divisor: 2
Resulting period: 15.2 ns
OK
Cancel

Conclusion

The enhanced configuration device is a single-chip configuration solution that provides designers with increased configuration flexibility and faster time-to-market. Features such as data compression, multiple clock sources, clock division, and parallel or concurrent programming significantly reduce configuration times, while the dynamic configuration mode and the external flash interface take intelligent system configuration to a higher level.

4. Serial Configuration Devices (EPCS1, EPCS4, EPCS16, & EPCS64) Features

C51014-1.5

Introduction

The serial configuration devices provide the following features:

- 1-, 4-, 16-, and 64-Mbit flash memory devices that serially configure Stratix® II FPGAs and the Cyclone™ series FPGAs using the active serial (AS) configuration scheme
- Easy-to-use four-pin interface
- Low cost, low pin count and non-volatile memory
- Low current during configuration and near-zero standby mode current
- 3.3-V operation
- Available in 8-pin and 16-pin small outline integrated circuit (SOIC) package
- Enables the Nios® processor to access unused flash memory through AS memory interface
- Re-programmable memory with more than 100,000 erase/program cycles
- Write protection support for memory sectors using status register bits
- In-system programming support with SRunner software driver
- In-system programming support with USB Blaster™, EthernetBlaster™, or ByteBlaster™ II download cables
- Additional programming support with the Altera® Programming Unit (APU) and programming hardware from BP Microsystems, System General, and other vendors
- Software design support with the Altera Quartus® II development system for Windows-based PCs as well as Sun SPARC station and HP 9000 Series 700/800
- Delivered with the memory array erased (all the bits set to `1`)

☞ Whenever the term "serial configuration device(s)" is used in this document, it refers to Altera EPCS1, EPCS4, EPCS16, and EPCS64 devices.

Functional Description

With SRAM-based devices such as Stratix II FPGAs and the Cyclone series FPGAs, configuration data must be reloaded each time the device powers up, the system initializes, or when new configuration data is needed. Serial configuration devices are flash memory devices with a serial interface that can store configuration data for a Stratix II FPGA or a Cyclone series device and reload the data to the device upon power-up or reconfiguration. Table 4–1 lists the serial configuration devices.

Table 4–1. Serial Configuration Devices (3.3-V Operation)

Device	Memory Size (Bits)
EPCS1	1,048,576
EPCS4	4,194,304
EPCS16	16,777,216
EPCS64	67,108,864

You can vertically migrate from the EPCS1 to the EPCS4 device since they are offered in the same device package. Similarly, you can vertically migrate from the EPCS16 to the EPCS64 device.

Table 4–2 lists the serial configuration device used with each Stratix II FPGA and the configuration file size. Stratix II devices can be used with EPCS16 or EPCS64 devices.

Table 4–2. Serial Configuration Device Support for Stratix II Devices

Stratix II Device	Raw Binary File Size (Bits) *(1)*	Serial Configuration Device		
		EPCS4	EPCS16	EPCS64
EP2S15	4,721,544	✓ *(2)*	✓	✓
EP2S30	9,640,672		✓	✓
EP2S60	16,951,824		✓ *(2)*	✓
EP2S90	25,699,104		✓ *(2)*	✓
EP2S130	37,325,760			✓
EP2S180	49,814,760			✓

Notes to Table 4–2:
(1) These are uncompressed file sizes.
(2) This is with the Stratix II compression feature enabled.

Table 4–3 lists the serial configuration device used with each Cyclone II FPGA and the configuration file size. Cyclone II devices can be used with all serial configuration devices.

Table 4–3. Serial Configuration Device for Cyclone II Devices

Cyclone II Device	Raw Binary File Size (Bits) *(1)*	Serial Configuration Device			
		EPCS1	EPCS4	EPCS16	EPCS64
EP2C5	1,265,792	✓ *(2)*	✓	✓	✓
EP2C8	1,983,536		✓	✓	✓
EP2C20	3,892,496		✓	✓	✓
EP2C35	6,848,608			✓	✓
EP2C50	9,951,104			✓	✓
EP2C70	14,319,216			✓	✓

Notes to Table 4–3:
(1) These are uncompressed file sizes.
(2) This is with the Cyclone II compression feature enabled.

Table 4–4 lists the serial configuration device used with each Cyclone FPGA and the configuration file size. Cyclone devices can only be used with EPCS1, EPCS4, or EPCS16 configuration devices.

Table 4–4. Serial Configuration Device Support for Cyclone Devices

Cyclone Device	Raw Binary File Size (Bits) *(1)*	Serial Configuration Device			
		EPCS1	EPCS4	EPCS16	EPCS64
EP1C3	627,376	✓	✓	✓	✓
EP1C4	924,512	✓	✓	✓	✓
EP1C6	1,167,216	✓ *(2)*	✓	✓	✓
EP1C12	2,323,240		✓	✓	✓
EP1C20	3,559,608		✓	✓	✓

Notes to Table 4–4:
(1) These are uncompressed file sizes.
(2) This is with the Cyclone compression feature enabled.

With the new data-decompression feature in the Stratix II and Cyclone FPGA families, designers can use smaller serial configuration devices to configure larger FPGAs.

Serial configuration devices cannot be cascaded.

Refer to *Configuring Stratix II Devices*, chapter 2 in volume 1 of the *Configuration Handbook* for more information regarding the Stratix II FPGA decompression feature.

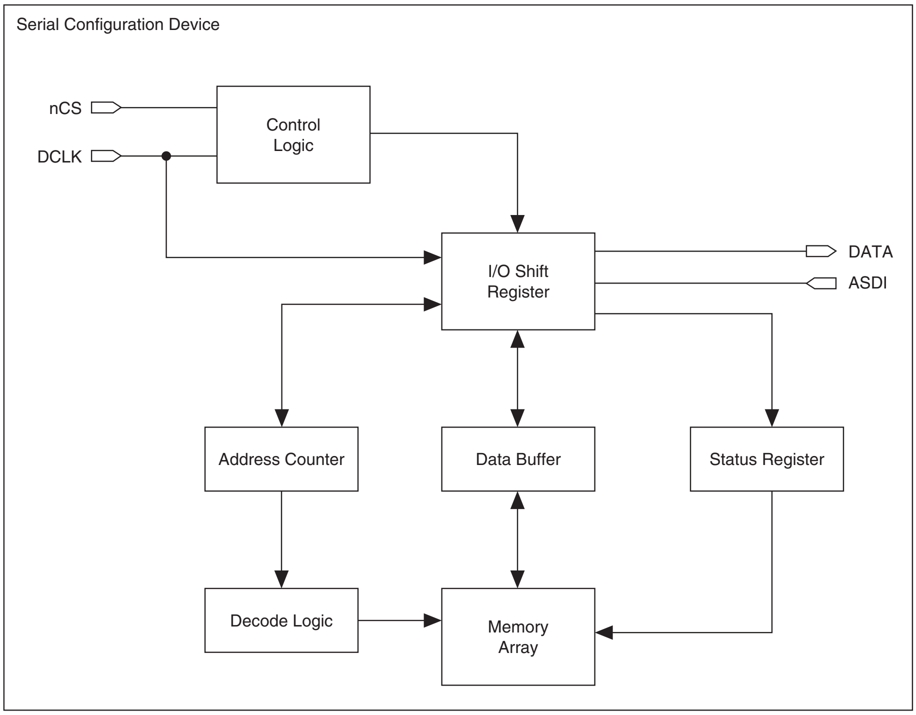
Refer to *Configuring Cyclone II Devices*, chapter 4 in volume 1 of the *Configuration Handbook* for more information regarding the Cyclone II FPGA decompression feature.

Refer to *Configuring Cyclone FPGAs*, chapter 5, in volume 1 of the *Configuration Handbook* for more information regarding the Cyclone FPGA decompression feature.

The serial configuration devices are designed to configure Stratix II FPGAs and the Cyclone series FPGAs and cannot configure other existing Altera FPGA device families.

Figure 4–1 shows the serial configuration device block diagram.

Figure 4–1. Serial Configuration Device Block Diagram



Accessing Memory in Serial Configuration Devices

You can access the unused memory locations of the serial configuration device to store or retrieve data through the Nios processor and SOPC Builder. SOPC Builder is an Altera tool for creating bus-based (especially microprocessor-based) systems in Altera devices. SOPC Builder assembles library components like processors and memories into custom microprocessor systems.

SOPC Builder includes the active serial memory interface (ASMI) peripheral, an interface core specifically designed to work with the serial configuration device. Using this core, you can create a system with a Nios embedded processor that allows software access to any memory location within the serial configuration device.

For more information on accessing memory within the serial configuration device, refer to the *Active Serial Memory Interface Data Sheet*.

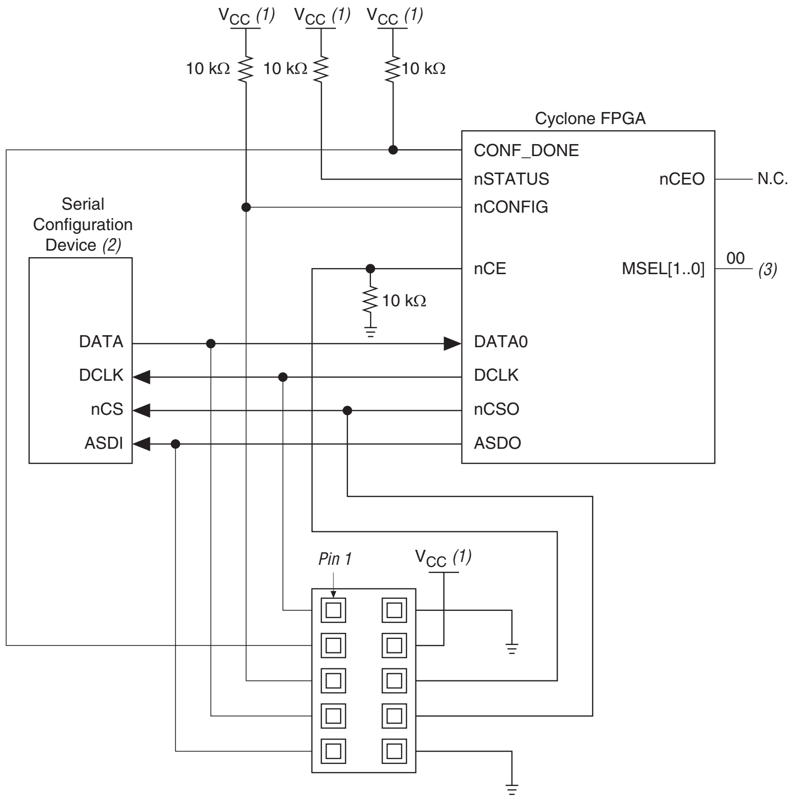
Active Serial FPGA Configuration

Stratix II FPGAs and the Cyclone series FPGAs can be configured with a serial configuration device through the AS configuration mode.

☞ This section is only relevant for FPGAs that support the Active Serial (AS) configuration scheme. Only Stratix II FPGAs and the Cyclone series FPGAs support the AS configuration scheme.

There are four signals on the serial configuration device that interface directly with the FPGA's control signals. The serial configuration device signals `DATA`, `DCLK`, `ASDI`, and `nCS` interface with `DATA0`, `DCLK`, `ASDO`, and `nCSO` control signals on the FPGA, respectively. Figure 4–2 shows a serial configuration device programmed via a download cable which configures an FPGA in AS mode. Figure 4–3 shows a serial configuration device programmed using the APU or a third-party programmer configuring an FPGA in AS configuration mode.

Figure 4–2. FPGA Configuration in AS Mode (Serial Configuration Device Programmed Using Download Cable)



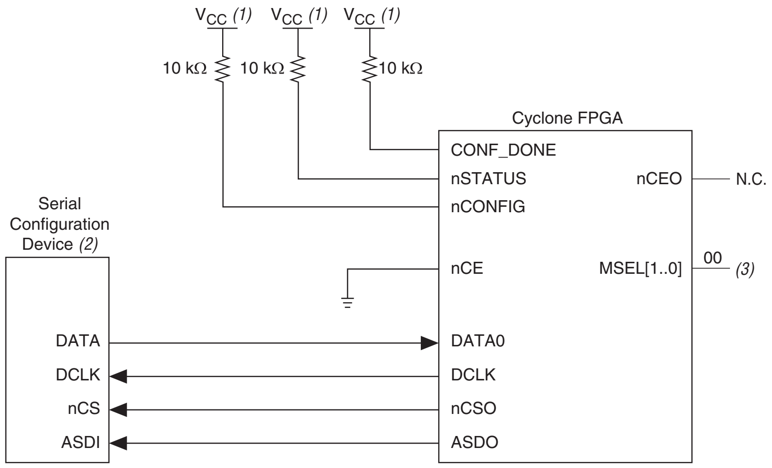
Notes to Figure 4–2:

(1) V_{CC} = 3.3 V.

(2) Serial configuration devices cannot be cascaded.

(3) Connect the FPGA `MSEL[]` input pins to select the AS configuration mode. For details, refer to the appropriate FPGA family chapter in the *Configuration Handbook*.

Figure 4–3. FPGA Configuration in AS Mode (Serial Configuration Device Programmed by APU or Third-Party Programmer)



Notes to Figure 4–3:
(1) V_{CC} = 3.3 V.
(2) Serial configuration devices cannot be cascaded.
(3) Connect the FPGA `MSEL[]` input pins to select the AS configuration mode. For details, refer to the appropriate FPGA family chapter in the *Configuration Handbook*.

The FPGA acts as the configuration master in the configuration flow and provides the clock to the serial configuration device. The FPGA enables the serial configuration device by pulling the `nCS` signal low via the `nCSO` signal (See Figures 4–2 and 4–3). Subsequently, the FPGA sends the instructions and addresses to the serial configuration device via the `ASDO` signal. The serial configuration device responds to the instructions by sending the configuration data to the FPGA's `DATA0` pin on the falling edge of `DCLK`. The data is latched into the FPGA on the `DCLK` signal's falling edge.

The FPGA controls the `nSTATUS` and `CONF_DONE` pins during configuration in AS mode. If the `CONF_DONE` signal does not go high at the end of configuration or if the signal goes high too early, the FPGA will pulse its `nSTATUS` pin low to start reconfiguration. Upon successful configuration, the FPGA releases the `CONF_DONE` pin, allowing the external 10-kΩ resistor to pull this signal high. Initialization begins after the `CONF_DONE` goes high. After initialization, the FPGA enters user mode.

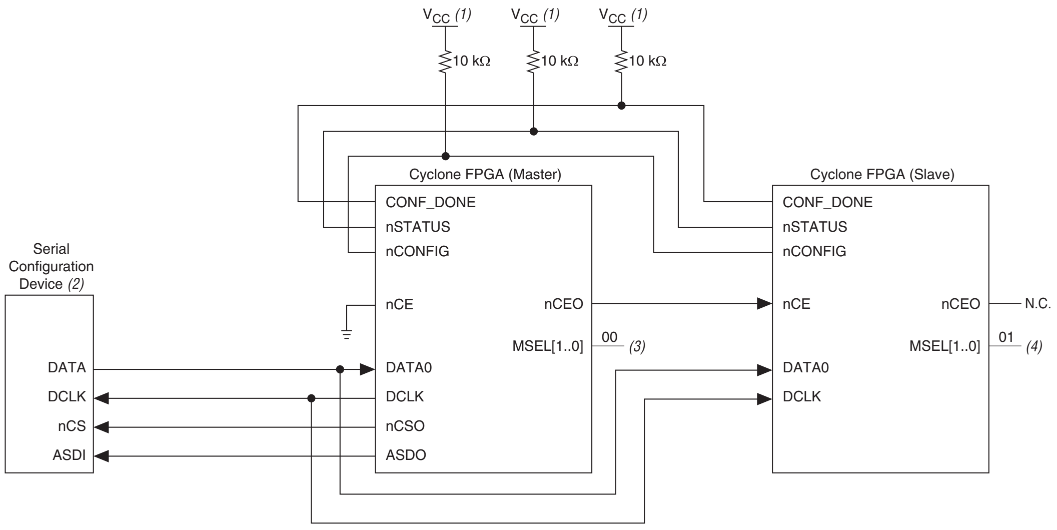
For more information on configuring Stratix II FPGAs in AS mode or other configuration modes, refer to *Configuring Stratix II Devices*, chapter 2 in volume 1 of the *Configuration Handbook*.

For more information on configuring Cyclone II FPGAs in AS mode or other configuration modes, refer to *Configuring Cyclone II Devices*, chapter 13 in volume 1 of the *Configuration Handbook*.

For more information on configuring Cyclone FPGAs in AS mode or other configuration modes, refer to *Configuring Cyclone FPGAs*, chapter 5 in volume 1 of the *Configuration Handbook*.

Multiple devices can be configured by a single EPCS device. However, serial configuration devices cannot be cascaded. Check Table 4–1 to ensure the programming file size of the cascaded FPGAs does not exceed the capacity of a serial configuration device. Figure 4–4 shows the AS configuration scheme with multiple FPGAs in the chain. The first Stratix II or Cyclone device is the configuration master and has its `MSEL[]` pins set to AS mode. The following FPGAs are configuration slave devices and have their `MSEL[]` pins set to PS mode.

Figure 4–4. Multiple Devices in AS Mode



Notes to Figure 4–4:

(1) V_{CC} = 3.3 V.

(2) Serial configuration devices cannot be cascaded.

(3) Connect the FPGA `MSEL[]` input pins to select the AS configuration mode. For details, refer to the appropriate FPGA family chapter in the *Configuration Handbook*.

(4) Connect the FPGA `MSEL[]` input pins to select the PS configuration mode. For details, refer to the appropriate FPGA family chapter in the *Configuration Handbook*

Serial Configuration Device Memory Access

This section describes the serial configuration device's memory array organization and operation codes. Timing specifications for the memory are provided in the "Timing Information" on page 4–30 section.

Memory Array Organization

Table 4–5 provides details on the memory array organization in EPCS64, EPCS16, EPCS4, and EPCS1 devices.

Table 4–5. Memory Array Organization in Serial Configuration Devices

Details	EPCS64	EPCS16	EPCS4	EPCS1
Bytes (bits)	8,388,608 bytes (64 Mbits)	2,097,152 bytes (16 Mbits)	524,288 bytes (4 Mbits)	131, 072 bytes (1 Mbit)
Number of sectors	128	32	8	4
Bytes (bits) per sector	65,536 bytes (512 Kbits)	65,536 bytes (512 Kbits)	65,536 bytes (512 Kbits)	32,768 bytes (256 Kbits)
Pages per sector	256	256	256	128
Total number of pages	32,768	8,192	2,048	512
Bytes per page	256 bytes	256 bytes	256 bytes	256 bytes

Tables 4–6, 4–7, 4–8, and 4–9 show the address range for each sector in the EPCS64, EPCS16, EPCS4, and EPCS1 devices, respectively.

Table 4–6. Address Range for Sectors in EPCS64 Devices (Part 1 of 5)

Sector	Address Range (Byte Addresses in HEX)	
	Start	End
127	`H'7F0000`	`H'7FFFFF`
126	`H'7E0000`	`H'7EFFFF`
125	`H'7D0000`	`H'7DFFFF`
124	`H'7C0000`	`H'7CFFFF`
123	`H'7B0000`	`H'7BFFFF`
122	`H'7A0000`	`H'7AFFFF`
121	`H'790000`	`H'79FFFF`
120	`H'780000`	`H'78FFFF`
119	`H'770000`	`H'77FFFF`
118	`H'760000`	`H'76FFFF`
117	`H'750000`	`H'75FFFF`

Table 4–6. Address Range for Sectors in EPCS64 Devices (Part 2 of 5)

Sector	Address Range (Byte Addresses in HEX)	
	Start	End
116	H'740000	H'74FFFF
115	H'730000	H'73FFFF
114	H'720000	H'72FFFF
113	H'710000	H'71FFFF
112	H'700000	H'70FFFF
111	H'6F0000	H'6FFFFF
110	H'6E0000	H'6EFFFF
109	H'6D0000	H'6DFFFF
108	H'6C0000	H'6CFFFF
107	H'6B0000	H'6BFFFF
106	H'6A0000	H'6AFFFF
105	H'690000	H'69FFFF
104	H'680000	H'68FFFF
103	H'670000	H'67FFFF
102	H'660000	H'66FFFF
101	H'650000	H'65FFFF
100	H'640000	H'64FFFF
99	H'630000	H'63FFFF
98	H'620000	H'62FFFF
97	H'610000	H'61FFFF
96	H'600000	H'60FFFF
95	H'5F0000	H'5FFFFF
94	H'5E0000	H'5EFFFF
93	H'5D0000	H'5DFFFF
92	H'5C0000	H'5CFFFF
91	H'5B0000	H'5BFFFF
90	H'5A0000	H'5AFFFF
89	H'590000	H'59FFFF
88	H'580000	H'58FFFF
87	H'570000	H'57FFFF
86	H'560000	H'56FFFF
85	H'550000	H'55FFFF

Table 4–6. Address Range for Sectors in EPCS64 Devices (Part 3 of 5)

Sector	Address Range (Byte Addresses in HEX)	
	Start	End
84	`H'540000`	`H'54FFFF`
83	`H'530000`	`H'53FFFF`
82	`H'520000`	`H'52FFFF`
81	`H'510000`	`H'51FFFF`
80	`H'500000`	`H'50FFFF`
79	`H'4F0000`	`H'4FFFFF`
78	`H'4E0000`	`H'4EFFFF`
77	`H'4D0000`	`H'4DFFFF`
76	`H'4C0000`	`H'4CFFFF`
75	`H'4B0000`	`H'4BFFFF`
74	`H'4A0000`	`H'4AFFFF`
73	`H'490000`	`H'49FFFF`
72	`H'480000`	`H'48FFFF`
71	`H'470000`	`H'47FFFF`
70	`H'460000`	`H'46FFFF`
69	`H'450000`	`H'45FFFF`
68	`H'440000`	`H'44FFFF`
67	`H'430000`	`H'43FFFF`
66	`H'420000`	`H'42FFFF`
65	`H'410000`	`H'41FFFF`
64	`H'400000`	`H'40FFFF`
63	`H'3F0000`	`H'3FFFFF`
62	`H'3E0000`	`H'3EFFFF`
61	`H'3D0000`	`H'3DFFFF`
60	`H'3C0000`	`H'3CFFFF`
59	`H'3B0000`	`H'3BFFFF`
58	`H'3A0000`	`H'3AFFFF`
57	`H'390000`	`H'39FFFF`
56	`H'380000`	`H'38FFFF`
55	`H'370000`	`H'37FFFF`
54	`H'360000`	`H'36FFFF`
53	`H'350000`	`H'35FFFF`

Table 4–6. Address Range for Sectors in EPCS64 Devices (Part 4 of 5)

Sector	Address Range (Byte Addresses in HEX)	
	Start	End
52	`H'340000`	`H'34FFFF`
51	`H'330000`	`H'33FFFF`
50	`H'320000`	`H'32FFFF`
49	`H'310000`	`H'31FFFF`
48	`H'300000`	`H'30FFFF`
47	`H'2F0000`	`H'2FFFFF`
46	`H'2E0000`	`H'2EFFFF`
45	`H'2D0000`	`H'2DFFFF`
44	`H'2C0000`	`H'2CFFFF`
43	`H'2B0000`	`H'2BFFFF`
42	`H'2A0000`	`H'2AFFFF`
41	`H'290000`	`H'29FFFF`
40	`H'280000`	`H'28FFFF`
39	`H'270000`	`H'27FFFF`
38	`H'260000`	`H'26FFFF`
37	`H'250000`	`H'25FFFF`
36	`H'240000`	`H'24FFFF`
35	`H'230000`	`H'23FFFF`
34	`H'220000`	`H'22FFFF`
33	`H'210000`	`H'21FFFF`
32	`H'200000`	`H'20FFFF`
31	`H'1F0000`	`H'1FFFFF`
30	`H'1E0000`	`H'1EFFFF`
29	`H'1D0000`	`H'1DFFFF`
28	`H'1C0000`	`H'1CFFFF`
27	`H'1B0000`	`H'1BFFFF`
26	`H'1A0000`	`H'1AFFFF`
25	`H'190000`	`H'19FFFF`
24	`H'180000`	`H'18FFFF`
23	`H'170000`	`H'17FFFF`
22	`H'160000`	`H'16FFFF`
21	`H'150000`	`H'15FFFF`

Table 4–6. Address Range for Sectors in EPCS64 Devices (Part 5 of 5)

Sector	Address Range (Byte Addresses in HEX)	
	Start	End
20	`H'140000`	`H'14FFFF`
19	`H'130000`	`H'13FFFF`
18	`H'120000`	`H'12FFFF`
17	`H'110000`	`H'11FFFF`
16	`H'100000`	`H'10FFFF`
15	`H'0F0000`	`H'0FFFFF`
14	`H'0E0000`	`H'0EFFFF`
13	`H'0D0000`	`H'0DFFFF`
12	`H'0C0000`	`H'0CFFFF`
11	`H'0B0000`	`H'0BFFFF`
10	`H'0A0000`	`H'0AFFFF`
9	`H'090000`	`H'09FFFF`
8	`H'080000`	`H'08FFFF`
7	`H'070000`	`H'07FFFF`
6	`H'060000`	`H'06FFFF`
5	`H'050000`	`H'05FFFF`
4	`H'040000`	`H'04FFFF`
3	`H'030000`	`H'03FFFF`
2	`H'020000`	`H'02FFFF`
1	`H'010000`	`H'01FFFF`
0	`H'000000`	`H'00FFFF`

Table 4–7. Address Range for Sectors in EPCS16 Devices

Sector	Address Range (Byte Addresses in HEX)	
	Start	End
31	`H'1F0000`	`H'1FFFFF`
30	`H'1E0000`	`H'1EFFFF`
29	`H'1D0000`	`H'1DFFFF`
28	`H'1C0000`	`H'1CFFFF`
27	`H'1B0000`	`H'1BFFFF`
26	`H'1A0000`	`H'1AFFFF`
25	`H'190000`	`H'19FFFF`
24	`H'180000`	`H'18FFFF`
23	`H'170000`	`H'17FFFF`
22	`H'160000`	`H'16FFFF`
21	`H'150000`	`H'15FFFF`
20	`H'140000`	`H'14FFFF`
19	`H'130000`	`H'13FFFF`
18	`H'120000`	`H'12FFFF`
17	`H'110000`	`H'11FFFF`
16	`H'100000`	`H'10FFFF`
15	`H'0F0000`	`H'0FFFFF`
14	`H'0E0000`	`H'0EFFFF`
13	`H'0D0000`	`H'0DFFFF`
12	`H'0C0000`	`H'0CFFFF`
11	`H'0B0000`	`H'0BFFFF`
10	`H'0A0000`	`H'0AFFFF`
9	`H'090000`	`H'09FFFF`
8	`H'080000`	`H'08FFFF`
7	`H'070000`	`H'07FFFF`
6	`H'060000`	`H'06FFFF`
5	`H'050000`	`H'05FFFF`
4	`H'040000`	`H'04FFFF`
3	`H'030000`	`H'03FFFF`
2	`H'020000`	`H'02FFFF`
1	`H'010000`	`H'01FFFF`
0	`H'000000`	`H'00FFFF`

Table 4–8. Address Range for Sectors in EPCS4 Devices

Sector	Address Range (Byte Addresses in HEX)	
	Start	End
7	H'70000	H'7FFFF
6	H'60000	H'6FFFF
5	H'50000	H'5FFFF
4	H'40000	H'4FFFF
3	H'30000	H'3FFFF
2	H'20000	H'2FFFF
1	H'10000	H'1FFFF
0	H'00000	H'0FFFF

Table 4–9. Address Range for Sectors in EPCS1 Devices

Sector	Address Range (Byte Addresses in HEX)	
	Start	End
3	H'18000	H'1FFFF
2	H'10000	H'17FFF
1	H'08000	H'0FFFF
0	H'00000	H'07FFF

Operation Codes

This section describes the operations that can be used to access the memory in serial configuration devices. The `DATA`, `DCLK`, `ASDI`, and `nCS` signals access to the memory in serial configuration devices. All serial configuration device operation codes, addresses and data are shifted in and out of the device serially, with the most significant bit (MSB) first.

The device samples the active serial data input on the first rising edge of the `DCLK` after the active low chip select (`nCS`) input signal is driven low. Shift the operation code (MSB first) serially into the serial configuration device through the active serial data input pin. Each operation code bit is latched into the serial configuration device on the rising edge of the `DCLK`.

Different operations require a different sequence of inputs. While executing an operation, you must shift in the desired operation code, followed by the address bytes, data bytes, both, or neither. The device must drive `nCS` high after the last bit of the operation sequence is shifted in. Table 4–10 shows the operation sequence for every operation supported by the serial configuration devices.

For the read byte, read status, and read silicon ID operations, the shifted-in operation sequence is followed by data shifted out on the `DATA` pin. You can drive the `nCS` pin high after any bit of the data-out sequence is shifted out.

For the write byte, erase bulk, erase sector, write enable, write disable, and write status operations, drive the `nCS` pin high exactly at a byte boundary (drive the `nCS` pin high a multiple of eight clock pulses after the `nCS` pin was driven low). Otherwise, the operation is rejected and will not be executed.

All attempts to access the memory contents while a write or erase cycle is in progress will not be granted, and the write or erase cycle will continue unaffected.

Table 4–10. Operation Codes for Serial Configuration Devices

Operation	Operation Code *(1)*	Address Bytes	Dummy Bytes	Data Bytes	DCLK f_{MAX} (MHz)
Write enable	`0000 0110`	0	0	0	25
Write disable	`0000 0100`	0	0	0	25
Read status	`0000 0101`	0	0	1 to infinite *(2)*	25
Read bytes	`0000 0011`	3	0	1 to infinite *(2)*	20
Read silicon ID	`1010 1011`	0	3	1 to infinite *(2)*	25
Write status	`0000 0001`	0	0	1	25
Write bytes	`0000 0010`	3	0	1 to 256 *(3)*	25
Erase bulk	`1100 0111`	0	0	0	25
Erase sector	`1101 1000`	3	0	0	25

Notes to Table 4–10:

(1) The MSB is listed first and the least significant bit (LSB) is listed last.

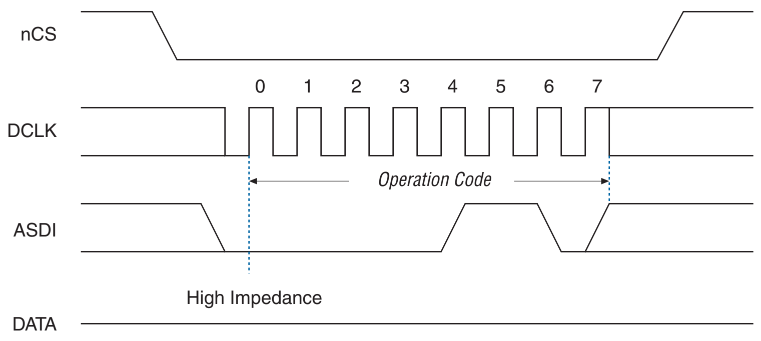
(2) The status register, data or silicon ID are read out at least once on the `DATA` pin and will continuously be read out until `nCS` is driven high

(3) Write bytes operation requires at least one data byte on the `DATA` pin. If more than 256 bytes are sent to the device, only the last 256 bytes are written to the memory.

Write Enable Operation

The write enable operation code is `b'0000 0110`, and the most significant bit is listed first. The write enable operation sets the write enable latch bit, which is bit 1 in the status register. Always set the write enable latch bit before write bytes, write status, erase bulk, and erase sector operations. Figure 4–5 shows the timing diagram for the write enable operation. Figures 4–7 and 4–8 show the status register bit definitions.

Figure 4–5. Write Enable Operation Timing Diagram



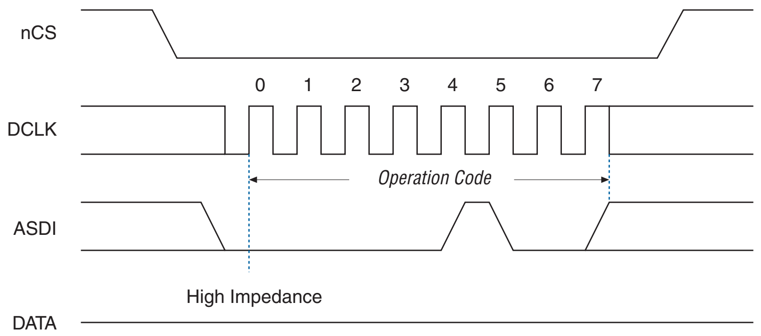
Write Disable Operation

The write disable operation code is `b'0000 0100`, with the MSB listed first. The write disable operation resets the write enable latch bit, which is bit 1 in the status register. To prevent the memory from being written unintentionally, the write enable latch bit is automatically reset when implementing the write disable operation as well as under the following conditions:

- Power up
- Write bytes operation completion
- Write status operation completion
- Erase bulk operation completion
- Erase sector operation completion

Figure 4–6 shows the timing diagram for the write disable operation.

Figure 4–6. Write Disable Operation Timing Diagram



Read Status Operation

The read status operation code is `b'0000 0101`, with the MSB listed first. You can use the read status operation to read the status register. Figures 4–7 and 4–8 show the status bits in the status register of both serial configuration devices.

Figure 4–7. EPCS4, EPCS16, and EPCS64 Status Register Status Bits

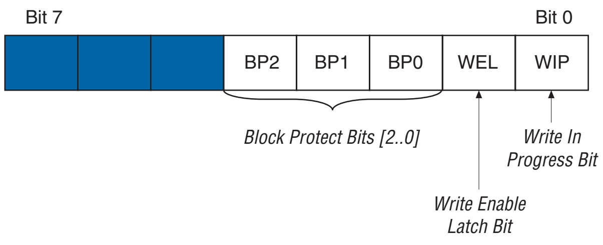
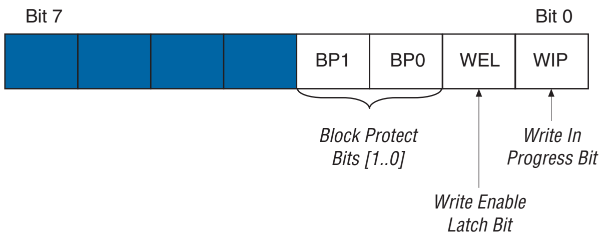


Figure 4–8. EPCS1 Status Register Status Bits



Setting the write in progress bit to 1 indicates that the serial configuration device is busy with a write or erase cycle. Resetting the write in progress bit to 0 means no write or erase cycle is in progress.

Resetting the write enable latch bit to `0` indicates that no write or erase cycle will be accepted. Set the write enable latch bit to `1` before every write bytes, write status, erase bulk, and erase sector operation.

The non-volatile block protect bits determine the area of the memory protected from being written or erased unintentionally. Tables 4–12 and 4–11 show the protected area in both serial configuration devices with reference to the block protect bits. The erase bulk operation is only available when all the block protect bits are 0. When any of the block protect bits are set to one, the relevant area is protected from being written by write bytes operations or erased by erase sector operations.

Table 4–11. Block Protection Bits in EPCS1

Status Register Content		Memory Content	
BP1 Bit	**BP0 Bit**	**Protected Area**	**Unprotected Area**
0	0	None	All four sectors: 0 to 3
0	1	Sector 3	Three sectors: 0 to 2
1	0	Two sectors: 2 and 3	Two sectors: 0 and 1
1	1	All sectors	None

Table 4–12. Block Protection Bits in EPCS4 Devices

Status Register Content			Memory Content	
BP2 Bit	**BP1 Bit**	**BP0 Bit**	**Protected Area**	**Unprotected Area**
0	0	0	None	All eight sectors: 0 to 7
0	0	1	Sector 7	Seven sectors: 0 to 6
0	1	0	Sectors 6 and 7	Six sectors: 0 to 5
0	1	1	Four sectors: 4 to 7	Four sectors: 0 to 3
1	0	0	All sectors	None
1	0	1	All sectors	None
1	1	0	All sectors	None
1	1	1	All sectors	None

Table 4–13. Block Protection Bits in EPCS16 (Part 1 of 2)

Status Register Content			Memory Content	
BP2 Bit	**BP1 Bit**	**BP0 Bit**	**Protected Area**	**Unprotected Area**
0	0	0	None	All sectors (32 sectors 0 to 31)
0	0	1	Upper 32nd (Sector 31)	Lower 31/32nds (31 sectors: 0 to 30)
0	1	0	Upper sixteenth (two sectors: 30 and 31)	Lower 15/16ths (30 sectors: 0 to 29)
0	1	1	Upper eighth (four sectors: 28 to 31)	Lower seven-eighths (28 sectors: 0 to 27)
1	0	0	Upper quarter (eight sectors: 24 to 31)	Lower three-quarters (24 sectors: 0 to 23)

Table 4–13. Block Protection Bits in EPCS16 (Continued) (Part 2 of 2)

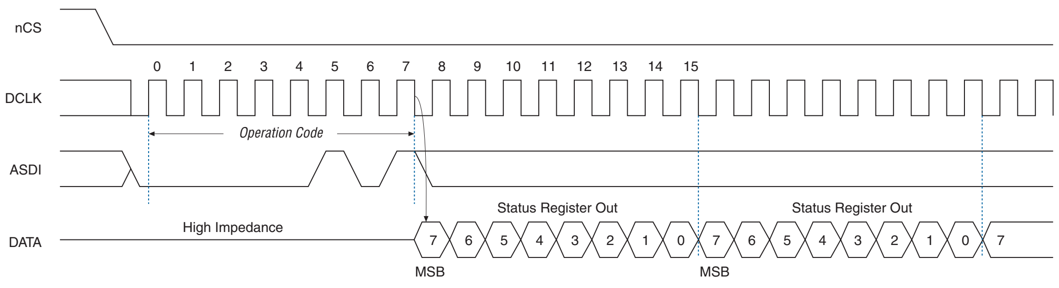
Status Register Content			Memory Content	
BP2 Bit	BP1 Bit	BP0 Bit	Protected Area	Unprotected Area
1	0	1	Upper half (sixteen sectors: 16 to 31)	Lower half (16 sectors: 0 to 15)
1	1	0	All sectors (32 sectors: 0 to 31)	None
1	1	1	All sectors (32 sectors: 0 to 31)	None

Table 4–14. Block Protection Bits in EPCS64

Status Register Content			Memory Content	
BP2 Bit	BP1 Bit	BP0 Bit	Protected Area	Unprotected Area
0	0	0	None	All sectors (128 sectors: 0 to 127)
0	0	1	Upper 64th (2 sectors: 126 and 127)	Lower 63/64ths (126 sectors: 0 to 125)
0	1	0	Upper 32nd (4 sectors: 124 to 127)	Lower 31/32nds (124 sectors: 0 to 123)
0	1	1	Upper sixteenth (8 sectors: 120 to 127)	Lower 15/16ths (120 sectors: 0 to 119)
1	0	0	Upper eighth (16 sectors: 112 to 127)	Lower seven-eighths (112 sectors: 0 to 111)
1	0	1	Upper quarter (32 sectors: 96 to 127)	Lower three-quarters (96 sectors: 0 to 95)
1	1	0	Upper half (64 sectors: 64 to 127)	Lower half (64 sectors: 0 to 63)
1	1	1	All sectors (128 sectors: 0 to 127)	None

The status register can be read at any time, even while a write or erase cycle is in progress. When one of these cycles is in progress, you can check the write in progress bit (bit `0` of the status register) before sending a new operation to the device. The device can also read the status register continuously, as shown in Figure 4–9.

Figure 4–9. Read Status Operation Timing Diagram



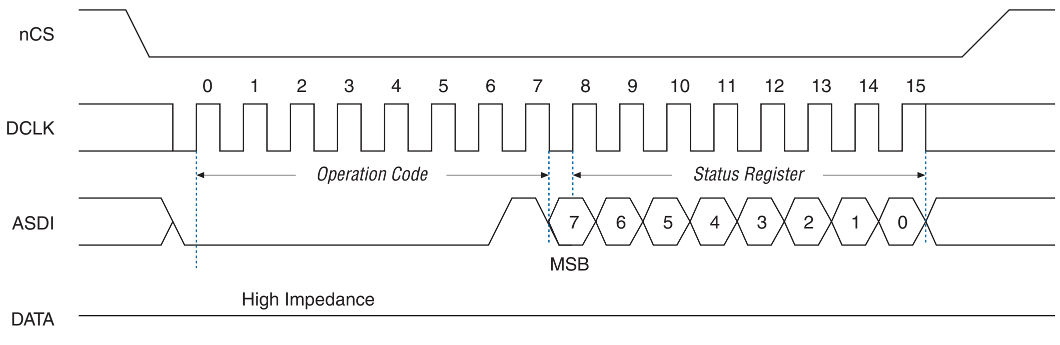
Write Status Operation

The write status operation code is `b'0000 0001`, with the MSB listed first. Use the write status operation to set the status register block protection bits. The write status operation has no effect on the other bits. Therefore, designers can implement this operation to protect certain memory sectors, as defined in Tables 4–12 and 4–11. After setting the block protect bits, the protected memory sectors are treated as read-only memory. Designers must execute the write enable operation before the write status operation so the device sets the status register's write enable latch bit to 1.

The write status operation is implemented by driving `nCS` low, followed by shifting in the write status operation code and one data byte for the status register on the `ASDI` pin. Figure 4–10 shows the timing diagram for the write status operation. `nCS` must be driven high after the eighth bit of the data byte has been latched in, otherwise, the write status operation is not executed.

Immediately after `nCS` is driven high, the device initiates the self-timed write status cycle. The self-timed write status cycle usually takes 5 ms for all serial configuration devices and is guaranteed to be less than 15 ms (see t_{WS} in Table 4–16). Designers must account for this delay to ensure that the status register is written with desired block protect bits. Alternatively, you can check the write in progress bit in the status register by executing the read status operation while the self-timed write status cycle is in progress. The write in progress bit is 1 during the self-timed write status cycle, and is 0 when it is complete.

Figure 4–10. Write Status Operation Timing Diagram

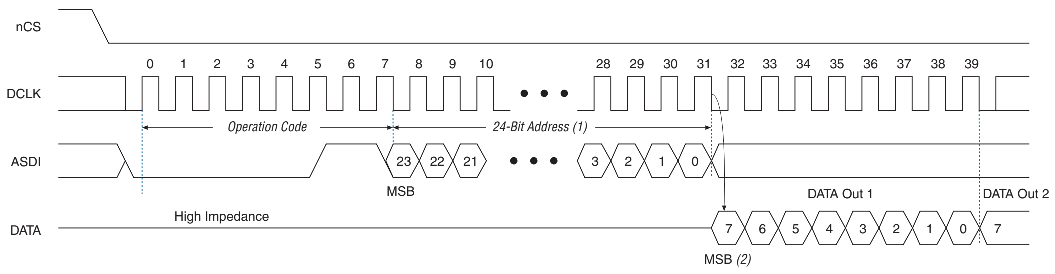


Read Bytes Operation

The read bytes operation code is `b'0000 0011`, with the MSB listed first. To read the memory contents of the serial configuration device, the device is first selected by driving `nCS` low. Then, the read bytes operation code is shifted in followed by a 3-byte address (`A[23..0]`). Each address bit must be latched in on the rising edge of the `DCLK`. After the address is latched in, the memory contents of the specified address are shifted out serially on the `DATA` pin, beginning with the MSB. For reading Raw Programming Data files (**.rpd**), the content is shifted out serially beginning with the LSB. Each data bit is shifted out on the falling edge of `DCLK`. The maximum `DCLK` frequency during the read bytes operation is 20 MHz. Figure 4–11 shows the timing diagram for read bytes operation.

The first byte addressed can be at any location. The device automatically increments the address to the next higher address after shifting out each byte of data. Therefore, the device can read the whole memory with a single read bytes operation. When the device reaches the highest address, the address counter restarts at `0x000000`, allowing the memory contents to be read out indefinitely until the read bytes operation is terminated by driving `nCS` high. The device can drive `nCS` high any time after data is shifted out. If the read bytes operation is shifted in while a write or erase cycle is in progress, the operation will not be executed. Additionally, it will not have any effect on the write or erase cycle in progress.

Figure 4–11. Read Bytes Operation Timing Diagram



Notes to Figure 4–11:

(1) Address bit `A[23]` is a don't-care bit in the EPCS64 device. Address bits `A[23..21]` are don't-care bits in the EPCS16 device. Address bits `A[23..19]` are don't-care bits in the EPCS4 device. Address bits `A[23..17]` are don't-care bits in the EPCS1 device.

(2) For RPD files, the read sequence shifts out the LSB of the data byte first.

Read Silicon ID Operation

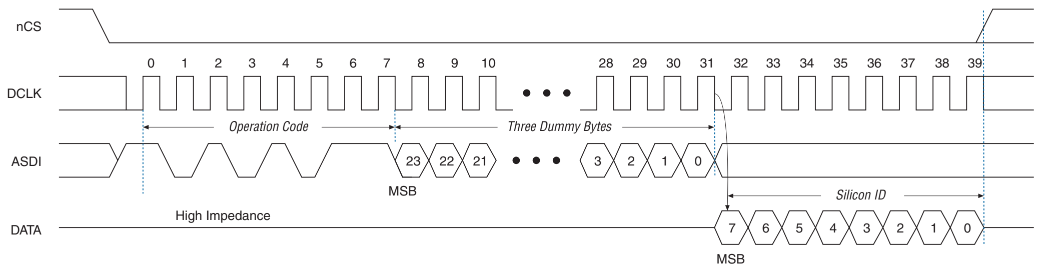
The read silicon ID operation code is `b'1010 1011`, with the MSB listed first. This operation reads the serial configuration device's 8-bit silicon ID from the `DATA` output pin. If this operation is shifted in during an erase or write cycle, it will be ignored and have no effect on the cycle that is in progress. Table 4–15 shows the serial configuration device silicon IDs.

Table 4–15. Serial Configuration Device Silicon ID

Serial Configuration Device	Silicon ID (Binary Value)
EPCS1	`b'0001 0000`
EPCS4	`b'0001 0010`
EPCS16	`b'0001 0100`
EPCS64	`b'0001 0110`

The device implements the read silicon ID operation by driving `nCS` low then shifting in the read silicon ID operation code followed by three dummy bytes on `ASDI`. The serial configuration device's 8-bit silicon ID is then shifted out on the `DATA` pin on the falling edge of `DCLK`, as shown in Figure 4–12. The device can terminate the read silicon ID operation by driving `nCS` high after the silicon ID has been read at least once. Sending additional clock cycles on `DCLK` while `nCS` is driven low can cause the silicon ID to be shifted out repeatedly.

Figure 4–12. Read Silicon ID Operation Timing Diagram



Write Bytes Operation

The write bytes operation code is `b'0000 0010`, with the MSB listed first. The write bytes operation allows bytes to be written to the memory. The write enable operation must be executed prior to the write bytes operation to set the write enable latch bit in the status register to `1`.

The write bytes operation is implemented by driving `nCS` low, followed by the write bytes operation code, three address bytes and a minimum one data byte on `ASDI`. If the eight least significant address bits (`A[7..0]`) are not all `0`, all sent data that goes beyond the end of the current page is not written into the next page. Instead, this data is written at the start address of the same page (from the address whose eight LSBs are all `0`). Drive `nCS` low during the entire write bytes operation sequence as shown in Figure 4–13.

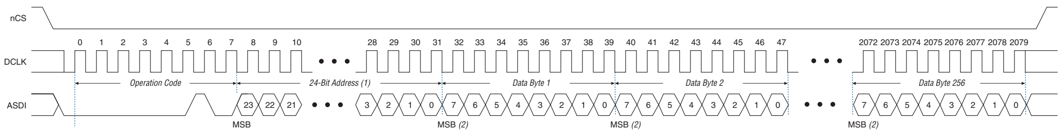
If more than 256 data bytes are shifted into the serial configuration device with a write bytes operation, the previously latched data is discarded and the last 256 bytes are written to the page. However, if less than 256 data bytes are shifted into the serial configuration device, they are guaranteed to be written at the specified addresses and the other bytes of the same page are unaffected.

If the design must write more than 256 data bytes to the memory, it needs more than one page of memory. Send the write enable and write bytes operation codes followed by three new targeted address bytes and 256 data bytes before a new page is written.

`nCS` must be driven high after the eighth bit of the last data byte has been latched in. Otherwise, the device will not execute the write bytes operation. The write enable latch bit in the status register is reset to `0` before the completion of each write bytes operation. Therefore, the write enable operation must be carried out before the next write bytes operation.

The device initiates the self-timed write cycle immediately after nCS is driven high. The self-timed write cycle usually takes 1.5 ms and is guaranteed to be less than 5 ms for all EPCS devices (see t_{WB} in Table 4–16). Therefore, the designer must account for this amount of delay before another page of memory is written. Alternatively, the designer can check the status register's write in progress bit by executing the read status operation while the self-timed write cycle is in progress. The write in progress bit is set to 1 during the self-timed write cycle, and is 0 when it is complete.

Figure 4–13. Write Bytes Operation Timing Diagram



Notes to Figure 4–13:

(1) Address bit A[23] is a don't-care bit in the EPCS64 device. Address bits A[23..21] are don't-care bits in the EPCS16 device. Address bits A[23..19] are don't-care bits in the EPCS4 device. Address bits A[23..17] are don't-care bits in the EPCS1 device.

(2) For RPD files, writes the LSB of the data byte first.

Erase Bulk Operation

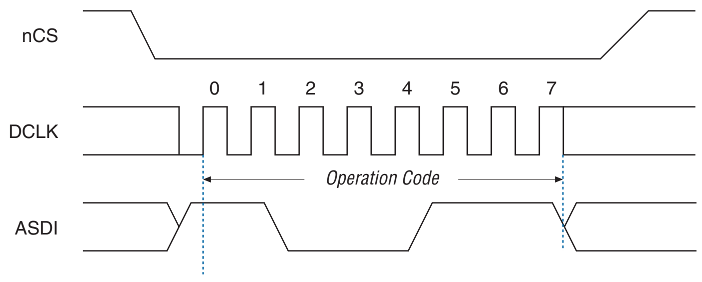
The erase bulk operation code is b'1100 0111, with the MSB listed first. The erase bulk operation sets all memory bits to 1 or 0xFF. Similar to the write bytes operation, the write enable operation must be executed prior to the erase bulk operation so that the write enable latch bit in the status register is set to 1.

Designers implement the erase bulk operation by driving nCS low and then shifting in the erase bulk operation code on the ASDI pin. nCS must be driven high after the eighth bit of the erase bulk operation code has been latched in. Figure 4–14 shows the timing diagram.

The device initiates the self-timed erase bulk cycle immediately after nCS is driven high. The self-timed erase bulk cycle usually takes 3 s for EPCS1 devices (guaranteed to be less than 6 s) and 5 s for EPCS4 devices (guaranteed to be less than 10 s). The erase bulk cycle times for EPCS16 is 17 s (guaranteed to be less than 40 s) and EPCS64 is 68 s (guaranteed to be less than 160 s). See t_{EB} in Table 4–16.

Designers must account for this delay before accessing the memory contents. Alternatively, designers can check the write in progress bit in the status register by executing the read status operation while the self-timed erase cycle is in progress. The write in progress bit is `1` during the self-timed erase cycle and is `0` when it is complete. The write enable latch bit in the status register is reset to `0` before the erase cycle is complete.

Figure 4–14. Erase Bulk Operation Timing Diagram



Erase Sector Operation

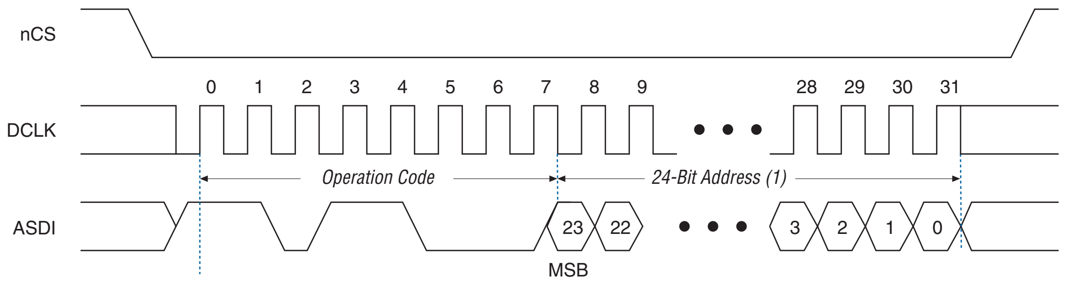
The erase sector operation code is `b'1101 1000`, with the MSB listed first. The erase sector operation allows the user to erase a certain sector in the serial configuration device by setting all bits inside the sector to `1` or `0xFF`. This operation is useful for users who access the unused sectors as general purpose memory in their applications.

The write enable operation must be executed prior to the erase sector operation so that the write enable latch bit in the status register is set to `1`.

The erase sector operation is implemented by first driving `nCS` low, then shifting in the erase sector operation code and the three address bytes of the chosen sector on the `ASDI` pin. The three address bytes for the erase sector operation can be any address inside the specified sector. (See Tables 4–6, 4–7, 4–8, and 4–9 for sector address range information.) Drive `nCS` high after the eighth bit of the erase sector operation code has been latched in. Figure 4–15 shows the timing diagram.

Immediately after the device drives `nCS` high, the self-timed erase sector cycle is initiated. The self-timed erase sector cycle usually takes 2 s and is guaranteed to be less than 3 s for all serial configuration devices. You must account for this amount of delay before the memory contents can be accessed. Alternatively, you can check the write in progress bit in the status register by executing the read status operation while the erase cycle is in progress. The write in progress bit is `1` during the self-timed erase cycle and is `0` when it is complete. The write enable latch bit in the status register is reset to `0` before the erase cycle is complete.

Figure 4–15. Erase Sector Operation Timing Diagram



Note to Figure 4–15:

(1) Address bit `A[23]` is a don't-care bit in the EPCS64 device. Address bits `A[23..21]` are don't-care bits in the EPCS16 device. Address bits `A[23..19]` are don't-care bits in the EPCS4 device. Address bits `A[23..17]` are don't-care bits in the EPCS1 device.

Power & Operation

This section describes the power modes, power-on reset (POR) delay, error detection, and initial programming state of serial configuration devices.

Power Mode

Serial configuration devices support active power and standby power modes. When `nCS` is low, the device is enabled and is in active power mode. The FPGA is configured while in active power mode. When `nCS` is high, the device is disabled but could remain in active power mode until all internal cycles have completed (such as write or erase operations). The serial configuration device then goes into stand-by power mode. The I_{CC1} parameter specifies the V_{CC} supply current when the device is in active power mode and the I_{CC0} parameter specifies the current when the device is in stand-by power mode (see Table 4–22).

Power-On Reset

During initial power-up, a POR delay occurs to ensure the system voltage levels have stabilized. During AS configuration, the FPGA controls the configuration and has a longer POR delay than the serial configuration device. Therefore, the POR delay is governed by the Stratix II FPGA (typically 12 ms or 100 ms) or Cyclone series FPGA (typically 100 ms).

Error Detection

During AS configuration with the serial configuration device, the FPGA monitors the configuration status through the nSTATUS and CONF_DONE pins. If an error condition occurs (nSTATUS drives low) or if the CONF_DONE pin does not go high, the FPGA will initiate reconfiguration by pulsing the nSTATUS and nCSO signals, which controls the chip select pin on the serial configuration device (nCS).

After an error, configuration automatically restarts if the *Auto-Restart Upon Frame Error* option is turned on in the Quartus II software. If the option is turned off, the system must monitor the nSTATUS signal for errors and then pulse the nCONFIG signal low to restart configuration.

Timing Information

Figure 4–16 shows the timing waveform for write operation to the serial configuration device.

Figure 4–16. Write Operation Timing

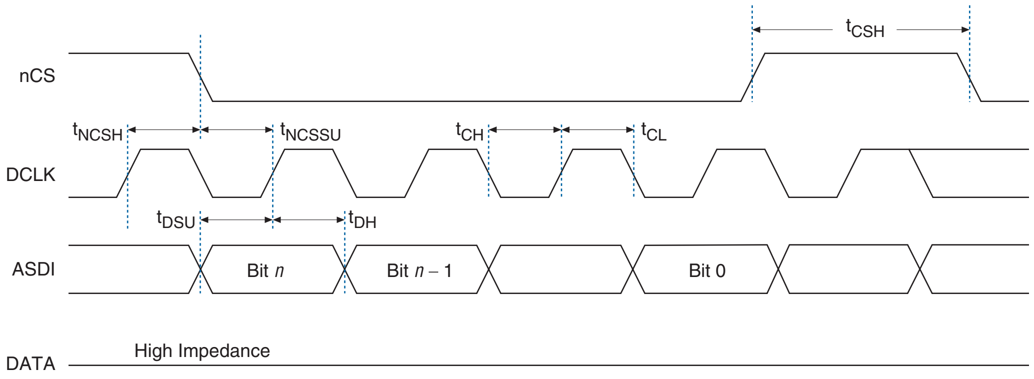


Table 4–16 defines the serial configuration device timing parameters for write operation.

Table 4–16. Write Operation Parameters

Symbol	Parameter	Min	Typ	Max	Unit
f_{WCLK}	Write clock frequency (from FPGA, download cable, or embedded processor) for write enable, write disable, read status, read silicon ID, write bytes, erase bulk, and erase sector operations			25	MHz
t_{CH}	`DCLK` high time	20			ns
t_{CL}	`DCLK` low time	20			ns
t_{NCSSU}	Chip select (`nCS`) setup time	10			ns
t_{NCSH}	Chip select (`nCS`) hold time	10			ns
t_{DSU}	Data (`ASDI`) in setup time before rising edge on `DCLK`	5			ns
t_{DH}	Data (`ASDI`) hold time after rising edge on `DCLK`	5			ns
t_{CSH}	Chip select high time	100			ns
t_{WB} *(1)*	Write bytes cycle time		1.5	5	ms
t_{WS} *(1)*	Write status cycle time		5	15	ms
t_{EB_EPCS1} *(1)*	Erase bulk cycle time for EPCS1 devices		3	6	s
t_{EB_EPCS4} *(1)*	Erase bulk cycle time for EPCS4 devices		5	10	s
t_{EB_EPCS16} *(1)*	Erase bulk cycle time for EPCS16 devices		17	40	s
t_{EB_EPCS64} *(1)*	Erase bulk cycle time for EPCS64 devices		68	160	s
t_{ES} *(1)*	Erase sector cycle time		2	3	s

Note to Table 4–16:

(1) These parameters are not shown in Figure 4–16.

Figure 4–17 shows the timing waveform for the serial configuration device's read operation.

Figure 4–17. Read Operation Timing

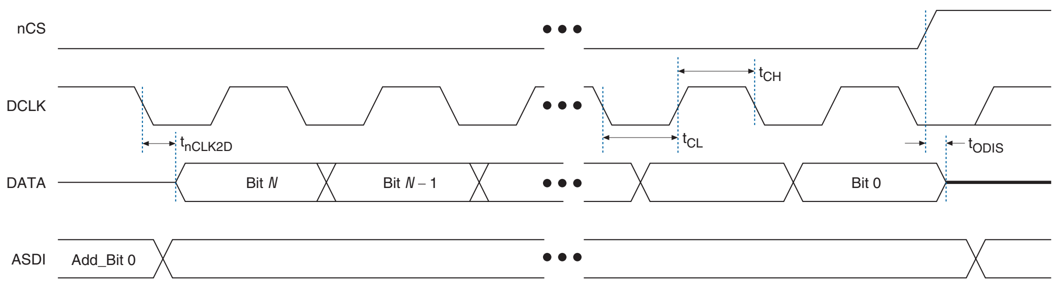


Table 4–17 defines the serial configuration device timing parameters for read operation.

Table 4–17. Read Operation Parameters

Symbol	Parameter	Min	Max	Unit
f_{RCLK}	Read clock frequency (from FPGA or embedded processor) for read bytes operation		20	MHz
t_{CH}	`DCLK` high time	25		ns
t_{CL}	`DCLK` low time	25		ns
t_{ODIS}	Output disable time after read		15	ns
t_{nCLK2D}	Clock falling edge to data		15	ns

Figure 4–18 shows the timing waveform for FPGA AS configuration scheme using a serial configuration device.

Figure 4–18. AS Configuration Timing

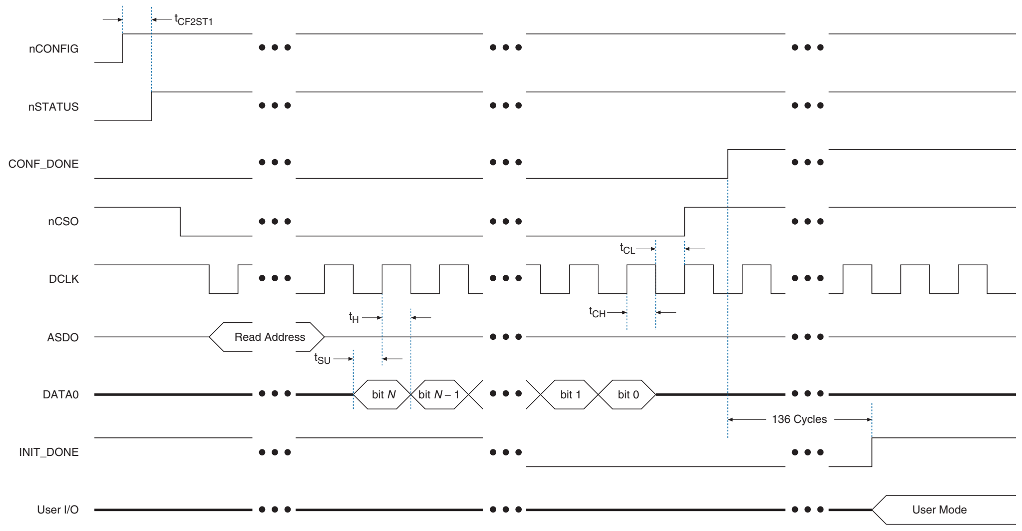


Table 4–18 shows the timing parameters for AS configuration mode.

Table 4–18. Timing Parameters for AS Configuration

Symbol	Parameter	Min	Typ	Max	Unit
f_{CLK}	DCLK frequency from Cyclone FPGA	14	17	20	MHz
f_{CLK}	DCLK frequency from Stratix II or Cyclone II FPGA	20	26	40	MHz
		10	13	20	MHz
t_{CH}	DCLK high time	10			ns
t_{CL}	DCLK low time	10			ns
t_{H}	Data hold time after rising edge on DCLK	0			ns
t_{SU}	Data set up time before rising edge on DCLK	5			ns
t_{POR}	POR delay			100	ms

Programming & Configuration File Support

The Quartus II design software provides programming support for serial configuration devices. After selecting the serial configuration device, the Quartus II software automatically generates the Programmer Object File (**.pof**) to program the device. The software allows users to select the appropriate serial configuration device density that most efficiently stores the configuration data for a selected FPGA.

The serial configuration device can be programmed in-system by an external microprocessor using SRunner. SRunner is a software driver developed for embedded serial configuration device programming that designers can customize to fit in different embedded systems. The SRunner can read RPD file and write to the serial configuration devices. The programming time is comparable to the Quartus II software programming time. Note that writing and reading the RPD file to the EPCS is different from other data and address bytes. The LSB of RPD bytes must be shifted out first during the read bytes instruction and the LSB of RPD bytes must be shifted in first during the write bytes instruction. This is because the FPGA reads the LSB of the RPD data first during the configuration process.

For more information about SRunner, refer to the *SRunner: An Embedded Solution for Serial Configuration Device Programming White Paper* and the source code on the Altera web site (**www.altera.com**).

Serial configuration devices can be programmed using the APU with the appropriate programming adapter (PLMSEPC-8 or PLMSEPC-16) via the Quartus II software, USB Blaster, EthernetBlaster, or the ByteBlaster II download cable via the Quartus II software. In addition, many third-party programmers, such as BP Microsystems and System General, offer programming hardware that supports serial configuration devices.

During in-system programming of a serial configuration device via the USB Blaster, EthernetBlaster, or ByteBlaster II download cable, the cable pulls `nCONFIG` low to reset the FPGA and overrides the 10-kΩ pull-down resistor on the FPGA's `nCE` pin (see Figure 4–2). The download cable then uses the four interface pins (`DATA`, `nCS`, `ASDI`, and `DCLK`) to program the serial configuration device. Once the programming is complete, the download cable releases the serial configuration device's four interface pins and the FPGA's `nCE` pin, and pulses `nCONFIG` to start configuration.

For more information on programming and configuration support, see the following documents:

- *Altera Programming Hardware Data Sheet*
- *Programming Hardware Manufacturers*
- *USB Blaster USB Port Download Cable Development Tools Data Sheet*
- *ByteBlaster II Parallel Port Download Cable Data Sheet*
- *EthernetBlaster Communications Cable User Guide*

Operating Conditions

Tables 4–19 through 4–23 provide information on absolute maximum ratings, recommended operating conditions, DC operating conditions, and capacitance for serial configuration devices.

Table 4–19. Absolute Maximum Ratings *Note (1)*

Symbol	Parameter	Condition	Min	Max	Unit
V_{CC}	Supply voltage	With respect to ground	–0.6	4.0	V
V_I	DC input voltage	With respect to ground	–0.6	4.0	V
I_{MAX}	DC V_{CC} or GND current			15	mA
I_{OUT}	DC output current per pin		–25	25	mA
P_D	Power dissipation			54	mW
T_{STG}	Storage temperature	No bias	–65	150	°C
T_{AMB}	Ambient temperature	Under bias	–65	135	°C
T_J	Junction temperature	Under bias		135	°C

Table 4–20. Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	*(2)*	3.0	3.6	V
V_I	Input voltage	Respect to GND	–0.3	0.3 + V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	70	°C
		For industrial use	–40	85	°C
t_R	Input rise time			5	ns
t_F	Input fall time			5	ns

Table 4–21. DC Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{IH}	High-level input voltage		0.7 × V_{CC}	V_{CC} + 0.4	V
V_{IL}	Low-level input voltage		–0.5	0.3 × V_{CC}	V
V_{OH}	High-level output voltage	I_{OH} = –100 µA *(3)*	V_{CC} –0.2		V
V_{OL}	Low-level output voltage	I_{OL} = 1.6 mA *(3)*		0.4	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10	10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–10	10	µA

Table 4–22. I_{CC} Supply Current

Symbol	Parameter	Conditions	Min	Max	Unit
I_{CC0}	V_{CC} supply current (standby)			50	µA
I_{CC1}	V_{CC} supply current (during active power mode)		5	15	mA

Table 4–23. Capacitance *Note (4)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input pin capacitance	V_{IN} = 0 V		6	pF
C_{OUT}	Output pin capacitance	V_{OUT} = 0 V		8	pF

Notes to Table 4–19 through 4–23:

(1) See the *Operating Requirements for Altera Devices Data Sheet*.

(2) Maximum V_{CC} rise time is 100 ms.

(3) The I_{OH} parameter refers to high-level TTL or CMOS output current; the I_{OL} parameter refers to low-level TTL or CMOS output current.

(4) Capacitance is sample-tested only at T_A = 25 °C and at a 20-MHz frequency.

Pin Information

As shown in Figure 4–19, the serial configuration device is an 8-pin or 16-pin device. The control pins on the serial configuration device are: serial data output (`DATA`), active serial data input (`ASDI`), serial clock (`DCLK`), and chip select (`nCS`). Table 4–24 shows the serial configuration device's pin descriptions.

Figure 4–19 shows the Altera serial configuration device 8-pin SOIC package and its pin-out diagram.

Figure 4–19. Altera Serial Configuration Device 8-Pin SOIC Package Pin-Out Diagram

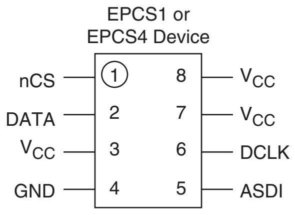
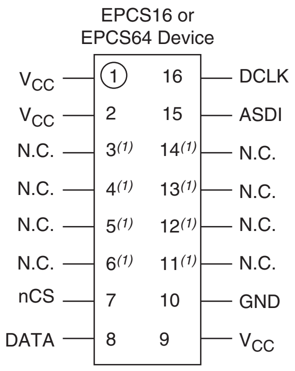


Figure 4–20 shows the Altera serial configuration device 16-pin SOIC package and its pin-out diagram.

Figure 4–20. Altera Serial Configuration Device 16-Pin SOIC Package Pin-Out Diagram



Note to Figure 4–20:

(1) These pins can be left floating or connected to V_{CC} or GND, whichever is more convenient on the board.

Table 4–24. Serial Configuration Device Pin Description (Part 1 of 2)

Pin Name	Pin Number in 8-Pin SOIC Package	Pin Number in 16-Pin SOIC Package	Pin Type	Description
DATA	2	8	Output	The DATA output signal transfers data serially out of the serial configuration device to the FPGA during read/configuration operation. During a read/configuration operations, the serial configuration device is enabled by pulling nCS low. The DATA signal transitions on the falling edge of DCLK.
ASDI	5	15	Input	The AS data input signal is used to transfer data serially into the serial configuration device. It receives the data that should be programmed into the serial configuration device. Data is latched in the rising edge of DCLK.
nCS	1	7	Input	The active low chip select input signal toggles at the beginning and end of a valid instruction. When this signal is high, the device is deselected and the DATA pin is tri-stated. When this signal is low, it enables the device and puts the device in an active mode. After power up, the serial configuration device requires a falling edge on the nCS signal before beginning any operation.

Table 4–24. Serial Configuration Device Pin Description (Part 2 of 2)

Pin Name	Pin Number in 8-Pin SOIC Package	Pin Number in 16-Pin SOIC Package	Pin Type	Description
`DCLK`	6	16	Input	`DCLK` is provided by the FPGA. This signal provides the timing of the serial interface. The data presented on `ASDI` is latched to the serial configuration device, at the rising edge of `DCLK`. Data on the `DATA` pin changes after the falling edge of `DCLK` and is latched into the FPGA on the rising edge.
V_{CC}	3, 7, 8	1,2,9	Power	Power pins connect to 3.3 V.
GND	4	10	Ground	Ground pin.

Package

All serial configuration devices are available in 8-pin or 16-pin plastic SOIC package.

For more information on Altera device packaging including mechanical drawing and specifications for this package, see the *Altera Device Package Information Data Sheet*.

Ordering Code

Table 4–25 shows the ordering codes for serial configuration devices.

Table 4–25. Serial Configuration Device Ordering Codes

Device	Ordering Code *(1)*
EPCS1	EPCS1SI8 EPCS1SI8N
EPCS4	EPCS4SI8 EPCS4SI8N
EPCS16	EPCS16SI16N
EPCS64	EPCS64SI16N

Note to Table 4–25:
(1) N: Lead free.

5. Configuration Devices for SRAM-Based LUT Devices Data Sheet

CF52005-2.1

Features

- Configuration device family for configuring Stratix® series, Cyclone™ series, APEX™ II, APEX 20K (including APEX 20K, APEX 20KC, and APEX 20KE), Mercury™, ACEX® 1K, and FLEX® (FLEX 10KE, and FLEX 10KA) devices
- Easy-to-use 4-pin interface to Altera® FPGAs
- Low current during configuration and near-zero standby current
- 5.0-V and 3.3-V operation
- Software design support with the Altera Quartus® II and MAX+PLUS® II development systems for Windows-based PCs as well as Sun SPARCstation, and HP 9000 Series 700/800
- Programming support with the Altera Programming Unit (APU) and programming hardware from Data I/O, BP Microsystems, and other third-party programmers
- Available in compact plastic packages
  - 8-pin plastic dual in-line package (PDIP)
  - 20-pin plastic J-lead chip carrier (PLCC) package
  - 32-pin plastic thin quad flat pack (TQFP) package
- EPC2 device has reprogrammable Flash configuration memory
  - 5.0-V and 3.3-V in-system programmability (ISP) through the built-in IEEE Std. 1149.1 Joint Test Action Group (JTAG) interface
  - Built-in JTAG boundary-scan test (BST) circuitry compliant with IEEE Std. 1149.1
  - ISP circuitry is compatible with IEEE Std. 1532
  - Supports programming through Serial Vector Format Files (**.svf**), Jam Standard Test and Programming Language (STAPL) Files (**.jam**), Jam STAPL Byte-Code Files (**.jbc**), and the Quartus II and MAX+PLUS II software via the USB Blaster, MasterBlaster™, ByteBlaster™ II, EthernetBlaster, or ByteBlasterMV™ download cable
  - `nINIT_CONF` pin allows INIT_CONF JTAG instruction to initiate FPGA configuration
  - Can be programmed with Programmer Object Files (**.pof**) for EPC1 and EPC1441 devices
  - Available in 20-pin PLCC and 32-pin TQFP packages

For detailed information on enhanced configuration devices, refer to *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet*. For detailed information on serial configuration devices, refer to *Serial Configuration Devices (EPCS1, EPCS4, EPCS16, & EPCS64) Data Sheet*.

Functional Description

With SRAM-based devices, configuration data must be reloaded each time the device powers up, the system initializes, or when new configuration data is needed. Altera configuration devices store configuration data for SRAM-based Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, FLEX 10K, and FLEX 6000 devices. Table 5–1 lists Altera configuration devices and their features.

Table 5–1. Altera Configuration Devices

Device	Memory Size (Bits)	ISP Support	Daisy Chain Support	Reprogrammable	Operating Voltage
EPC2	1,695,680	Yes	Yes	Yes	5.0 or 3.3 V
EPC1	1,046,496	No	Yes	No	5.0 or 3.3 V
EPC1441	440,800	No	No	No	5.0 or 3.3 V
EPC1213	212,942	No	Yes	No	5.0 V
EPC1064	65,536	No	No	No	5.0 V
EPC1064V	65,536	No	No	No	3.3 V

Table 5–2 lists the supported configuration device(s) required to configure a Stratix, Stratix GX, Cyclone, APEX II, APEX 20K, Mercury, ACEX 1K or FLEX device.

Table 5–2. Configuration Devices Required (Part 1 of 4)

Family	Device	Data Size (Bits) *(1)*	EPC1064 /1064V	EPC1213	EPC1441	EPC1	EPC2
Stratix II (1.2 V) *(2)*	EP2S15	5,000,000					3
	EP2S30	10,100,000					7
	EP2S60	17,100,000					11
	EP2S90	27,500,000					17
	EP2S130	39,600,000					24
	EP2S180	52,400,000					31
Stratix (1.5 V)	EP1S10	3,534,640					3 *(3)*
	EP1S20	5,904,832					4
	EP1S25	7,894,144					5
	EP1S30	10,379,368					7
	EP1S40	12,389,632					8
	EP1S60	17,543,968					11
	EP1S80	23,834,032					15

Table 5–2. Configuration Devices Required (Part 2 of 4)

Family	Device	Data Size (Bits) *(1)*	EPC1064 /1064V	EPC1213	EPC1441	EPC1	EPC2
Stratix GX (1.5 V)	EP1SGX10	3,534,640					3
	EP1SGX25	7,894,144					5
	EP1SGX40	12,389,632					8
Cyclone II (1.2 V) *(2)*	EP2C5	1,223,980					1
	EP2C8	1,983,792					2
	EP2C20	3,930,986					3
	EP2C35	7,071,234					5
	EP2C50	9,122,148					6
	EP2C70	10,249,694					7
Cyclone (1.5 V)	EP1C3	627,376				1	1
	EP1C4	925,000				1	1
	EP1C6	1,167,216				1 *(4)*	1
	EP1C12	2,326,528					1 *(4)*
	EP1C20	3,559,608					2 *(4)*
APEX II (1.5 V)	EP2A15	1,168,688					3
	EP2A25	1,646,544					4
	EP2A40	2,543,016					6
	EP2A70	4,483,064					11
Mercury (1.8 V)	EP1M120	1,303,120					1
	EP1M350	4,394,032					3
APEX 20KC (1.8 V)	EP20K200C	1,968,016					2
	EP20K400C	3,909,776					3
	EP20K600C	5,673,936					4
	EP20K1000C	8,960,016					6

Table 5–2. Configuration Devices Required (Part 3 of 4)

Family	Device	Data Size (Bits) (1)	EPC1064 /1064V	EPC1213	EPC1441	EPC1	EPC2
APEX 20KE (1.8 V)	EP20K30E	354,832			1	1	1
	EP20K60E	648,016				1	1
	EP20K100E	1,008,016				1	1
	EP20K160E	1,524,016					1
	EP20K200E	1,968,016					2
	EP20K300E	2,741,616					2
	EP20K400E	3,909,776					3
	EP20K600E	5,673,936					4
	EP20K1000E	8,960,016					6
	EP20K1500E	12,042,256					8
APEX 20K (2.5 V)	EP20K100	993,360				1	1
	EP20K200	1,950,800					2
	EP20K400	3,880,720					3
ACEX 1K (2.5 V)	EP1K10	159,160			1	1	1
	EP1K30	473,720				1	1
	EP1K50	784,184				1	1
	EP1K100	1,335,720					1
FLEX 10KE (2.5 V)	EPF10K30E	473,720				1	1
	EPF10K50E	784,184				1	1
	EPF10K50S	784,184				1	1
	EPF10K100B	1,200,000					1
	EPF10K100E	1,335,720					1
	EPF10K130E	1,838,360					2
	EPF10K200E	2,756,296					2
	EPF10K200S	2,756,296					2
FLEX 10KA (3.3V)	EPF10K10A	120,000			1	1	1
	EPF10K30A	406,000			1	1	1
	EPF10K50V	621,000				1	1
	EPF10K100A	1,200,000					1
	EPF10K130V	1,600,000					1
	EPF10K250A	3,300,000					2

Table 5–2. Configuration Devices Required (Part 4 of 4)

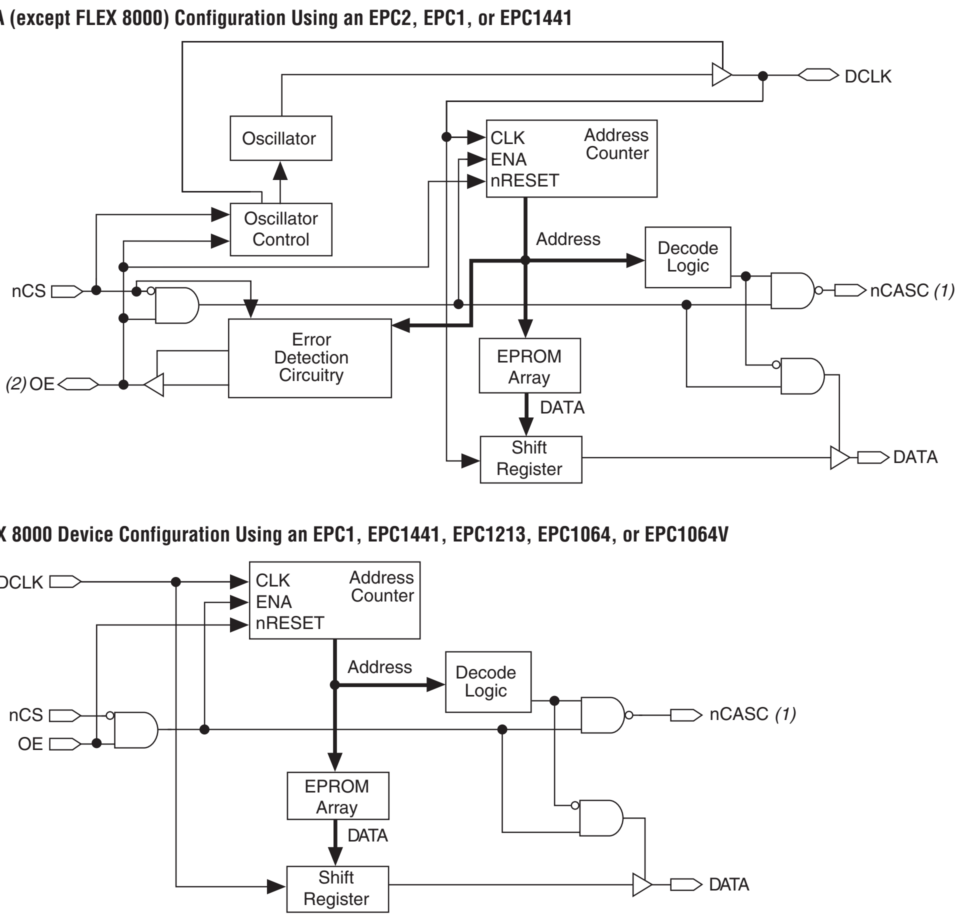
Family	Device	Data Size (Bits) (1)	EPC1064 /1064V	EPC1213	EPC1441	EPC1	EPC2
FLEX 10K (5.0V)	EPF10K10	118,000			1	1	1
	EPF10K20	231,000			1	1	1
	EPF10K30	376,000			1	1	1
	EPF10K40	498,000				1	1
	EPF10K50	621,000				1	1
	EPF10K70	892,000				1	1
	EPF10K100	1,200,000					1
FLEX 6000/A (3.3 V)	EPF6010A	260,000			1	1	
	EPF6016 (5.0V) / EPF6016A	260,000			1	1	
	EPF6024A	398,000			1	1	
FLEX 8000A (5.0V)	EPF8282A / EPF8282AV (3.3 V)	40,000	1	1	1	1	
	EPF8452A	64,000	1	1	1	1	
	EPF8636A	96,000		1	1	1	
	EPF8820A	128,000		1	1	1	
	EPF81188A	192,000		1	1	1	
	EPF81500A	250,000			1	1	

Notes to Table 5–2:

(1) Raw Binary Files (.rbf) were used to determine these sizes.
(2) Information is preliminary.
(3) EP1S10 ES devices requires four EPC2 devices
(4) This is with the Stratix II or Cyclone series compression feature enabled.

Figure 5–1 shows the configuration device block diagram.

Figure 5–1. Configuration Device Block Diagram



Notes to Figure 5–1:

(1) The EPC1441 devices do not support data cascading. The EPC2, EPC1, and EPC1213 devices support data cascading.

(2) The `OE` pin is a bidirectional open-drain pin.

Device Configuration

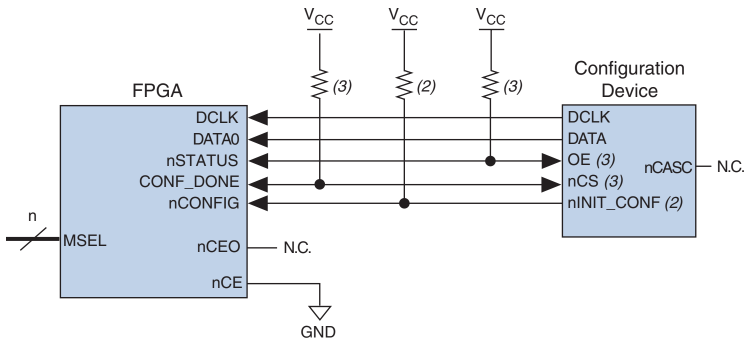
The EPC2, EPC1, and EPC1441 devices store configuration data in its EPROM array and serially clock data out using an internal oscillator. The `OE`, `nCS`, and `DCLK` pins supply the control signals for the address counter and the DATA output tri-state buffer. The configuration device sends a serial bitstream of configuration data to its DATA pin, which is routed to the DATA0 input of the FPGA.

The control signals for configuration devices (OE, nCS, and DCLK) interface directly with the FPGA control signals (nSTATUS, CONF_DONE, and DCLK, respectively). All Altera FPGAs can be configured by a configuration device without requiring an external intelligent controller.

☞ An EPC2 device cannot configure FLEX 8000 or FLEX 6000 devices. See Table 5–2 for the configuration devices that support FLEX 8000 and FLEX 6000 devices.

Figure 5–2 shows the basic configuration interface connections between the configuration device and the Altera FPGA. For specific details on configuration interface connections, including pull-up resistor values, supply voltages and MSEL pin setting, refer to the appropriate FPGA family chapter in the Configuration Handbook.

***Figure 5–2. Altera FPGA Configured Using an EPC2, EPC1, or EPC1441 Configuration Device** Note (1)*



Notes to Figure 5–2:

(1) For specific details on configuration interface connections refer to the FPGA family chapter in the Configuration Handbook.

(2) The nINIT_CONF pin (available on EPC2 devices) has an internal pull-up resistor that is always active. This means an external pull-up resistor is not required on the nINIT_CONF/nCONFIG line. The nINIT_CONF pin does not need to be connected if its functionality is not used. If nINIT_CONF is not used or not available, nCONFIG must be pulled to V_{CC} either directly or through a resistor.

(3) EPC2 devices have internal programmable pull-up resistors on OE and nCS. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the **Disable nCS and OE pull-ups on configuration device** option when generating programming files.

The EPC2 device allows the user to initiate configuration of the FPGA via an additional pin, `nINIT_CONF`. The `nINIT_CONF` pin of the EPC2 device can be connected to the `nCONFIG` of the FPGA(s), which allows the `INIT_CONF` JTAG instruction to initiate FPGA configuration. The `INIT_CONF` JTAG instruction causes the EPC2 device to drive `nINIT_CONF` low, which in turn pulls `nCONFIG` low. Pulling `nCONFIG` low on the FPGA will reset the device. When the JTAG state machine exits this state, `nINIT_CONF` is released and pulled high by an internal 1-kΩ resistor, which in turn pulls `nCONFIG` high to initiate configuration. If its functionality is not used, the `nINIT_CONF` pin does not need to be connected and `nCONFIG` of the FPGA must be pulled to V_{CC} either directly or through a resistor.

The EPC2 device's `OE` and `nCS` pins have internal programmable pull-up resistors. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the **Disable nCS and OE pull-ups on configuration device** option when generating programming files.

The configuration device's `OE` and `nCS` pins control the tri-state buffer on its `DATA` output pin, and enable the address counter and oscillator. When `OE` is driven low, the configuration device resets the address counter and tri-states its `DATA` pin. The `nCS` pin controls the `DATA` output of the configuration device. If `nCS` is held high after the `OE` reset pulse, the counter is disabled and the `DATA` output pin is tri-stated. If `nCS` is driven low after the OE reset pulse, the counter and `DATA` output pin are enabled. When `OE` is driven low again, the address counter is reset and the `DATA` output pin is tri-stated, regardless of the state of `nCS`.

If the FPGA's configuration data exceeds the capacity of a single EPC2 or EPC1 configuration device, multiple EPC2 or EPC1 devices can be cascaded together. If multiple EPC2 or EPC1 devices are required, the `nCASC` and `nCS` pins provide handshaking between the configuration devices.

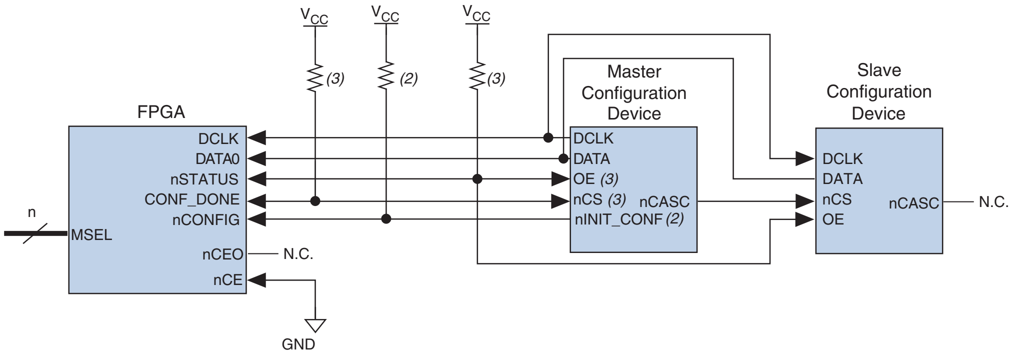
☞ EPC1441 and EPC1064/V devices cannot be cascaded.

When configuring Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10K devices with cascaded EPC2 or EPC1 devices, the position of the EPC2 or EPC1 device in the chain determines its mode of operation. The first configuration device in the chain is the master, while subsequent configuration devices are slaves. The `nINIT_CONF` pin of the master EPC2 device can be connected to the `nCONFIG` of the FPGAs, which allows the `INIT_CONF` JTAG instruction to initiate FPGA configuration. The `nCS` pin of the master configuration device is connected to the `CONF_DONE` of the FPGA(s), while its `nCASC` pin is connected to `nCS` of the next slave configuration device in the chain. Additional EPC2 or EPC1 devices can be chained together by connecting `nCASC` to `nCS` of the next slave EPC2 or EPC1 device in the chain. The last device's `nCS` input comes from the previous device, while its `nCASC` pin is left floating. All other configuration pins (`DCLK`, `DATA`, and `OE`) are connected to every device in the chain.

Figure 5–3 shows the basic configuration interface connections between a configuration device chain and the Altera FPGA.

For specific details on configuration interface connections, including pull-up resistor values, supply voltages and `MSEL` pin setting, refer to the appropriate FPGA family chapter in the *Configuration Handbook*.

Figure 5–3. Altera FPGA Configured Using Two EPC2 or EPC1 Configuration Devices Note (1)



Notes to Figure 5–3:

(1) For specific details on configuration interface connections refer to the appropriate FPGA family chapter in the Configuration Handbook.

(2) The `nINIT_CONF` pin (available on EPC2 devices) has an internal pull-up resistor that is always active. This means an external pull-up resistor is not required on the `nINIT_CONF/nCONFIG` line. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available, `nCONFIG` must be pulled to VCC either directly or through a resistor.

(3) EPC2 devices have internal programmable pull-up resistors on `OE` and `nCS`. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the **Disable nCS and OE pull-ups on configuration device** option when generating programming files.

When the first device in a configuration device chain is powered-up or reset, its `nCS` pin is driven low since it is connected to the `CONF_DONE` of the FPGA(s). Because both `OE` and `nCS` are low, the first device in the chain will recognize it is the master device and will control configuration. Since the slave devices' `nCS` pin is fed by the previous devices' `nCASC` pin, its `nCS` pin will be high upon power-up and reset. In the slave configuration devices, the `DATA` output is tri-stated and `DCLK` is an input. During configuration, the master device supplies the clock through `DCLK` to the FPGA(s) and to any slave configuration devices. The master EPC2 or EPC1 device also provides the first stream of data to the FPGA during multi-device configuration. After the master EPC2 or EPC1 device finishes sending configuration data, it tri-states its `DATA` pin to avoid contention with other configuration devices. The master EPC2 or EPC1 device will also drive its `nCASC` pin low, which pulls the `nCS` pin of the next device low. This action signals the slave EPC2 or EPC1 device to start sending configuration data to the FPGAs.

The master EPC2 or EPC1 device clocks all slave configuration devices until configuration is complete. Once all configuration data is transferred and the `nCS` pin on the master EPC2 or EPC1 device is driven high by the FPGA's `CONF_DONE` pin, the master EPC2 or EPC1 device then goes into zero-power (idle) state. The master EPC2 device drives `DATA` high and `DCLK` low, while the EPC1 and EPC1441 device tri-state `DATA` and drive `DCLK` low.

If `nCS` on the master EPC2 or EPC1 device is driven high before all configuration data is transferred, the master EPC2 or EPC1 device drives its `OE` signal low, which in turn drives the FPGA's `nSTATUS` pin low, indicating a configuration error. Additionally, if the configuration device sends all of its data and detects that `CONF_DONE` has not gone high, it recognizes that the FPGA has not configured successfully. EPC2 and EPC1 devices wait for 16 `DCLK` cycles after the last configuration bit was sent for `CONF_DONE` to reach a high state. In this case, the configuration device pulls its `OE` pin low, which in turn drives the target device's `nSTATUS` pin low. Configuration automatically restarts if the **Auto-restart configuration on error** option is turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box or the MAX+PLUS II software's **Global Project Device Options** dialog box (Assign menu).

For more information on FPGA configuration and configuration interface connections between configuration devices and Altera FPGA(s), refer to the appropriate FPGA family chapter in the *Configuration Handbook*.

Power & Operation

This section describes Power-On Reset (POR) delay, error detection, and 3.3-V and 5.0-V operation of Altera configuration devices.

Power-On Reset (POR)

During initial power-up, a POR delay occurs to permit voltage levels to stabilize. When configuring an FPGA with an EPC2, EPC1, or EPC1441 device, the POR delay occurs inside the configuration device, and the POR delay is a maximum of 200 ms. When configuring a FLEX 8000 device with an EPC1213, EPC1064, or EPC1064V device, the POR delay occurs inside the FLEX 8000 device, and the POR delay is typically, 100 ms, with a maximum of 200 ms.

During POR, the configuration device drives its `OE` pin low. This low signal delays configuration because the `OE` pin is connected to the target FPGA's `nSTATUS` pin. When the configuration device completes POR, it releases its open-drain `OE` pin, which is then pulled high by a pull-up resistor.

The FPGA(s) should be powered up before the configuration device exits POR to avoid the master configuration device from entering slave mode.

If the FPGA is not powered up before the configuration device exits POR, the `CONF_DONE/nCS` line will be high because of the pull-up resistor. When the configuration device exits POR and releases `OE`, it sees `nCS` high, which signals the configuration device to enter slave mode. Therefore, configuration will not begin (the `DATA` output is tri-stated and `DCLK` is an input pin in slave mode).

Error Detection Circuitry

The EPC2, EPC1, and EPC1441 configuration devices have built-in error detection circuitry for configuring Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, FLEX 10K or FLEX 6000 devices.

Built-in error-detection circuitry uses the `nCS` pin of the configuration device, which monitors the `CONF_DONE` pin on the FPGA. If `nCS` on the master EPC2 or EPC1 device is driven high before all configuration data is transferred, the master EPC2 or EPC1 device drives its `OE` signal low, which in turn drives the FPGA's `nSTATUS` pin low, indicating a configuration error. Additionally, if the configuration device sends all of its data and detects that `CONF_DONE` has not gone high, it recognizes that the FPGA has not configured successfully. EPC2 and EPC1 devices wait for 16 `DCLK` cycles after the last configuration bit was sent for `CONF_DONE` to reach a high state. In this case, the configuration device pulls its OE pin low, which in turn drives the target device's `nSTATUS` pin low. Configuration automatically restarts if the **Auto-restart configuration on error** option is turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box or the MAX+PLUS II software's **Global Project Device Options** dialog box (Assign menu).

In addition, if the FPGA detects a cyclic redundancy code (CRC) error in the received data, it will flag the error by driving `nSTATUS` low. This low signal on `nSTATUS` will drive the `OE` pin of the configuration device low, which will reset the configuration device. CRC checking is performed when configuring all Altera FPGAs.

3.3-V or 5.0-V Operation

The EPC2, EPC1 an EPC 1441 configuration device may be powered at 3.3 V or 5.0 V. For each configuration device, an option must be set for 5.0-V or 3.3-V operation.

For EPC1 and EPC1441 configuration devices, 3.3-V or 5.0-V operation is controlled by a programming bit in the POF. The **Low-Voltage mode** option in the **Options** tab of the **Configuration Device Options** dialog box in the Quartus II software or the **Use Low-Voltage Configuration EPROM** option in the **Global Project Device Options** dialog box (Assign menu) in the MAX+PLUS II software sets this parameter. For example, EPC1 devices are programmed automatically to operate in 3.3-V mode when configuring FLEX 10KA devices, which have a V_{CC} voltage of 3.3 V. In this example, the EPC1 device's V_{CC} pin is connected to a 3.3-V power supply.

For EPC2 devices, this option is set externally by the `VCCSEL` pin. In addition, the EPC2 device has an externally controlled option, set by the `VPPSEL` pin, to adjust the programming voltage to 5.0 V or 3.3 V. The functions of the `VCCSEL` and `VPPSEL` pins are described below. These pins are only available in the EPC2 devices.

- `VCCSEL` pin - For EPC2 configuration devices, 5.0-V or 3.3-V operation is controlled by the `VCCSEL` option pin. The device functions in 5.0-V mode when `VCCSEL` is connected to GND; the device functions in 3.3-V mode when `VCCSEL` is connected to V_{CC}.
- `VPPSEL` pin - The EPC2 `VPP` programming power pin is normally tied to V_{CC}. For EPC2 devices operating at 3.3 V, it is possible to improve in-system programming times by setting `VPP` to 5.0 V. For all other configuration devices, `VPP` must be tied to V_{CC}. The EPC2 device's `VPPSEL` pin must be set in accordance with the EPC2 `VPP` pin. If the `VPP` pin is supplied by a 5.0-V supply, `VPPSEL` must be connected to GND; if the `VPP` pin is supplied by a 3.3-V power supply, `VPPSEL` must be connected to V_{CC}.

Table 5–3 describes the relationship between the V_{CC} and V_{PP} voltage levels and the required logic level for `VCCSEL` and `VPPSEL`. A logic level of high means the pin should be connected to V_{CC}, while a low logic level means the pin should be connected to GND.

Table 5–3. VCCSEL & VPPSEL Pin Functions on the EPC2

V_{CC} Voltage Level (V)	V_{PP} Voltage Level (V)	VCCSEL Pin Logic Level	VPPSEL Pin Logic Level
3.3	3.3	High	High
3.3	5.0	High	Low
5.0	5.0	Low	Low

At 3.3-V operation, all EPC2 inputs are 5.0-V tolerant, except DATA, DCLK, and nCASC. The DATA and DCLK pins are used only to interface between the EPC2 device and the FPGA it is configuring. The voltage tolerances of all EPC2 pins at 5.0 V and 3.3 V are listed in Table 5–4.

Table 5–4. EPC2 Input & Bidirectional Pin Voltage Tolerance

Pin	5.0-V Operation		3.3-V Operation	
	5.0-V Tolerant	3.3-V Tolerant	5.0-V Tolerant	3.3-V Tolerant
DATA	✓	✓		✓
DCLK	✓	✓		✓
nCASC	✓	✓		✓
OE	✓	✓	✓	✓
nCS	✓	✓	✓	✓
VCCSEL	✓	✓	✓	✓
VPPSEL	✓	✓	✓	✓
nINIT_CONF	✓	✓	✓	✓
TDI	✓	✓	✓	✓
TMS	✓	✓	✓	✓
TCK	✓	✓	✓	✓

If an EPC2, EPC1 or EPC1441 configuration device is powered at 3.3 V, the nSTATUS and CONF_DONE pull-up resistors must be connected to 3.3 V. If these configuration devices are powered at 5.0 V, the nSTATUS and CONF_DONE pull-up resistors can be connected to 3.3 V or 5.0 V.

Programming & Configuration File Support

The Quartus II and MAX+PLUS II development systems provide programming support for Altera configuration devices. During compilation, the Quartus II and MAX+PLUS II software automatically generates a POF, which can be used to program the configuration device(s). In a multi-device project, the software can combine the programming files for multiple Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10K devices into one or more configuration devices. The software allows you to select the appropriate configuration device to most efficiently store the data for each FPGA.

All Altera configuration devices are programmable using Altera programming hardware in conjunction with the Quartus II or MAX+PLUS II software. In addition, many third part programmers offer programming hardware that supports Altera configuration devices.

An EPC2 device can be programmed with a POF generated for an EPC1 or EPC1441 device. An EPC1 device can be programmed using a POF generated for an EPC1441 device.

EPC2 configuration devices can be programmed in-system through its industry-standard 4-pin JTAG interface. ISP capability in the EPC2 devices provides ease in prototyping and FPGA functionality. When programming multiple EPC2 devices in a JTAG chain, the Quartus II and MAX+PLUS II software and other programming methods employ concurrent programming to simultaneously program multiple devices and reduce programming time. EPC2 devices can be programmed and erased up to 100 times.

After programming an EPC2 device in-system, FPGA configuration can be initiated by the EPC2 INIT_CONF JTAG instruction. See Table 5–6.

For more information on programming and configuration support, refer to the following documents:

- *Altera Programming Hardware Data Sheet*
- *USB Blaster USB Port Download Cable Data Sheet*
- *MasterBlaster Serial/USB Communications Cable Data Sheet*
- *ByteBlaster II Parallel Port Download Cable Data Sheet*
- *ByteBlasterMV Parallel Port Download Cable Data Sheet*
- *ByteBlaster Parallel Port Download Cable Data Sheet*
- *BitBlaster Parallel Port Download Cable Data Sheet*

You can also program configuration devices using the Quartus II or MAX+PLUS II software with the Altera Programming Unit (APU), and the appropriate configuration device programming adapter. Table 5–5 shows which programming adapter to use with each configuration device.

Table 5–5. Programming Adapters

Device	Package	Adapter
EPC2	20-pin J-Lead 32-pin TQFP	PLMJ1213 PLMT1213
EPC1	8-pin DIP 20-pin J-Lead	PLMJ1213 PLMJ1213
EPC1441	8-pin DIP 20-pin J-Lead 32-pin TQFP	PLMJ1213 PLMJ1213 PLMT1064

The following steps explain how to program Altera configuration devices using the Quartus II software and the APU:

1. Choose the **Quartus II Programmer** (Tools menu).

2. Load the appropriate POF by clicking **Add**. The **Device** column displays the device for the current programming file.

3. Insert a blank configuration device into the programming adapter's socket.

4. Turn on the **Program/Configure**. You can also turn on **Verify** to verify the contents of a programmed device against the programming data loaded from a programming file.

5. Click **Start**.

6. After successful programming, you can place the configuration device on the PCB to configure the FPGA device.

The following steps explain how to program Altera configuration devices using the MAX+PLUS II software and the APU:

1. Open the MAX+PLUS II Programmer.

2. Load the appropriate POF using the **Select Programming File** dialog box (File menu). By default, the **Programmer** loads the current project's POF. The **Device** field displays the device for the current programming file.

3. Insert a blank configuration device into the programming adapter's socket.

4. Click **Program**.

5. After successful programming, you can place the configuration device on the PCB to configure the FPGA device.

If you are cascading EPC1 or EPC2 devices, you must generate multiple POFs. The first device POF will have the same name as the project, while the second device POF will have the same name as the first, but with a "_1" extension (e.g., **top_1.pof**).

IEEE Std. 1149.1 (JTAG) Boundary-Scan Testing

The EPC2 provides JTAG BST circuitry that complies with the IEEE Std. 1149.1-1990 specification. JTAG boundary-scan testing can be performed before or after configuration, but not during configuration. The EPC2 device supports the JTAG instructions shown in Table 6.

The ISP circuitry in EPC2 devices is compatible with tools that support the IEEE Std. 1532. The IEEE Std. 1532 is a standard developed to allow concurrent ISP between multiple PLD vendors.

Table 5–6. EPC2 JTAG Instructions (Part 1 of 2)

JTAG Instruction	OPCODE	Description
SAMPLE/PRELOAD	`00 0101 0101`	Allows a snapshot of a signal at the device pins to be captured and examined during normal device operation, and permits an initial data pattern output at the device pins.
EXTEST	`00 0000 0000`	Allows the external circuitry and board-level interconnections to be tested by forcing a test pattern at the output pins and capturing results at the input pins.
BYPASS	`11 1111 1111`	Places the 1-bit bypass register between the `TDI` and `TDO` pins, which allows the BST data to pass synchronously through a selected device to adjacent devices during normal device operation.
IDCODE	`00 0101 1001`	Selects the device IDCODE register and places it between `TDI` and `TDO`, allowing the device IDCODE to be serially shifted out of `TDO`. The device IDCODE for the EPC2 configuration device is shown below: 0000 0001000000000010 00001101110 1
USERCODE	`00 0111 1001`	Selects the USERCODE register and places it between `TDI` and `TDO`, allowing the USERCODE to be serially shifted out of `TDO`. The 32-bit USERCODE is a programmable user-defined pattern.

Table 5–6. EPC2 JTAG Instructions (Part 2 of 2)

JTAG Instruction	OPCODE	Description
INIT_CONF	`00 0110 0001`	This function initiates the FPGA re-configuration process by pulsing the `nINIT_CONF` pin low, which is connected to the FPGA(s) `nCONFIG` pin(s). After this instruction is updated, the `nINIT_CONF` pin is pulsed low when the JTAG state machine enters the Run-Test/Idle state. The `nINIT_CONF` pin is then released and `nCONFIG` is pulled high by the resistor after the JTAG state machine goes out of Run-Test/Idle state. The FPGA configuration starts after `nCONFIG` goes high. As a result, the FPGA is configured with the new configuration data stored in the configuration device. This function can be added to your programming file (POF, JAM, JBC) in the Quartus II software by enabling the **Initiate configuration after programming** option in the **Programmer options** window (Options menu). This instruction is also used by the MAX+PLUS II software, Jam STAPL Files, and JBC Files.
ISP Instructions	-	These instructions are used when programming an EPC2 device via JTAG ports with a USB Blaster, MasterBlaster, ByteBlaster II, EthernetBlaster, or ByteBlaster MV download cable, or using a Jam STAPL File (**.jam**), Jam STAPL Byte-Code File (**.jbc**), or SVF file via an embedded processor.

For more information, refer to *Application Note 39 (IEEE 1149.1 (JTAG) Boundary-Scan Testing in Altera Devices)* or the EPC2 BSDL files on the Altera web site.

Figure 5–4 shows the timing requirements for the JTAG signals.

Figure 5–4. EPC2 JTAG Waveforms

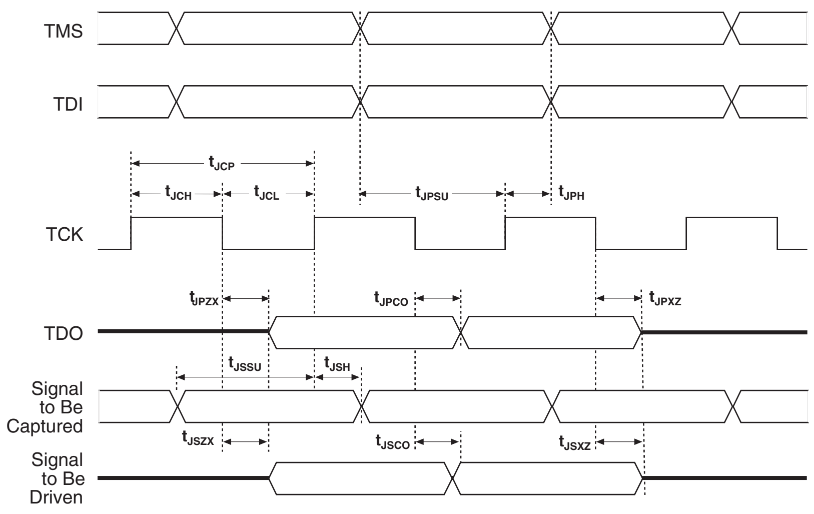


Table 5–7 shows the timing parameters and values for configuration devices.

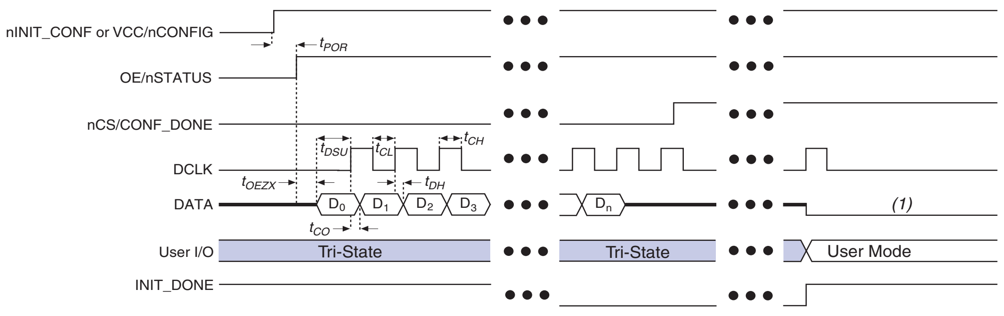
Table 5–7. JTAG Timing Parameters & Values

Symbol	Parameter	Min	Max	Unit
t_{JCP}	`TCK` clock period	100		ns
t_{JCH}	`TCK` clock high time	50		ns
t_{JCL}	`TCK` clock low time	50		ns
t_{JPSU}	JTAG port setup time	20		ns
t_{JPH}	JTAG port hold time	45		ns
t_{JPCO}	JTAG port clock to output		25	ns
t_{JPZX}	JTAG port high impedance to valid output		25	ns
t_{JPXZ}	JTAG port valid output to high impedance		25	ns
t_{JSSU}	Capture register setup time	20		ns
t_{JSH}	Capture register hold time	45		ns
t_{JSCO}	Update register clock to output		25	ns
t_{JSZX}	Update register high-impedance to valid output		25	ns
t_{JSXZ}	Update register valid output to high impedance		25	ns

Timing Information

Figure 5–5 shows the timing waveform when using a configuration device.

Figure 5–5. Timing Waveform Using a Configuration Device



Note to Figure 5–5:

(1) The EPC2 device will drive DCLK low and DATA high after configuration. The EPC1 and EPC1441 device will drive DCLK low and tri-state DATA after configuration.

Table 5–8 defines the timing parameters when using EPC2 devices at 3.3 V.

Table 5–8. Timing Parameters when Using EPC2 devices at 3.3 V (Part 1 of 2)

Symbol	Parameter	Min	Typ	Max	Units
t_{POR}	POR delay *(1)*			200	ms
t_{OEZX}	OE high to DATA output enabled			80	ns
t_{CE}	OE high to first rising edge on DCLK			300	ns
t_{DSU}	Data setup time before rising edge on DCLK	30			ns
t_{DH}	Data hold time after rising edge on DCLK	0			ns
t_{CO}	DCLK to DATA out			30	ns
t_{CDOE}	DCLK to DATA enable/disable			30	ns
f_{CLK}	DCLK frequency	5	7.7	12.5	MHz
t_{MCH}	DCLK high time for the first device in the configuration chain	40	65	100	ns
t_{MCL}	DCLK low time for the first device in the configuration chain	40	65	100	ns
t_{SCH}	DCLK high time for subsequent devices	40			ns
t_{SCL}	DCLK low time for subsequent devices	40			ns
t_{CASC}	DCLK rising edge to nCASC			25	ns

Table 5–8. Timing Parameters when Using EPC2 devices at 3.3 V (Part 2 of 2)

Symbol	Parameter	Min	Typ	Max	Units
t_{CCA}	nCS to nCASC cascade delay			15	ns
t_{OEW}	OE low pulse width (reset) to guarantee counter reset	100			ns
t_{OEC}	OE low (reset) to DCLK disable delay			30	ns
t_{NRCAS}	OE low (reset) to nCASC delay			30	ns

Note to Table 5–8:

(1) During initial power-up, a POR delay occurs to permit voltage levels to stabilize. Subsequent reconfigurations do not incur this delay.

Table 5–9 defines the timing parameters when using EPC1 and EPC1441 devices at 3.3 V.

Table 5–9. Timing Parameters when Using EPC1 & EPC1441 Devices at 3.3 V (Part 1 of 2)

Symbol	Parameter	Min	Typ	Max	Units
t_{POR}	POR delay *(1)*			200	ms
t_{OEZX}	OE high to DATA output enabled			80	ns
t_{CE}	OE high to first rising edge on DCLK			300	ns
t_{DSU}	Data setup time before rising edge on DCLK	30			ns
t_{DH}	Data hold time after rising edge on DCLK	0			ns
t_{CO}	DCLK to DATA out			30	ns
t_{CDOE}	DCLK to DATA enable/disable			30	ns
f_{CLK}	DCLK frequency	2	4	10	MHz
t_{MCH}	DCLK high time for the first device in the configuration chain	50	125	250	ns
t_{MCL}	DCLK low time for the first device in the configuration chain	50	125	250	ns
t_{SCH}	DCLK high time for subsequent devices	50			ns
t_{SCL}	DCLK low time for subsequent devices	50			ns
t_{CASC}	DCLK rising edge to nCASC			25	ns
t_{CCA}	nCS to nCASC cascade delay			15	ns
t_{OEW}	OE low pulse width (reset) to guarantee counter reset	100			ns
t_{OEC}	OE low (reset) to DCLK disable delay			30	ns

Table 5–9. Timing Parameters when Using EPC1 & EPC1441 Devices at 3.3 V (Part 2 of 2)

Symbol	Parameter	Min	Typ	Max	Units
t_{NRCAS}	`OE` low (reset) to `nCASC` delay			30	ns

Note to Table 5–9:
(1) During initial power-up, a POR delay occurs to permit voltage levels to stabilize. Subsequent reconfigurations do not incur this delay.

Table 5–10 defines the timing parameters when using EPC2, EPC1, and EPC1441 devices at 5.0 V.

Table 5–10. Timing Parameters when Using EPC2, EPC1 & EPC1441 Devices at 5.0 V

Symbol	Parameter	Min	Typ	Max	Units
t_{POR}	POR delay (1)			200	ms
t_{OEZX}	`OE` high to `DATA` output enabled			50	ns
t_{CE}	`OE` high to first rising edge on `DCLK`			200	ns
t_{DSU}	Data setup time before rising edge on `DCLK`	30			ns
t_{DH}	Data hold time after rising edge on `DCLK`	0			ns
t_{CO}	`DCLK` to `DATA` out			20	ns
t_{CDOE}	`DCLK` to `DATA` enable/disable			20	ns
f_{CLK}	`DCLK` frequency	6.7	10	16.7	MHz
t_{MCH}	`DCLK` high time for the first device in the configuration chain	30	50	75	ns
t_{MCL}	`DCLK` low time for the first device in the configuration chain	30	50	75	ns
t_{SCH}	`DCLK` high time for subsequent devices	30			ns
t_{SCL}	`DCLK` low time for subsequent devices	30			ns
t_{CASC}	`DCLK` rising edge to `nCASC`			20	ns
t_{CCA}	`nCS` to `nCASC` cascade delay			10	ns
t_{OEW}	`OE` low pulse width (reset) to guarantee counter reset	100			ns
t_{OEC}	`OE` low (reset) to `DCLK` disable delay			20	ns
t_{NRCAS}	`OE` low (reset) to `nCASC` delay			25	ns

Note to Table 5–10:
(1) During initial power-up, a POR delay occurs to permit voltage levels to stabilize. Subsequent reconfigurations do not incur this delay.

Table 5–11 defines the timing parameters when using EPC1, EPC1441, EPC1213, EPC1064, and EPC1064V devices when configuring FLEX 8000 device.

Table 5–11. FLEX 8000 Device Configuration Parameters Using EPC1, EPC1441, EPC1213, EPC1064 & EPC1064V Devices

Symbol	Parameter	EPC1064V		EPC1064 EPC1213		EPC1 EPC1441		Unit
		Min	Max	Min	Max	Min	Max	
t_{OEZX}	`OE` high to `DATA` output enabled		75		50		50	ns
t_{CSZX}	`nCS` low to `DATA` output enabled		75		50		50	ns
t_{CSXZ}	`nCS` high to `DATA` output disabled		75		50		50	ns
t_{CSS}	`nCS` low setup time to first `DCLK` rising edge	150		100		50		ns
t_{CSH}	`nCS` low hold time after `DCLK` rising edge	0		0		0		ns
t_{DSU}	Data setup time before rising edge on `DCLK`	75		50		50		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		0		0		ns
t_{CO}	`DCLK` to `DATA` out delay		100		75		75	ns
t_{CK}	Clock period	240		160		100		ns
f_{CK}	Clock frequency		4		6		8	MHz
t_{CL}	`DCLK` low time	120		80		50		ns
t_{CH}	`DCLK` high time	120		80		50		ns
t_{XZ}	`OE` low or `nCS` high to `DATA` output disabled		75		50		50	ns
t_{OEW}	`OE` pulse width to guarantee counter reset	150		100		100		ns
t_{CASC}	Last `DCLK` + 1 to `nCASC` low delay		90		60		50	ns
t_{CKXZ}	Last `DCLK` + 1 to `DATA` tri-state delay		75		50		50	ns
t_{CEOUT}	`nCS` high to `nCASC` high delay		150		100		100	ns

Operating Conditions

Tables 5–12 through 5–19 provide information on absolute maximum ratings, recommended operating conditions, DC operating conditions, and capacitance for configuration devices.

Table 5–12. Absolute Maximum Ratings Note (1)

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to ground (2)	–2.0	7.0	V
V_I	DC input voltage	With respect to ground (2)	–2.0	7.0	V
I_{MAX}	DC V_{CC} or ground current			50	mA
I_{OUT}	DC output current, per pin		–25	25	mA
P_D	Power dissipation			250	mW
T_{STG}	Storage temperature	No bias	–65	150	° C
T_{AMB}	Ambient temperature	Under bias	–65	135	° C
T_J	Junction temperature	Under bias		135	° C

Table 5–13. Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage for 5.0-V operation	(3), (4)	4.75 (4.50)	5.25 (5.50)	V
	Supply voltage for 3.3-V operation	(3), (4)	3.0 (3.0)	3.6 (3.6)	V
V_I	Input voltage	With respect to ground	–0.3	V_{CC} + 0.3 (5)	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	70	° C
		For industrial use	–40	85	° C
t_R	Input rise time			20	ns
t_F	Input fall time			20	ns

Table 5–14. DC Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{IH}	High-level input voltage		2.0	$V_{CC} + 0.3$ (5)	V
V_{IL}	Low-level input voltage		–0.3	0.8	V
V_{OH}	5.0-V mode high-level TTL output voltage	I_{OH} = –4 mA DC (6)	2.4		V
	3.3-V mode high-level CMOS output voltage	I_{OH} = –0.1 mA DC (6)	$V_{CC} – 0.2$		V
V_{OL}	Low-level output voltage	I_{OL} = 4 mA DC (6)		0.4	V
I_I	Input leakage current	$V_I = V_{CC}$ or ground	–10	10	μA
I_{OZ}	Tri-state output off-state current	$V_O = V_{CC}$ or ground	–10	10	μA

Table 5–15. EPC1213, EPC1064 & EPC1064V Device I_{CC} Supply Current Values

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{CC0}	V_{CC} supply current (standby)			100	200	μA
I_{CC1}	V_{CC} supply current (during configuration)			10	50	mA

Table 5–16. EPC2 Device Values

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{CC0}	V_{CC} supply current (standby)	V_{CC} = 5.0 V or 3.3 V		50	100	μA
I_{CC1}	V_{CC} supply current (during configuration)	V_{CC} = 5.0 V or 3.3 V		18	50	μA
R_{CONF}	Configuration pins	Internal pull up (`OE`, `nCS`, `nINIT_CONF`)		1		kΩ

Table 5–17. EPC1 Device I_{CC} Supply Current Values

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{CC0}	V_{CC} supply current (standby)			50	100	μA
I_{CC1}	V_{CC} supply current (during configuration)	V_{CC} = 5.0 V		30	50	mA
		V_{CC} = 3.3 V		10	16.5	mA

Table 5–18. EPC1441 Device I_{CC} Supply Current Values

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{CC0}	V_{CC} supply current (standby)			30	60	µA
I_{CC1}	V_{CC} supply current (during configuration)	V_{CC} = 5.0 V		15	30	mA
I_{CC1}	V_{CC} supply current (during configuration)	V_{CC} = 3.3 V		5	10	mA

Table 5–19. Capacitance *Note (7)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output pin capacitance	V_{OUT} = 0 V, f = 1.0 MHz		10	pF

Notes to Tables 5–12 through 5–19:
(1) See the Operating Requirements for Altera Devices Data Sheet.
(2) The minimum DC input is -0.3 V. During transitions, the inputs may undershoot to -2.0 V or overshoot to 7.0 V for input currents less than 100 mA and periods shorter than 20 ns under no-load conditions.
(3) Numbers in parentheses are for industrial temperature range devices.
(4) Maximum V_{CC} rise time is 100 ms.
(5) Certain EPC2 pins may be driven to 5.75 V when operated with a 3.3-V V_{CC}. See Table 5–4.
(6) The I_{OH} parameter refers to high-level TTL or CMOS output current; the I_{OL} parameter refers to low-level TTL or CMOS output current.
(7) Capacitance is sample-tested only.

Pin Information

Table 5–20 describes EPC2, EPC1, and EPC1441 pin functions during device configuration.

For pin information on enhanced configuration devices, refer to the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet*. For pin information on serial configuration devices, refer to the *Serial Configuration Devices (EPCS1, EPCS4, EPCS16 & EPCS64) Data Sheet*.

Table 5–20. EPC2, EPC1 & EPC1441 Pin Functions During Configuration (Part 1 of 4)

Pin Name	Pin Number			Pin Type	Description
	8-Pin PDIP *(1)*	20-Pin PLCC	32-Pin TQFP *(2)*		
`DATA`	1	2	31	Output	Serial data output. The `DATA` pin connects to the `DATA0` of the FPGA. `DATA` is latched into the FPGA on the rising edge of `DCLK`. The `DATA` pin is tri-stated before configuration and when the nCS pin is high. After configuration, the EPC2 device will drive `DATA` high, while the EPC1 and EPC1441 device will tri-state `DATA`.
`DCLK`	2	4	2	Bidirectional	Clock output when configuring with a single configuration device or when the configuration device is the first (master) device in a chain. Clock input for the next (slave) configuration devices in a chain. The `DCLK` pin connects to the `DCLK` of the FPGA. Rising edges on `DCLK` increment the internal address counter and present the next bit of data on the `DATA` pin. The counter is incremented only if the OE input is held high, the `nCS` input is held low, and all configuration data has not been transferred to the target device. After configuration or when `OE` is low, the EPC2, EPC1 and EPC1441 device will drive DCLK low.

Table 5–20. EPC2, EPC1 & EPC1441 Pin Functions During Configuration (Part 2 of 4)

Pin Name	Pin Number			Pin Type	Description
	8-Pin PDIP *(1)*	20-Pin PLCC	32-Pin TQFP *(2)*		
OE	3	8	7	Open-Drain Bidirectional	Output enable (active high) and reset (active low). The OE pin connects to the nSTAUTUS of the FPGA. A low logic level resets the address counter. A high logic level enables DATA and the address counter to count. If this pin is low (reset) during configuration, the internal oscillator becomes inactive and DCLK drives low. See "Error Detection Circuitry" on page 5–12. The OE pin has an internal programmable 1-kΩ resistor in EPC2 devices. If internal pull-up resistors are use, external pull-up resistors should not be used on these pins. The internal pull-up resistors can be disabled through the **Disable nCS and OE pull-ups on configuration device** option.
nCS	4	9	10	Input	Chip select input (active low). The nCS pin connects to the CONF_DONE of the FPGA. A low input allows DCLK to increment the address counter and enables DATA to drive out. If the EPC2 or EPC1 is reset (OE pulled low) while nCS is low, the device initializes as the master device in a configuration chain. If the EPC2 or EPC1 device is reset (OE pulled low) while nCS is high, the device initializes as a slave device in the chain. The nCS pin has an internal programmable 1-kΩ resistor in EPC2 devices. If internal pull-up resistors are use, external pull-up resistors should not be used on these pins.The internal pull-up resistors can be disabled through the **Disable nCS and OE pull-ups on configuration device** option.

Table 5–20. EPC2, EPC1 & EPC1441 Pin Functions During Configuration (Part 3 of 4)

Pin Name	Pin Number			Pin Type	Description
	8-Pin PDIP *(1)*	20-Pin PLCC	32-Pin TQFP *(2)*		
`nCASC`	6	12	15	Output	Cascade select output (active low). This output goes low when the address counter has reached its maximum value. When the address counter has reached its maximum value, the configuration device has sent all its configuration data to the FPGA. In a chain of EPC2 or EPC1 devices, the `nCASC` pin of one device is connected to the `nCS` pin of the next device, which permits `DCLK` to clock data from the next EPC2 or EPC1 device in the chain. For single EPC2 or EPC1 devices and the last device in the chain, `nCASC` is left floating. This pin is only available in EPC2 and EPC1 devices, which support data cascading.
`nINIT_CONF`	N/A	13	16	Open-Drain Output	Allows the INIT_CONF JTAG instruction to initiate configuration. The `nINIT_CONF` pin connects to the `nCONFIG` of the FPGA. If multiple EPC2 devices are used to configure a FPGA(s), the `nINIT_CONF` of the first EPC2 pin is tied to the FPGA's `nCONFIG` pin, while subsequent devices' `nINIT_CONF` pins are left floating. The `INIT_CONF` pin has an internal 1-kΩ pull-up resistor that is always active in EPC2 devices. This pin is only available in EPC2 devices.
`TDI`	N/A	11	13	Input	JTAG data input pin. Connect this pin to V_{CC} if the JTAG circuitry is not used. This pin is only available in EPC2 devices.
`TDO`	N/A	1	28	Output	JTAG data output pin. Do not connect this pin if the JTAG circuitry is not used. This pin is only available in EPC2 devices.
`TMS`	N/A	19	25	Input	JTAG mode select pin. Connect this pin to V_{CC} if the JTAG circuitry is not used. This pin is only available in EPC2 devices.
`TCK`	N/A	3	32	Input	JTAG clock pin. Connect this pin to ground if the JTAG circuitry is not used. This pin is only available in EPC2 devices.

Table 5–20. EPC2, EPC1 & EPC1441 Pin Functions During Configuration (Part 4 of 4)

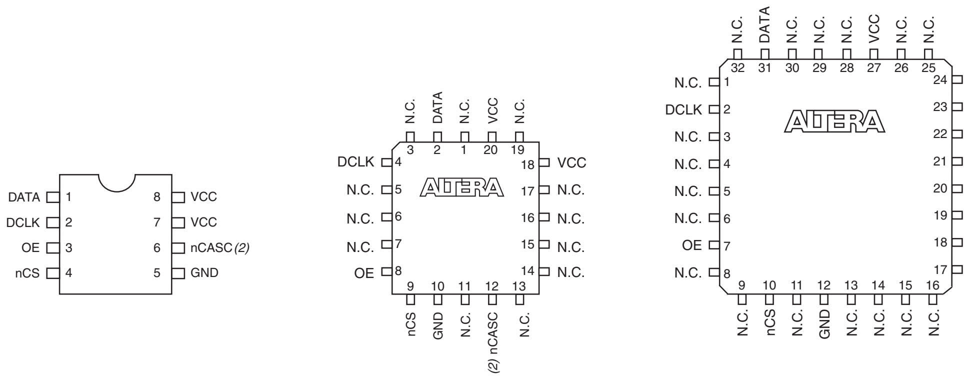
Pin Name	Pin Number			Pin Type	Description
	8-Pin PDIP *(1)*	20-Pin PLCC	32-Pin TQFP *(2)*		
`VCCSEL`	N/A	5	3	Input	Mode select for V_{CC} supply. `VCCSEL` must be connected to ground if the device uses a 5.0-V power supply (VCC = 5.0 V). `VCCSEL` must be connected to VCC if the device uses a 3.3-V power supply (V_{CC} = 3.3 V). This pin is only available in EPC2 devices.
`VPPSEL`	N/A	14	17	Input	Mode select for VPP. VPPSEL must be connected to ground if VPP uses a 5.0-V power supply (V_{PP} = 5.0 V). VPPSEL must be connected to V_{CC} if VPP uses a 3.3-V power supply (V_{PP} = 3.3 V). This pin is only available in EPC2 devices.
`VPP`	N/A	18	23	Power	Programming power pin. For the EPC2 device, this pin is normally tied to VCC. If the EPC2 V_{CC} is 3.3 V, `VPP` can be tied to 5.0 V to improve in-system programming times. For EPC1 and EPC1441 devices, `VPP` must be tied to V_{CC}. This pin is only available in EPC2 devices.
`VCC`	7, 8	20	27	Power	Power pin.
`GND`	5	10	12	Ground	Ground pin. A 0.2-µF decoupling capacitor must be placed between the V_{CC} and GND pins.

Notes to Table 5–20:
(1) This package is available for EPC1 and EPC1441 devices only.
(2) This package is available for EPC2 and EPC1441 devices only.

Package

Figures 5–6 and 5–7 show the configuration device package pin-outs.

Figure 5–6. EPC1, EPC1441, EPC1213, EPC1064 & EPC1064V Package Pin-Out Diagrams Note (1)



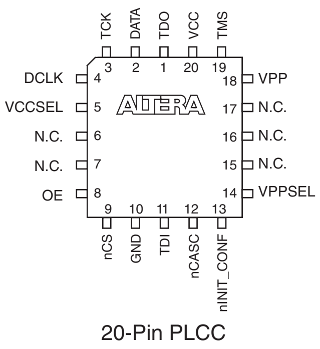
8-Pin PDIP	20-Pin PLCC	32-Pin TQFP
EPC1	EPC1	EPC1441
EPC1441	EPC1441	EPC1064
EPC1213	EPC1213	EPC1064V
EPC1064	EPC1064	
EPC1064V	EPC1064V	

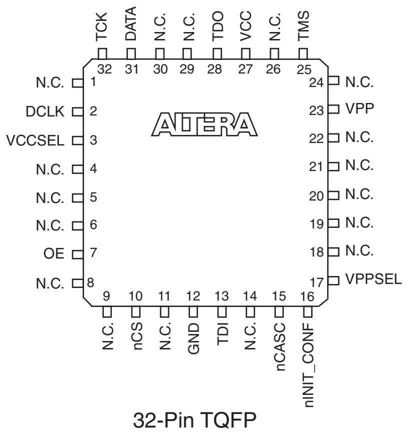
Notes to Figure 5–6:

(1) EPC1 and EPC1441 devices are one-time programmable devices. ISP is not available in these devices.

(2) The `nCASC` pin is available on EPC1 devices, which allows them to be cascaded. On the EPC1441 devices, `nCASC` is a reserved pin and should be left unconnected.

Figure 5–7. EPC2 Package Pin-Out Diagrams





For package outlines and drawings, refer to the *Altera Device Package Information Data Sheet*.

Ordering Codes

Table 5–21. shows the ordering codes for the EPC2, EPC1, and EPC1441 configuration devices.

Table 5–21. Configuration Device Ordering Codes

Device	Package	Temperature	Ordering Code
EPC2	32-pin TQFP	Commercial	EPC2TC32
EPC2	32-pin TQFP	Industrial	EPC2TI32
EPC2	20-pin PLCC	Commercial	EPC2LC20
EPC2	20-pin PLCC	Industrial	EPC2LI20
EPC1	20-pin PLCC	Commercial	EPC1LC20
EPC1	20-pin PLCC	Industrial	EPC1LI20
EPC1	8-pin PDIP	Commercial	EPC1PC8
EPC1	8-pin PDIP	Industrial	EPC1PI8
EPC1441	32-pin TQFP	Commercial	EPC1441TC32
EPC1441	32-pin TQFP	Industrial	EPC1441TI32
EPC1441	20-pin PLCC	Commercial	EPC1441LC20
EPC1441	20-pin PLCC	Industrial	EPC1441LI20
EPC1441	8-pin PDIP	Commercial	EPC1441PC8
EPC1441	8-pin PDIP	Industrial	EPC1441PI8

Section II. Software Settings

Configuration options can be set in the Quartus® II and MAX+PLUS® II development software. You can also specify which configuration file formats Quartus II or MAX+PLUS II generates. This section discusses the configuration options available, how to set these options in the software, and how to generate programming files.

This section includes the following chapters:

- Chapter 6, Device Configuration Options
- Chapter 7, Configuration File Formats

Revision History

The table below shows the revision history for Chapters 6 through 7.

Chapter	Date/Version	Changes Made
6	August 2005, v2.1	● Removed active cross references refering to document outside Chapter 6.
	July 2004, v2.0	● Added Stratix II and Cyclone II device information throughout chapter. ● Updated Default Configuration and Modified Configuration of “auto-restart configuration after error” option in Table 6–1. ● Added paragraph regarding Initiate Configuration After Programming option on page 6–9.
	September 2003, v1.0	Initial Release.
7	August 2005 v2.1	Removed active cross references refering to document outside Chapter 7.
	July 2004, v2.0	Added paragraph regarding difference of **.rpd** from **.rbf** in the Raw Programming Data File (.rpd) section.
	September 2003, v1.0	Initial Release.

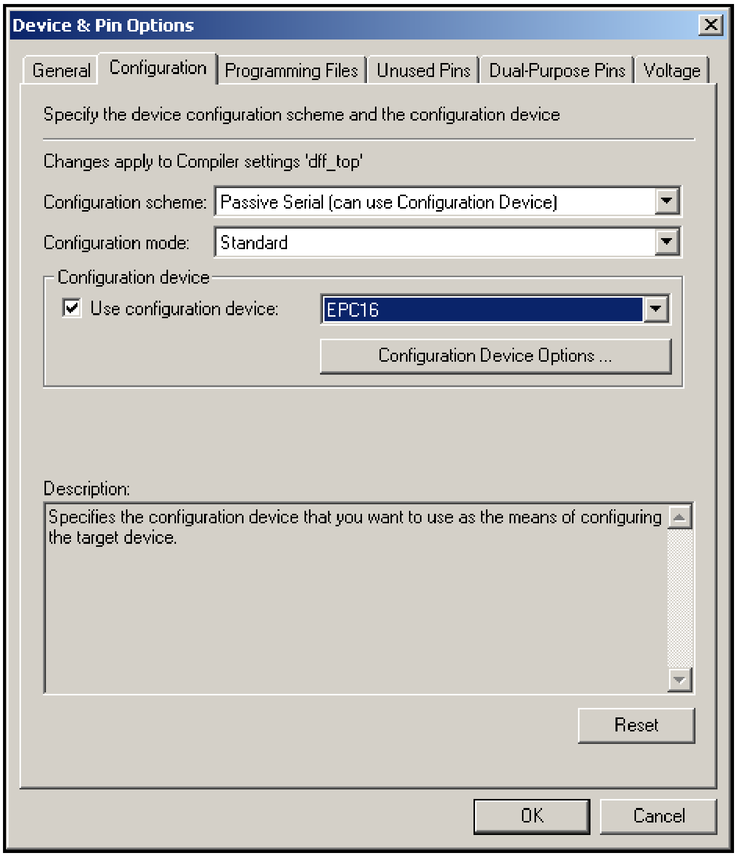
6. Device Configuration Options

CF52006-2.1

Introduction

Device configuration options can be set in the **Device & Pin Options** dialog box. To open this dialog box, choose **Device** (Assignments menu), then click on the **Device & Pin Options...** radio button. You can specify your configuration scheme, configuration mode, and your configuration device used (if applicable) in the **Configuration** tab of the **Device & Pin Options** dialog box (Figure 6–1).

Figure 6–1. Configuration Dialog Box



Specify the device configuration scheme and the configuration device

Changes apply to Compiler settings 'dff_top'

Configuration scheme: Passive Serial (can use Configuration Device)

Configuration mode: Standard

Configuration device

☑ Use configuration device: EPC16

Description:

Specifies the configuration device that you want to use as the means of configuring the target device.

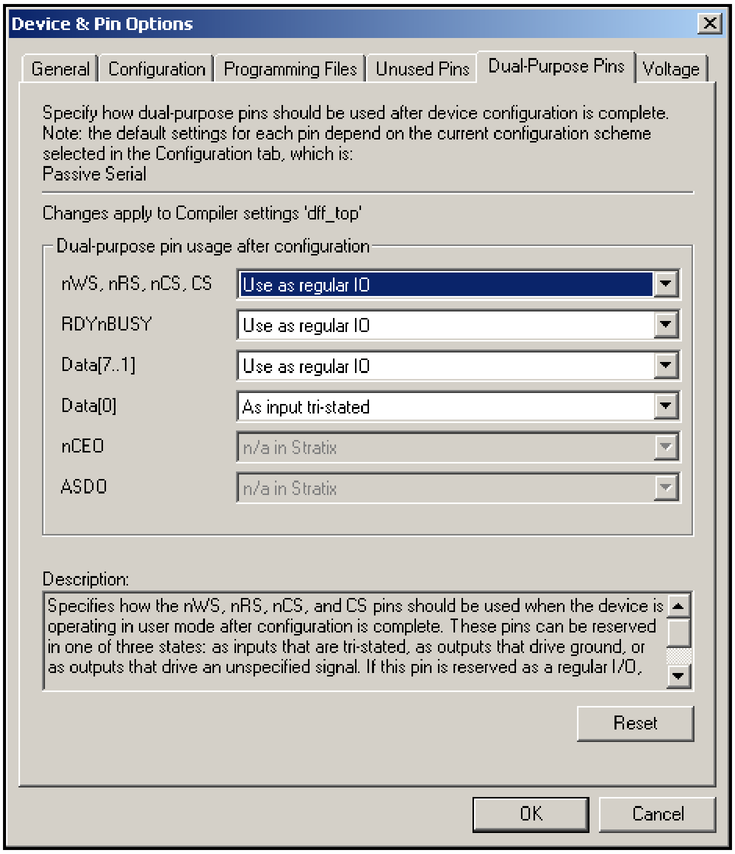
The **Configuration scheme** drop-down list will change with the chosen device family to only show the configuration schemes supported by that device family. The **Configuration mode** selection is only available for devices that support remote and local update, such as Stratix® and Stratix GX devices. If you are not using remote or local update, you should select **Standard** as your configuration mode. For devices that do not support remote or local update, the **Configuration mode** selection will be greyed out.

If you are using a configuration device, turn on **Use configuration device** and specify which configuration device you are using. Choosing the configuration device will direct the Quartus® II compiler to generate the appropriate programming object file (**.pof**). The **Use configuration device** drop-down list will change with the chosen device family to only show the configuration devices that can be used to configure the target device family. If you choose **Auto** as the configuration device, the compiler will automatically choose the smallest density configuration device that fits your design and the **Configuration Device Options…** radio button will be greyed out.

For more information about configuration device options, refer to the appropriate configuration device data sheet.

You can specify how dual-purpose pins should be used after FPGA configuration is complete through the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box. Figure 6–2 shows the **Dual-Purpose Pins** tab for a design that targets a Stratix device and is being configured through PS mode.

Figure 6–2. Dual-Purpose Pins Dialog Box



Specify how dual-purpose pins should be used after device configuration is complete. Note: the default settings for each pin depend on the current configuration scheme selected in the Configuration tab, which is:
Passive Serial

Changes apply to Compiler settings 'dff_top'

Dual-purpose pin usage after configuration

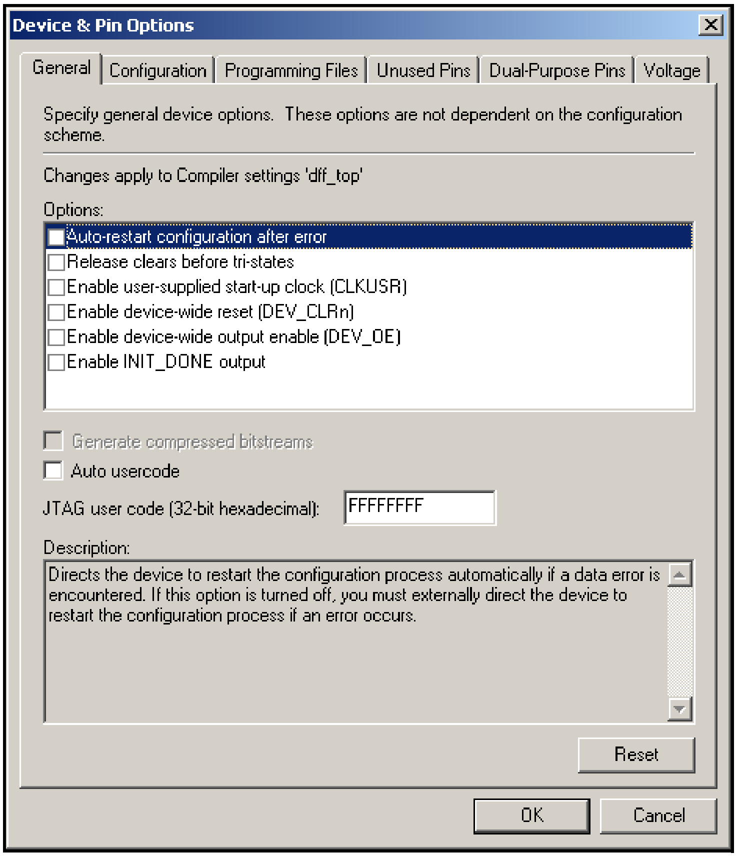
nWS, nRS, nCS, CS	Use as regular IO
RDYnBUSY	Use as regular IO
Data[7..1]	Use as regular IO
Data[0]	As input tri-stated
nCEO	n/a in Stratix
ASDO	n/a in Stratix

Description:
Specifies how the nWS, nRS, nCS, and CS pins should be used when the device is operating in user mode after configuration is complete. These pins can be reserved in one of three states: as inputs that are tri-stated, as outputs that drive ground, or as outputs that drive an unspecified signal. If this pin is reserved as a regular I/O,

The drop-down lists will be greyed-out if the pins are not available in the target device family. For the pins that are available, they can be used in one of four states after configuration: as a regular I/O pin, as inputs that are tri-stated, as outputs that drive ground, or as outputs that drive an unspecified signal. If the pin is not used in the mode, the drop-down list will default to the **Use as regular IO** choice. For example, in PS mode, `DATA[7..1]` are not used, therefore the default usage after configuration for these pins is as a regular I/O pin. If the pin is reserved as a regular I/O, the pin can be used as a regular user I/O after configuration.

You can set device options in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box (see Figure 6–3).

Figure 6–3. Configuration Options Dialog Box



Specify general device options. These options are not dependent on the configuration scheme.

Changes apply to Compiler settings 'dff_top'

Options:

- ☐ Auto-restart configuration after error
- ☐ Release clears before tri-states
- ☐ Enable user-supplied start-up clock (CLKUSR)
- ☐ Enable device-wide reset (DEV_CLRn)
- ☐ Enable device-wide output enable (DEV_OE)
- ☐ Enable INIT_DONE output

☐ Generate compressed bitstreams

☐ Auto usercode

JTAG user code (32-bit hexadecimal): FFFFFFFF

Description:

Directs the device to restart the configuration process automatically if a data error is encountered. If this option is turned off, you must externally direct the device to restart the configuration process if an error occurs.

You can set device options in the MAX+PLUS® II development software by choosing **Global Project Device Options** (Assign menu). Table 6–1 summarizes each of these options.

Table 6–1. Configuration Options (Part 1 of 5)

Device Option	Option Usage	Default Configuration (Option Off)	Modified Configuration (Option On)
Auto-restart configuration after error	When a configuration error occurs, the FPGA drives `nSTATUS` low, which resets itself internally. The FPGA will release its `nSTATUS` pin after a reset time-out period. The `nSTATUS` pin is then pulled to V_{CC} by a pull-up resistor, indicating that reconfiguration can begin. You can choose whether reconfiguration is started automatically or manually (by toggling the `nCONFIG` pin).	Reconfiguration starts automatically and the system does not need to pulse `nCONFIG`. After the FPGA releases `nSTATUS` and it is pulled to V_{CC} by a pull-up resistor, reconfiguration can begin. In passive configuration schemes that use a configuration device, the FPGA's `nSTATUS` pin is tied to the configuration device's `OE` pin. Hence, the `nSTATUS` reset pulse also resets the configuration device automatically. Once the FPGA releases `nSTATUS` and the configuration device releases its `OE` pin (which is pulled high), reconfiguration begins. For more information on how long the reset time-out period is, see the appropriate device family chapters.	The configuration process stops and to start reconfiguration the system must pulse `nCONFIG` from high-to-low and back high.
Release clears before tri-states	During configuration, the device I/O pins are tri-stated. During initialization, you can choose the order for releasing the tri-states and clearing the registers.	The device releases the tri-states on its I/O pins before releasing the clear signal on its registers.	The device releases the clear signals on its registers before releasing the tri-states. This option allows the design to operate before the device drives out, so all outputs do not start up low.

Table 6–1. Configuration Options (Part 2 of 5)

Device Option	Option Usage	Default Configuration (Option Off)	Modified Configuration (Option On)
Enable user-supplied start-up clock (`CLKUSR`) (Stratix series, Cyclone series, APEX™ II, APEX 20K, and Mercury™ devices only).	This option allows you to select which clock source is used for initialization, either the internal oscillator or external clocks provided on the `CLKUSR` pin.	The device's internal oscillator (typically 10 MHz) supplies the initialization clock and the FPGA will take care to provide itself with enough clock cycles for proper initialization. The `CLKUSR` pin is available as a user I/O pin.	The initialization clock must be provided on the `CLKUSR` pin. This clock can synchronize the initialization of multiple devices. The clock should be supplied when the last data byte is transferred. Supplying a clock on `CLKUSR` will not affect the configuration process. For more information on how many clock cycles are needed to properly initialize a device, see the appropriate device family chapters.
Enable user-supplied start-up clock (`CLKUSR`) (ACEX 1K, FLEX 10K, and FLEX 6000 devices only.)	This option allows you to select which clock source is used for initialization, either external clocks provided on the `CLKUSR` pin or on the `DCLK` pin.	In PS and PPS schemes, the internal oscillator is disabled. Thus, external circuitry, such as a configuration device or microprocessor, must provide the initialization clock on the `DCLK` pin. Programming files generated by the Quartus II or MAX+PLUS II software already have these initialization clock cycles included in the file. In the PPA and PSA configuration schemes, the device's internal oscillator (typically 10 MHz) supplies the initialization clock and the FPGA will take care to provide itself with enough clock cycles for proper initialization. The `CLKUSR` pin is available as a user I/O pin.	The initialization clock must be provided on the `CLKUSR` pin. This clock can synchronize the initialization of multiple devices. The clock should be supplied when the last data byte is transferred. Supplying a clock on `CLKUSR` will not affect the configuration process. For more information on how many clock cycles are needed to properly initialize a device, see the appropriate device family chapters.

Table 6–1. Configuration Options (Part 3 of 5)

Device Option	Option Usage	Default Configuration (Option Off)	Modified Configuration (Option On)
Enable device-wide reset (`DEV_CLRn`)	Enables a single pin, `DEV_CLRn`, to reset all device registers.	Chip-wide reset is not enabled. The `DEV_CLRn` pin is available as a user I/O pin.	Chip-wide reset is enabled for all registers in the device. All registers are cleared when the `DEV_CLRn` pin is driven low. The `DEV_CLRn` pin cannot be used to clear only some of the registers; every device register is affected by this global signal.
Enable device-wide output enable (`DEV_OE`)	Enables a single pin, `DEV_OE`, to control all device tri-states.	Chip-wide output enable is not enabled. The `DEV_OE` pin is available as a user I/O pin.	Chip-wide output enable is enabled for all device tri-states. After configuration, all user I/O pins are tri-stated when `DEV_OE` is low. The `DEV_OE` pin cannot be used to tri-state only some of the output pins; every output pin is affected by this global signal.

Table 6–1. Configuration Options (Part 4 of 5)

Device Option	Option Usage	Default Configuration (Option Off)	Modified Configuration (Option On)
Enable `INIT_DONE` output	Enables the `INIT_DONE` pin, which signals the end of initialization and the start of user-mode with a low-to-high transition.	The `INIT_DONE` signal is not available. The `INIT_DONE` pin is available as a user I/O pin.	The `INIT_DONE` signal is available on the open-drain `INIT_DONE` pin. When `nCONFIG` is low and during the beginning of configuration, the `INIT_DONE` pin will be high due to an external pull-up. Once the option bit to enable `INIT_DONE` is programmed into the device (during the first frame of configuration data), the `INIT_DONE` pin will go low. When initialization is complete, the `INIT_DONE` pin will be released and pulled high. This low-to-high transition signals the FPGA has entered user mode. For more information on the value of the external pull-up resistor, see the appropriate device family chapters.
Enable JTAG BST support (FLEX 6000 devices only.)	Enables post-configuration JTAG boundary-scan testing (BST) support in FLEX® 6000 devices.	JTAG BST can be performed before configuration; however, it cannot be performed during or after configuration. During JTAG BST, `nCONFIG` must be held low.	JTAG BST can be performed before or after device configuration via the four JTAG pins (`TDI`, `TDO`, `TMS`, and `TCK`); however, it cannot be performed during configuration. When JTAG boundary-scan testing is performed before device configuration, `nCONFIG` must be held low.

Table 6–1. Configuration Options (Part 5 of 5)

Device Option	Option Usage	Default Configuration (Option Off)	Modified Configuration (Option On)
Generate compressed bitstreams (Stratix II and Cyclone series devices only)	Enables Stratix II and Cyclone series FPGAs to receive compressed configuration bitstreams in AS and PS configuration schemes.	The Quartus II software generates uncompressed programming files and Stratix II and Cyclone series FPGAs do not decompress the configuration data.	The Quartus II software generates compressed programming files and Stratix II and Cyclone series FPGAs decompress the bitstream during configuration.
Auto usercode (Not available in FLEX 6000 devices.)	Allows you to program a 32-bit user electronic signature into the device during programming (typically for design version control). When the USERCODE instruction is loaded into the device, you can shift the signature out of the device.	If this option is off, the JTAG user code option is available and you can specify a 32-bit hexadecimal number for the target device. The JTAG user code is an extension of the option register. This data can be read with the JTAG USERCODE instruction.	Uses the checksum value from the SRAM Object File (**.sof**) as the JTAG user code. If this option is on, the JTAG user code option is dimmed to indicate that it is not available.

After enhanced configuration and EPC2 device programming you can choose to automatically configure the targeted FPGAs on board. This can be done by selecting the **Initiate Configuration After Programming** option under the **Programmer** section of the **Options** window (**Tools** menu). This option is similar to issuing the INIT_CONF JTAG instruction, which means `nINIT_CONF` of the enhanced configuration or EPC2 devices must be connected to the `nCONFIG` of the FPGA.

For more information on the INIT_CONF JTAG instruction, refer to the *Enhanced Configuration devices Data Sheet* or the *Configuration Devices for SRAM-Based LUT Devices Data Sheet*.

7. Configuration File Formats

CF52007-2.1

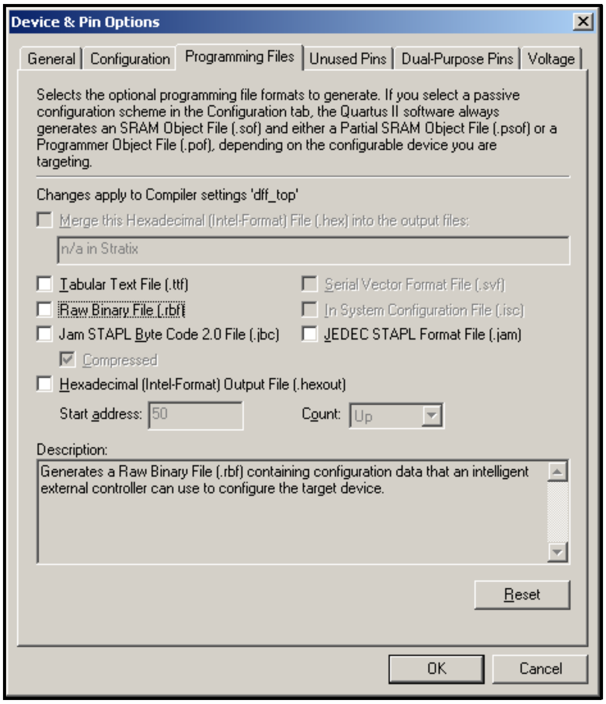
Introduction

Altera's Quartus® II and MAX+PLUS® II development tools can create one or more configuration and programming files to support the configuration schemes discussed in Volume I. When you compile a design in the Quartus II and MAX+PLUS II software for a device that has programming file support, the software will automatically generate a SRAM Object File (**.sof**) and a Programmer Object File (**.pof**) for a configuration device.

Generating Configuration Files

To instruct Quartus II to generate other configuration file formats during compilation, go to **Programming Files** tab of the **Device & Pin Options** dialog box (see Figure 7–1).

Figure 7–1. Programming Files Dialog Box



Selects the optional programming file formats to generate. If you select a passive configuration scheme in the Configuration tab, the Quartus II software always generates an SRAM Object File (.sof) and either a Partial SRAM Object File (.psof) or a Programmer Object File (.pof), depending on the configurable device you are targeting.

Changes apply to Compiler settings 'dff_top'

☐ Merge this Hexadecimal (Intel-Format) File (.hex) into the output files:

n/a in Stratix

☐ Tabular Text File (.ttf)
☐ Serial Vector Format File (.svf)
☐ Raw Binary File (.rbf)
☐ In System Configuration File (.isc)
☐ Jam STAPL Byte Code 2.0 File (.jbc)
☐ JEDEC STAPL Format File (.jam)
☑ Compressed
☐ Hexadecimal (Intel-Format) Output File (.hexout)

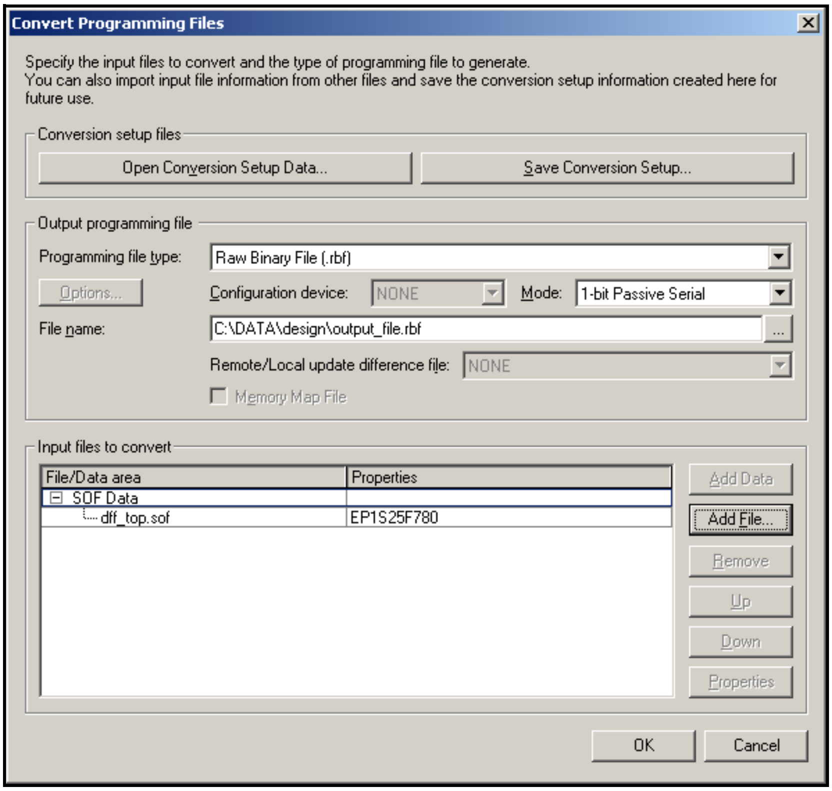
Start address: 50
Count: Up

Description:

Generates a Raw Binary File (.rbf) containing configuration data that an intelligent external controller can use to configure the target device.

You can also convert SOF and POF files through the **Convert Programming Files** window (File menu). Figure 7–2 shows an example of the **Convert Programming Files** dialog box set-up to convert an SOF to a Raw Binary File (**.rbf**).

Figure 7–2. Convert Programming Files Dialog Box


Convert Programming Files

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Programming file type: Raw Binary File (.rbf)
Configuration device: NONE
Mode: 1-bit Passive Serial
File name: C:\DATA\design\output_file.rbf
Remote/Local update difference file: NONE
Memory Map File

File/Data area	Properties
SOF Data	
dff_top.sof	EP1S25F780

When performing multi-device configuration using a configuration device, you must generate the configuration device's POF from each project's SOF. You can combine multiple SOFs using the Convert Programming Files dialog box in the Quartus II software. The following steps explain how to combine multiple SOF files into a POF file(s).

1. Choose **Convert Programming Files...** command (File menu).

2. In the **Programming file type** list, choose **Programming Object File (.pof)**.

3. In the **Configuration device** list, choose the appropriate configuration device.

4. In the **Mode** list, choose the appropriate configuration scheme.

5. You can set configuration devices options by selecting the **Options...** radio button.

6. Specify the name of the output file in the **File name** box.

7. In the **Input files to convert** box, click on **SOF Data**, so that the **Add File…** button becomes active.

8. Click on the **Add File…** button and select the SOF file to be converted. This step can be repeated to combine multiple SOF files into a POF file(s). The order of the SOF files should match the order of the devices in the chain.

9. Click **OK**.

10. When generating multiple POFs for EPC2 or EPC1 devices, the first device's POF file name will be as specified, while the second device's POF file name will have a "_1" extension (e.g., **top_1.pof**)

When performing multi-device configuration using an external host, such as a microprocessor or CPLD, you should generate one combined configuration file from each project's SOF. You can combine multiple SOFs using the **Convert Programming Files** dialog box in the Quartus II software. The following steps explain how to combine multiple SOF files into one configuration file.

1. Choose the **Convert Programming Files...** command (File menu).

2. In the **Programming file** type list, choose the appropriate file format (Hexadecimal (Intel-Format) Output File for SRAM (**.hexout**), RBF, or Tabular Text File (**.ttf**)).

3. In the **Mode** list, choose the appropriate configuration scheme.

4. Specify the name of the output file in the **File name** box.

5. In the **Input files to convert** box, click on **SOF Data**, so that the **Add File…** button becomes active.

6. Click on the **Add File…** button and select the SOF file to be converted. This step can be repeated to combine multiple SOF files into one configuration file. The order of the SOF files (from top to bottom) should match the order of the devices in the chain.

7. Click **OK**.

The following steps explain how to convert a SOF for ACEX® 1K, FLEX® 10K or FLEX 6000 devices using the MAX+PLUS II software.

1. In the MAX+PLUS II Compiler or Programmer, choose the **Convert SRAM Object Files** command (File menu.)

2. In the **Convert SRAM Object Files** dialog box, click on the **Select Programming File…** radio button to specify which SOF file to convert. This step can be repeated to combine multiple SOF files into one configuration file. The order of the SOF files (from top to bottom) should match the order of the devices in the chain.

3. Specify the name of the output file in the **File Name** box.

4. Choose the appropriate configuration file format through the **File Format** pull-down list.

5. Click **OK**.

The following sections give a description of the supported configuration file formats.

SRAM Object File (.sof)

You should use a SOF during PS configuration when the configuration data is downloaded directly to the FPGA using the Quartus II or MAX+PLUS II software with a USB Blaster, MasterBlaster™, ByteBlaster™ II, EthernetBlaster™ or ByteBlasterMV™ cable. The Quartus II and MAX+PLUS II compiler automatically generates the SOF for your design. When using a SOF, the Quartus II or MAX+PLUS II software controls the configuration sequence and automatically inserts the appropriate headers into the configuration data stream. All other configuration files are created from the SOF.

Programmer Object File (.pof)

A POF is used by the Altera® programming hardware to program a configuration device. The Quartus II and MAX+PLUS II compiler automatically generate a POF for your design. For smaller devices (e.g., EPF10K20 devices), multiple SOFs can fit into one configuration device; for larger devices (e.g., APEX 20K devices), multiple configuration devices may be required to hold the configuration data.

Raw Binary File (.rbf)

The RBF is a binary file containing the configuration data. The RBF does not contain byte separators (e.g. commas or carriage returns); it is literally a raw binary file that contains a binary bitstream of configuration data. For example, one byte of RBF data is 8 configured bits `10000101` (85 Hex). Data must be stored so that the least significant bit (LSB) of each data byte is loaded first. The converted image can be stored on a mass storage device. The microprocessor can then read data from the binary file and load it into the FPGA. You can also use the microprocessor to perform real-time conversion during configuration. In the PS configuration schemes, each byte of data should be sent with LSB first. In the FPP, PPS, and PPA configuration schemes, the target device receives its information in parallel from the data bus, a data port on the microprocessor, or some other byte-wide channel.

For more information on creating RBFs, search for "RBF" in Quartus II or MAX+PLUS II Help.

Raw Programming Data File (.rpd)

The RPD File is a binary file containing a binary bitstream of Cyclone™ configuration data. This file is stored in the serial configuration devices in an embedded environment outside the Quartus II software. The Cyclone FPGA can then be configured by using the Active Serial (AS) configuration scheme where the Cyclone FPGA loads the RPD file stored in the serial configuration device. The RPD file size is equal to the memory size of the targeted serial configuration device. A RPD file can only be generated from a POF in the **Convert Programming Files** dialog box (File menu).

The RPD file is different from the RBF file, even for a single device configuration file. In multi-device chains, the RPD file is not the concatenation of the corresponding RBF files. The LSB of each byte in the RPD file should be written to the serial configuration device first.

For more information on creating RPDs, search for "RPD" in Quartus II Help or refer to the *SRunner: An embedded Solution for Serial Configuration Device Programming White Paper.*

Hexadecimal (Intel-Format) File (.hex) or (.hexout)

A HEX File is an ASCII file in the Intel HEX format. Microprocessors or external hosts can use the HEX file to store and transmit configuration data using the configuration schemes supported by microprocessors. This file can also be used by third-party programmers to program Altera's configuration devices.

For more information on creating Hex Files, search for "Hex File" in Quartus II or MAX+PLUS II Help.

Tabular Text File (.ttf)

The TTF is a tabular ASCII file that provides a comma-separated version of the configuration data for the FPP, PPS, PPA, and bit-wide PS configuration schemes. In some applications, the storage device containing the configuration data is neither dedicated to nor connected directly to the target device. For example, a configuration device can also contain executable code for a system (e.g., BIOS routines) and other data. The TTF allows you to include the configuration data as part of the microprocessor's source code using the include or source commands. The microprocessor can access this data from a configuration device or mass-storage device and load it into the target device. A TTF can be imported into nearly any assembly language or high-level language compiler.

For more information on creating TTFs, search for “TTF” in Quartus II or MAX+PLUS II Help.

Serial Bitstream File (.sbf)

An SBF is used in PS schemes to configure FLEX 10K and FLEX 6000 devices in-system with the BitBlaster™ cable.

The BitBlaster is obsolete. SBFs are supported by the MAX+PLUS II software only.

For more information on creating SBFs, search for “SBF” in MAX+PLUS II Help.

Jam File (.jam)

A Jam™ File is an ASCII text file in the Jam device programming language that stores device programming information. These files are used to program, verify, and blank-check one or more devices in the Quartus II or MAX+PLUS II Programmer or in an embedded processor environment.

For more information on creating Jam Files, search for “Jam” in Quartus II or MAX+PLUS II Help.

Jam Byte-Code File (.jbc)

A JBC File is a binary file of a Jam File in a byte-code representation. JBC files store device programming information used to program, verify, and blank-check one or more devices in the Quartus II or MAX+PLUS II Programmer or in an embedded processor environment.

For more information on creating JBC Files, search for “JBC” in Quartus II or MAX+PLUS II Help.

Section III. Advanced Configuration Schemes

This section discusses configuring configuration chains that contain a mixture of Altera® device families, combining different configuration schemes on your board and using a CPLD and flash memory to configure your Altera FPGA. It is recommended that you read the chapters in Volume I for your target device family before reading this section.

This section includes the following chapters:

- Chapter 8, Configuring Mixed Altera FPGA Chains
- Chapter 9, Combining Different Configuration Schemes
- Chapter 10, Using Flash Memory to Configure FPGAs

Revision History

The table below shows the revision history for Chapters 8 through 10.

Chapter	Date/Version	Changes Made
8	August 2005, v2.1	Removed active cross references refering to document outside Chapter 8.
	July 2004, v2.0	Added Stratix II and Cyclone II device information throughout chapter.
	September 2003, v1.0	Initial Release.
9	August 2005, v2.1	Removed active cross references refering to document outside Chapter 9.
	July 2004, v2.0	Added Stratix II and Cyclone II device information throughout chapter.
	September 2003, v1.0	Initial Release.
10	Augut 2005, v2.1	● Removed active cross references refering to document outside Chapter 10.
	July 2004, v2.0	● Removed Intel flash reference design. ● Updated Figure 10–1. ● Removed Flash Memory Content Verification section.
	September 2003, v1.0	Initial Release.

8. Configuring Mixed Altera FPGA Chains

CF52008-2.1

Introduction

A mixture of Stratix® series, Cyclone™ series, APEX™ II, Mercury™, APEX 20K, ACEX® 1K and FLEX® 10K devices can be configured in the same configuration chain, provided that all devices in the chain support the selected configuration method, such as passive serial (PS). This chapter discusses guidelines you should follow when combining different device families in the same configuration chain.

General Guidelines

If any devices in your configuration chain require 10-kΩ pull-up resistors, external 10-kΩ pull-up resistors should be used to support all devices in the chain. The pull-up resistors should be tied to a supply that provides an acceptable input voltage high level for all devices in the chain.

The `DCLK`, `DATA0`, `nCONFIG`, `nSTATUS`, and `CONF_DONE` signals should be tied together for every device in the configuration chain. For concurrent configuration using enhanced configuration devices, each device or chain of devices is fed a separate `DCLK` line, while `DCLK`, `nCONFIG`, `nSTATUS`, and `CONF_DONE` are shared. This ensures that configuration begins and ends at the same time for each device. Additionally, if one device detects an error and pulls `nSTATUS` low, all devices in the chain will reset and restart configuration.

For more information about connecting the configuration control signals together, refer to "*Board Layout Tips & Debugging Techniques*" on page 10–1 in the *Configuration Handbook*.

Any FPGA or configuration device that supports JTAG programming can be placed in the same JTAG chain. For multi-device JTAG chains, external 1-kΩ resistors should be used on the `TCK`, `TDI`, and `TMS` pins. The pull-up resistors on `TDI` and `TMS` should be pulled up to a supply that provides an acceptable input voltage high level for all devices in the chain.

To interface the `TDI` and `TDO` signals of the JTAG pins of Altera devices that have different V_{CCIO} levels, you may need to insert level shifters. You will need to insert level shifters if the `TDI` pin is not tolerant to the voltage driven out by the previous device's `TDO` pin. For example, if the `TDO` of the first device in the JTAG chain is in a back where the V_{CCIO} is set to 5.0-V and the `TDI` of the second device is in a bank where the V_{CCIO} is set to 1.8-V and cannot accept 5.0-V inputs, you need to insert a level shifter in-between the devices' `TDO`-`TDI` interface.

Additionally, you will need to insert level shifters if the TDI pin will not recognize the voltage driven out by the previous device’s TDO pin as an input voltage high level (V_{IH}). For example, if the TDO of the first device in the JTAG chain is in a back where the V_{CCIO} is set to 1.8-V and the TDI of the second device is in a bank where the V_{CCIO} is set to 5.0-V and does not recognize 1.8-V as a logic high level, you need to insert a level shifter in-between the devices' TDO-TDI interface.

Guidelines for Configuration Chains with APEX 20KE Devices

If your configuration chain contains an APEX 20KE device(s), you must follow the APEX 20KE power sequencing requirement as outlined in the *Configuring APEX 20KE and APEX 20KC Devices* Chapter of the Configuration Handbook. The guidelines below should be followed for successful configuration of your configuration chain with APEX 20KE Devices:

- For all configuration schemes, use 10-kΩ pull-up resistors on nCONFIG, nSTATUS, and CONF_DONE.
- For all configuration schemes, ensure nCONFIG is held low upon power-up until both the V_{CCINT} and V_{CCIO} power supplies are stable.
- If you are using a configuration device, nCONFIG must be pulled-up to V_{CCINT} of the APEX 20KE device.
- If you are using the nINIT_CONF pin of the enhanced configuration device or EPC2 device, you need to isolate the 1.8-V V_{CCINT} from the configuration device’s 3.3-V supply by adding a diode between the nCONFIG pin and the nINIT_CONF pin.

9. Combining Different Configuration Schemes

CF52009-2.1

Introduction

This chapter shows you how to configure Altera® FPGAs using multiple configuration schemes on the same board. Combining JTAG configuration with passive serial (PS) or active serial (AS) configuration on your board is useful in the prototyping environment because it allows multiple methods to configure your FPGA. For example, if your production environment calls for PS configuration using a configuration device, you would have to reprogram your configuration device every time you wanted to test a design change in your FPGA. If you include the FPGA in the same JTAG chain as the configuration device, the FPGA can be reconfigure via JTAG without having to reprogram the configuration device.

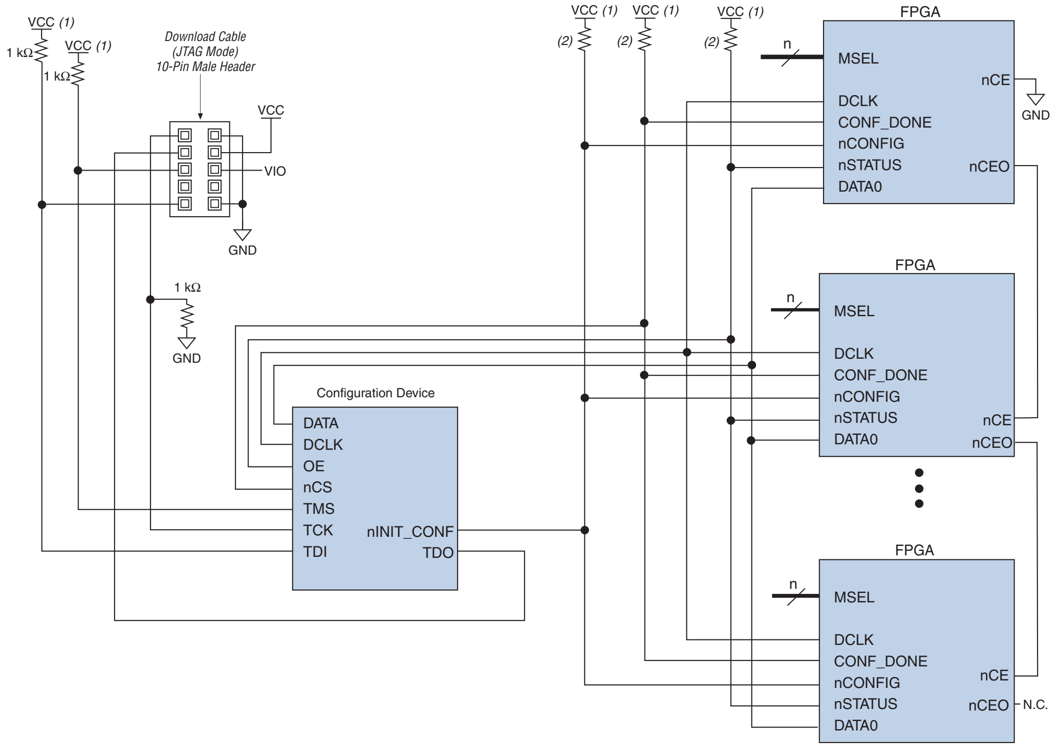
In this chapter, the generic term "download cable" includes the Altera USB Blaster universal serial bus (USB) port download cable, MasterBlaster™ serial/USB communications cable, EthernetBlaster, ByteBlaster™ II parallel port download cable, and the ByteBlasterMV™ parallel port download cable. In this section, the generic term "FPGA" includes Stratix® series, Cyclone™ series, APEX™ II, APEX 20K, Mercury™, ACEX® 1K, and FLEX® 10K devices.

☞ The figures in this chapter will only show the configuration interface connections. For detailed information about pull-up resistor values or other pins on the specific FPGA or configuration device, refer to the appropriate chapter in the Configuration Handbook.

Passive Serial & JTAG

Figure 9–1 shows the configuration interface connections when you are using a download cable to JTAG program a configuration device and the configuration device is used to configure the FPGAs. In Figure 9–1, multiple FPGAs are daisy-chained together and the MSEL pins should be set to select PS as the configuration mode.

Figure 9–1. JTAG Programming of Configuration Device with PS Configuration of FPGA Using a Configuration Device



Notes to Figure 9–1:

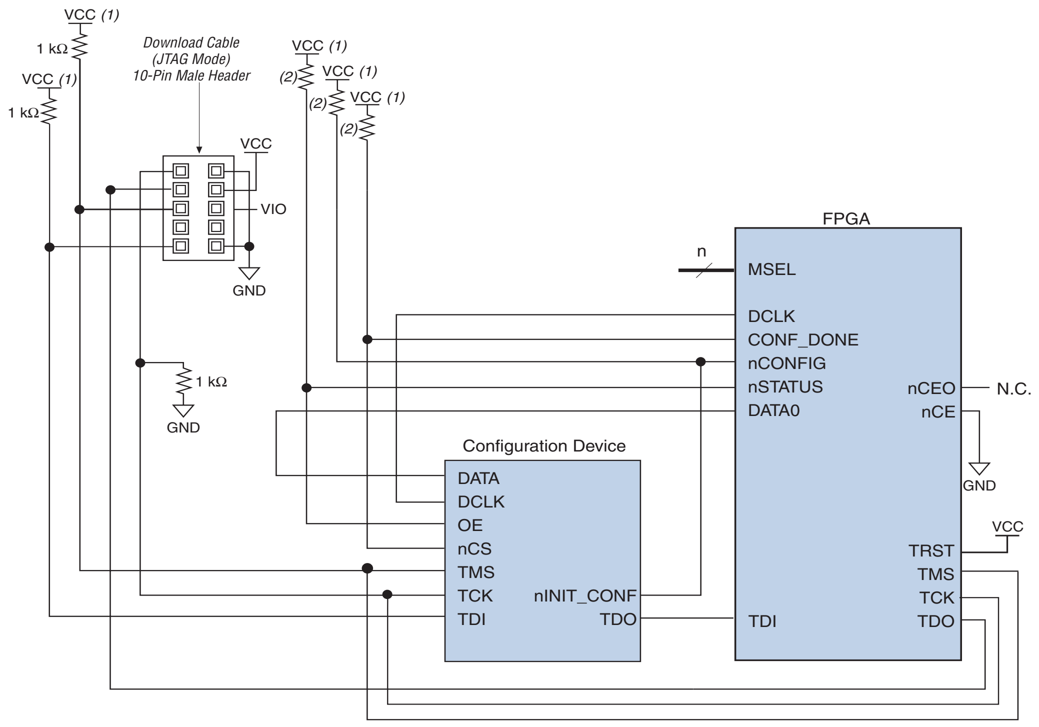
(1) V_{CC} should be connected to the same supply voltage as the configuration device. For APEX 20KE devices, nCONFIG should be pulled up to V_{CCINT}.

(2) If the internal pull-up resistors of the configuration device are used, external pull-up resistors should not be used on these pins.

Figure 9–2 shows the configuration interface connections when the configuration device and the FPGA are in the same JTAG chain. Make sure the TDO signal drives out a high enough voltage to meet the next device's TDI minimum high-level input voltage (V_{IH}). The TDO output will drive out the voltage of the I/O bank's V_{CCIO} where it resides. For example, if the TDO pin resides in an I/O bank whose V_{CCIO} is set to 3.3 V, the TDO pin will drive out 3.3 V. The download cable is used to JTAG program the configuration device and the FPGA. The configuration device is used to configure the FPGA. The MSEL pins should be set to select PS as the configuration mode.

If there is a configuration device on board, upon power-up you should allow the FPGA to finish configuration before attempting JTAG configuration.

Figure 9–2. JTAG Programming of Configuration Device and FPGA with PS Configuration of FPGA Using a Configuration Device



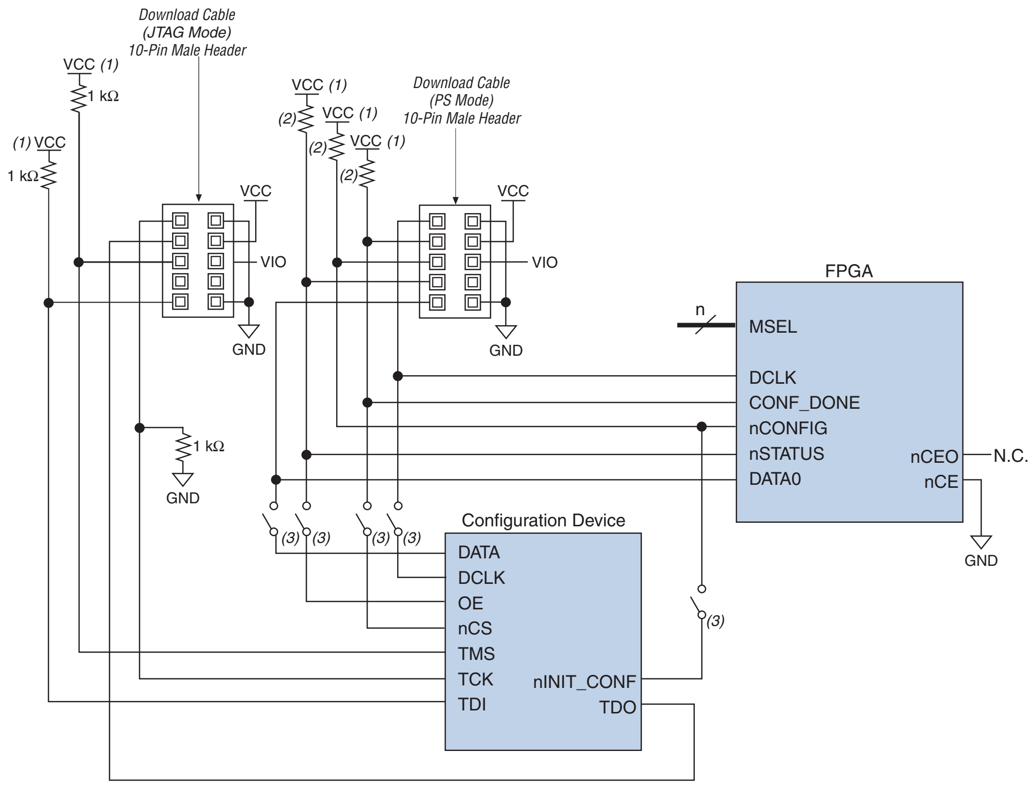
Notes to Figure 9–2:

(1) V_{CC} should be connected to the same supply voltage as the configuration device. For APEX 20KE devices, `nCONFIG` should be pulled up to V_{CCINT}.

(2) If the internal pull-up resistors of the configuration device are used, external pull-up resistors should not be used on these pins.

The download cables can be used in different modes (e.g., JTAG mode or PS mode) and in each mode, the header of the download cable connects to different pins on the FPGA. Therefore, two separate 10-pin headers are required on your board in order to support two different modes of the download cable. Figure 9–3 shows a schematic with two download cables. One download cable is used in JTAG mode to JTAG program the configuration device. The second download cable is used in PS mode to configure the FPGA using PS configuration. The `MSEL` pins should be set to select PS as the configuration mode.

Figure 9–3. JTAG Programming of Configuration Device with PS Configuration of FPGA Using a Configuration Device & Download Cable



Notes to Figure 9–3:

(1) V_{CC} should be connected to the same supply voltage as the configuration device. For APEX 20KE devices, `nCONFIG` should be pulled up to V_{CCINT}.

(2) If the internal pull-up resistors of the configuration device are used, external pull-up resistors should not be used on these pins.

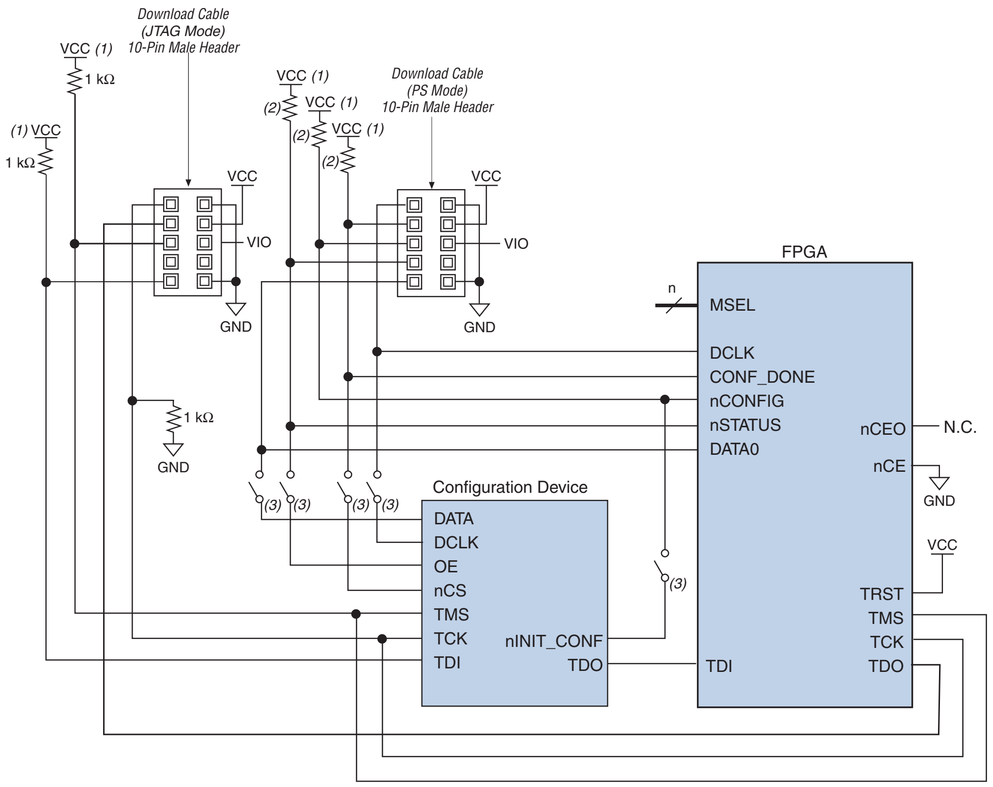
(3) To configure the FPGA with a download cable, you should either remove the configuration device from its socket or place a switch on the five common signals between the download cable and the configuration device.

You should not attempt PS configuration with a download cable while a configuration device is connected to an FPGA.

If you try to configure the FPGA using the download cable while the configuration device is connected to the FPGA, the low signals driven on the nSTATUS and CONF_DONE pins will pull the OE and nCS pins of the configuration device low. This will reset the configuration device and cause it to try to configure the FPGA. To perform PS configuration with the download cable, you should either remove the configuration device from its socket when using the download cable, or place a switch on the five common signals between the download cable and the configuration device.

Figure 9–4 shows a schematic which allows configuration of the FPGA with either a PS mode download cable or JTAG mode download cable. Additionally, the FPGA can be configured using the configuration device. One download cable is used in JTAG mode to JTAG program the configuration device and FPGA. In Figure 9–4 the configuration device and FPGA are in the same JTAG chain. Make sure the TDO signal drives out a high enough voltage to meet the next device's TDI minimum high-level input voltage (V_{IH}). The TDO output will drive out the voltage of the I/O bank's V_{CCIO} where it resides. For example, if the TDO pin resides in an I/O bank whose V_{CCIO} is set to 3.3 V, the TDO pin will drive out 3.3 V. The second download cable is used in PS mode to configure the FPGA using PS configuration. The MSEL pins should be set to select PS as the configuration mode.

Figure 9–4. Combining JTAG Programming of Configuration Device & FPGA with PS Configuration of FPGA Using a Configuration Device and Download Cable



Notes to Figure 9–4*:*

(1) V_{CC} should be connected to the same supply voltage as the configuration device. For APEX 20KE devices, `nCONFIG` should be pulled up to V_{CCINT}.

(2) If the internal pull-up resistors of the configuration device are used, external pull-up resistors should not be used on these pins.

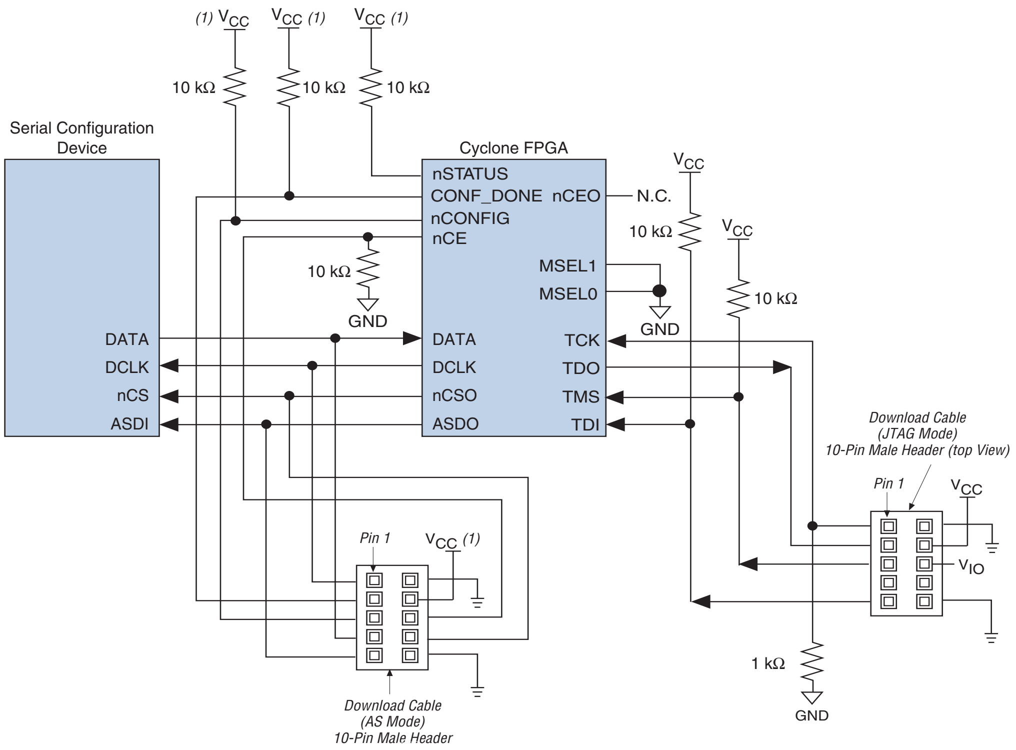
(3) To configure the FPGA with a download cable, you should either remove the configuration device from its socket or place a switch on the five common signals between the download cable and the configuration device.

Figures 9–1 and 9–4 also apply for fast passive parallel (FPP) mode, except `DATA[7..0]` is connected from the enhanced configuration device to the FPGA(s) that supports FPP configuration. The `MSEL` pins need to be set accordingly.

Active Serial & JTAG

For devices that support AS configuration (e.g., Stratix II or Cyclone series devices), you can combine the AS configuration scheme with JTAG-based configuration (see Figure 9–5). This setup uses two 10-pin download cable headers on the board. One download cable is used in JTAG mode to configure the Stratix II or Cyclone series FPGA directly via the JTAG interface. The other download cable is used in AS mode to program the serial configuration device in-system via the AS programming interface. The MSEL pins should be set to select the AS configuration mode. If you try configuring the device using both schemes simultaneously, JTAG configuration takes precedence and AS configuration will be terminated.

Figure 9–5. Combining JTAG Programming of Configuration Device & FPGA with PS Configuration of FPGA Using a Configuration Device & Download Cable



Note to Figure 9–5:

(1) V_{CC} should be connected to 3.3 V.

10. Using Flash Memory to Configure FPGAs

CF52010-2.1

Introduction

As Altera introduces higher-density FPGAs, the configuration bit stream size also increases. As a result, designs require more configuration devices to store the data and configure these devices. As an alternative, flash memory can be used to store configuration data. A flash memory controller is required to read and write to the flash memory and perform configuration. You can use a MAX® 3000A or MAX 7000 device to implement the flash memory controller.

Device Configuration Using Flash Memory & MAX 3000A Devices

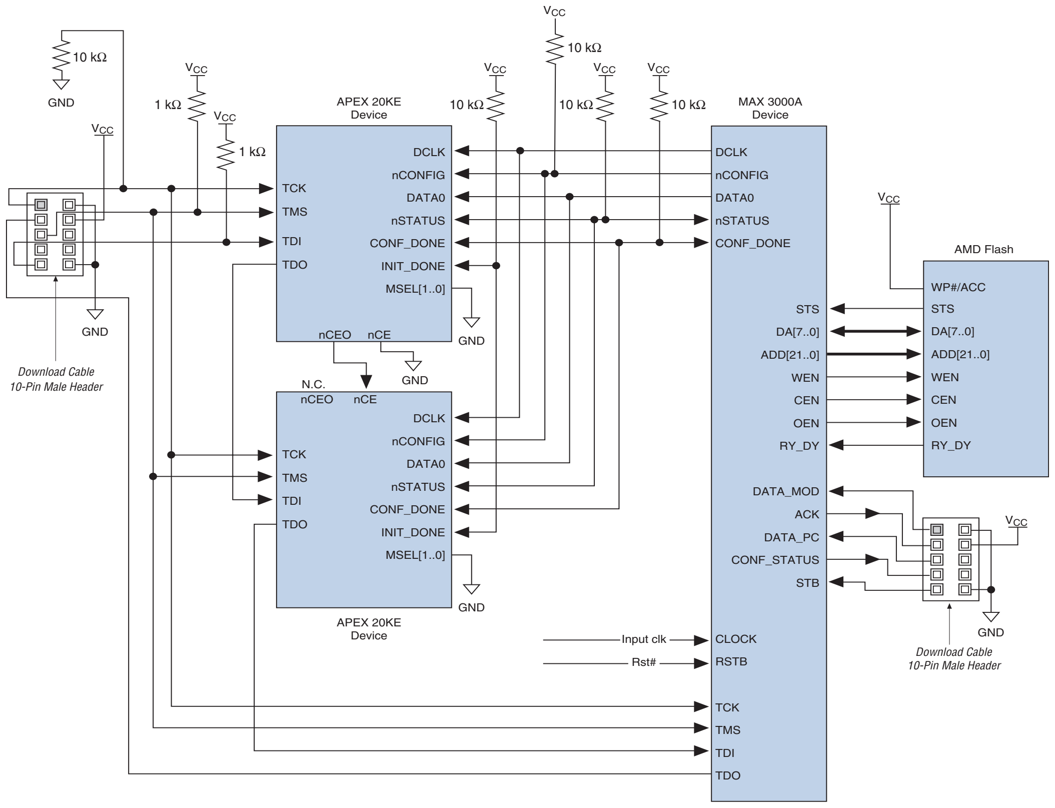
The flash memory controller can interface with a PC or microprocessor to receive configuration data via a parallel port (Figure 10–1). The controller generates a programming command sequence to program the flash memory and extract configuration data to configure FPGAs.

The flash memory controller supports various commands such as:

- Programming the flash memory
- Configuring FPGAs

A reference design that uses the MAX 3000 device is available on the Altera web site. The reference design can be used with an AMD or Fujitsu flash.

Figure 10–1. Configuring an FPGA through Flash Memory & MAX 3000A Controller



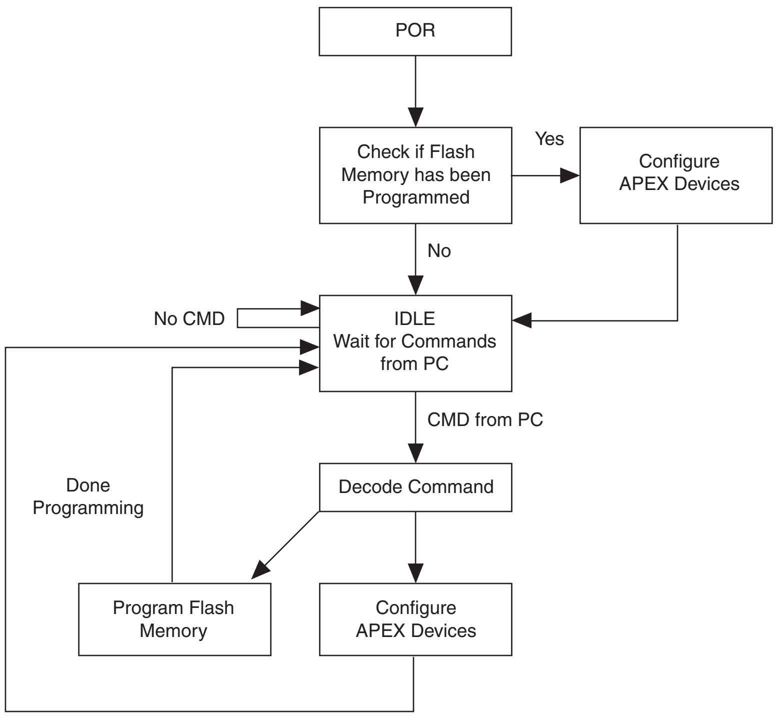
Flash Memory Controller Design Specification

The controller will check to see if the flash memory is programmed successfully after the board powers up. If the flash memory is programmed successfully, then the controller configures the FPGAs. If flash memory is not programmed successfully, then the controller waits for commands from the PC or microprocessor. The receiver decodes the commands it receives from the PC or microprocessor as one of the following:

- Program flash memory
- Configure FPGA

After a command is executed, the controller returns to idle mode and waits for the next command. Figure 10–2 shows the controller state machine.

Figure 10–2. Flash Memory Controller State Machine



Flash Memory Controller Functionality

The controller writes a byte to a special location in the flash memory when it programs the memory. After POR, the controller checks this special location in the flash memory to see if the byte is written there or not.

If the byte is written, then the flash memory has been programmed and the controller can proceed to configuring the FPGAs by reading data from the flash memory. If this byte is not there or the value is not as expected, the controller will go idle and wait to be programmed by the PC or microprocessor.

Getting Data from the PC or Microprocessor

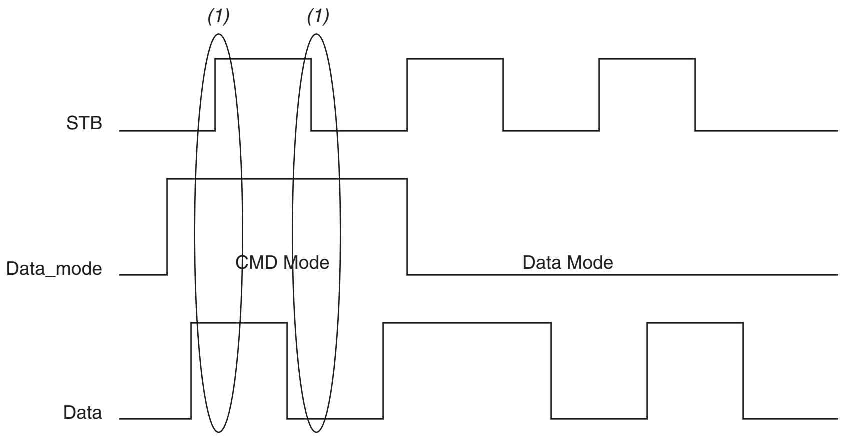
The PC or microprocessor uses the parallel port to interface with the controller. There are two types of signals involved in this connection (see Figure 10–3), a 3-bit input signal from the PC or microprocessor to the controller, and a 2-bit output signal from the controller to the PC or microprocessor. The input signal includes the following three signals:

- STB: Strobe signal from the PC or microprocessor to indicate that the PC or microprocessor's data is valid.
- data_mode: Indicates whether the controller is in command mode or data mode. When data_mode is high, the controller is in command mode; when data_mode is low, the controller is in data mode.
- data: Content of this signal depends on data_mode. It can be data for command mode or data mode.

The output signal contains the following two signals:

- ACK: Acknowledge signal is a handshaking signal from the controller to the PC or microprocessor.
- conf_status: Indicates configuration status.

Figure 10–3. Getting Data from PC or Microprocessor

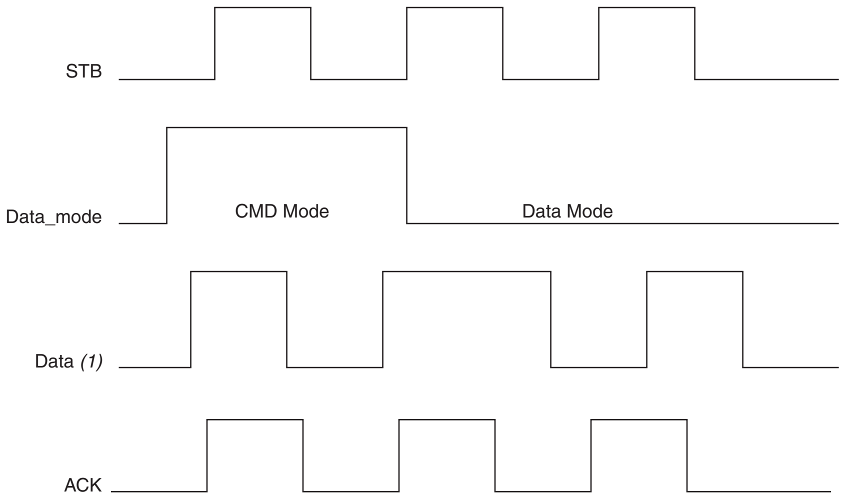


Note to Figure 10–3:

(1) Data is sent on both positive and negative edges of the STB signal.

The controller receives a bit of data or a command from the PC or microprocessor on the rising and falling edges of the STB signal. After receiving this data, the controller will send an acknowledgement signal to the PC or microprocessor to initiate sending of the next bit of data. The acknowledge signal (ACK) should be the same logic level as the last received STB signal. By de-asserting ACK, the controller can stop the PC or microprocessor from sending data. Figure 10–4 shows the STB and ACK relationship.

Figure 10–4. Sending Acknowledge Signal (ACK) to PC or Microprocessor



Note to Figure 10–4:
(1) One bit of data is received at each STB signal edge (both positive and negative).

Programming Flash Memory

After receiving a command from the PC or microprocessor, the controller first erases and then starts programming the flash memory. A separate state machine is required to generate a programming command sequence and programming pulse width.

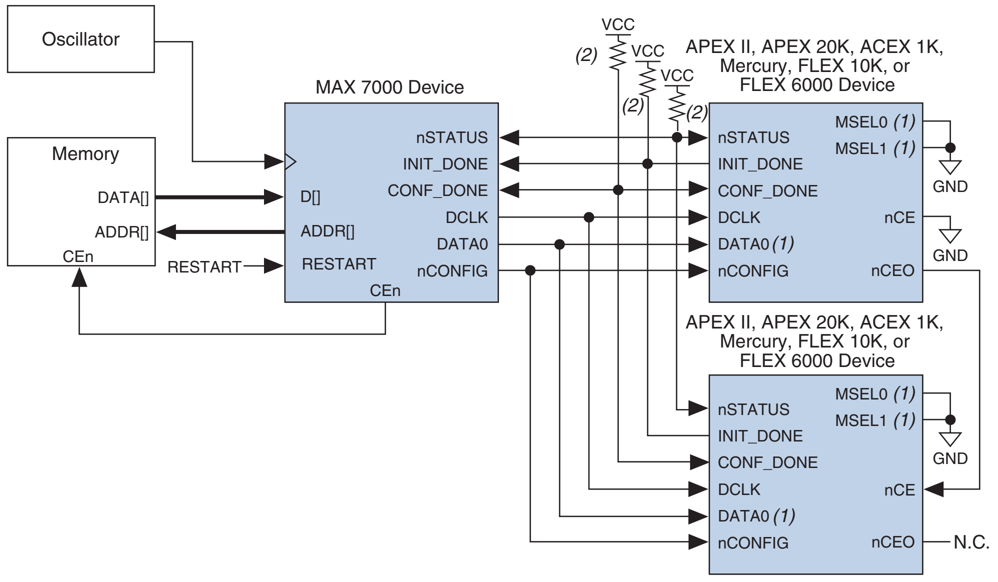
While programming the flash memory, the controller must check if a command (data_mode=1) has been received or not. A command indicates the end of data from the PC or microprocessor, and the controller will exit the Program_Flash_memory state and go into idle mode.

Another state machine is required to read and serialize byte data from the flash memory and generate `DCLK` and `DATA0`. The controller needs to monitor `CONF_DONE` signals from the FPGAs to determine if configuration is complete. When configuration is done, the controller exits the configure state and goes back to idle mode.

Device Configuration Using Flash Memory & MAX 7000 Devices

Figure 10–5 shows the schematic for this configuration scheme with a MAX 7000 device. Two sample design files for the MAX 7000 device (Design File for Configuring APEX™ 20K Devices and Design File for Configuring FLEX® 10K and FLEX 6000 Devices) are available on the Altera web site.

Figure 10–5. Device Configuration Using External Memory & a MAX 7000 Device



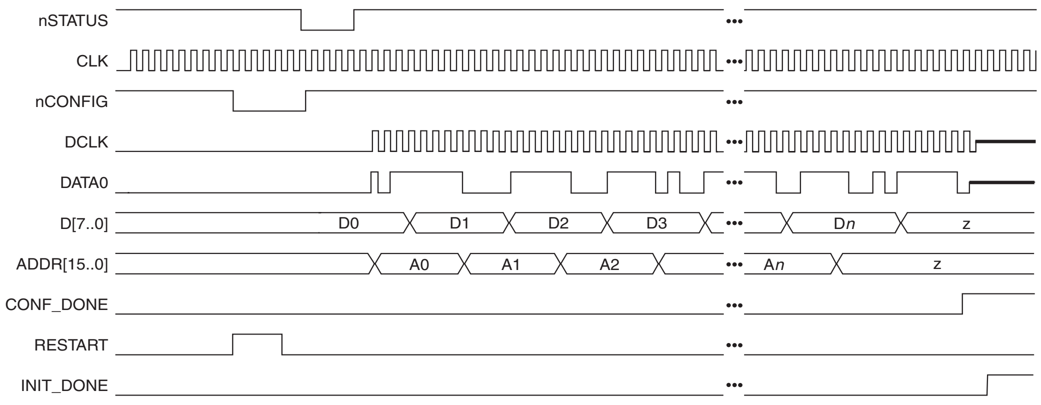
Notes to Figure 10–5:

(1) FLEX 6000 devices have a single `MSEL` pin, which is tied to ground, and its `DATA0` pin is renamed `DATA`.

(2) All pull-up resistors are 1 kΩ On APEX 20KE and APEX 20KC devices, pull-up resistors for `nSTATUS`, `CONF_DONE`, and `INIT_DONE` pins should be 10 kΩ

Figure 10–6 shows the timing waveform for configuring an APEX™ II, APEX 20K, Mercury™, ACEX® 1K, FLEX 10K, or FLEX 6000 device using external memory and a MAX 7000 device.

Figure 10–6. Timing Waveform for Configuration Using External Memory & a MAX 7000 Device



Design Example Using MAX 3000 Devices

A MAX® 3000 device can be used to stream the data from the flash memory into a large FPGA. This configuration technique allows faster configuration times. Since a fixed-frequency oscillator (or any available clock on the system) is used to generate the clock for the configuration, the clock frequency can be as high as 57 MHz (the maximum for an APEX 20KE device).

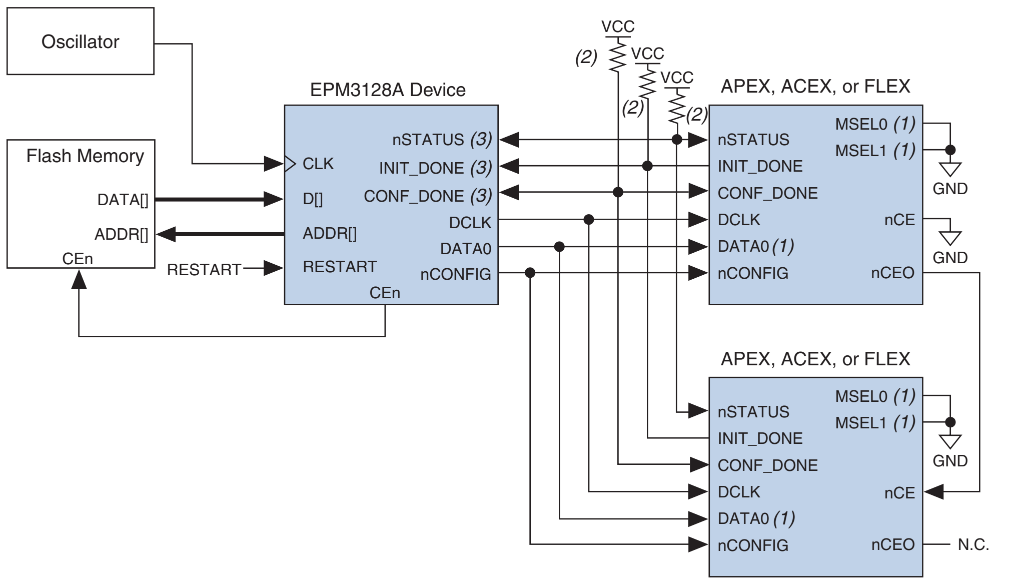
Flash memory is a type of nonvolatile memory that can be used as a data storage device. Flash memory can be erased and reprogrammed in units of memory called blocks.

This section describes how to configure an FPGA with flash memory. By using a MAX 3000 device to configure higher density FPGAs, the flash memory can store configuration data and the MAX 3000 device can serialize and transmit the data to the FPGA. This configuration technique can be used with APEX, ACEX, or FLEX devices.

Configuring FPGAs

Figure 10–7 shows a device that uses an EPM3128A device and flash memory to configure the FPGAs.

Figure 10–7. Device Configuration Using Flash Memory & EPM3128A Device



Notes to Figure 10–7

(1) FLEX 6000 devices have a single `MSEL` pin, which is tied to ground. Additionally, its `DATA0` pin is renamed `DATA`.

(2) Pull-up resistors are 1 kΩ except for APEX 20KE devices. For APEX 20KE devices, pull up resistors are 10 kΩ

(3) The `nSTATUS`, `CONF_DONE`, and `INIT_DONE` pins are open-drain on the APEX, ACEX, and FLEX devices. The corresponding pins on the EPM3128A should also be open_drain.

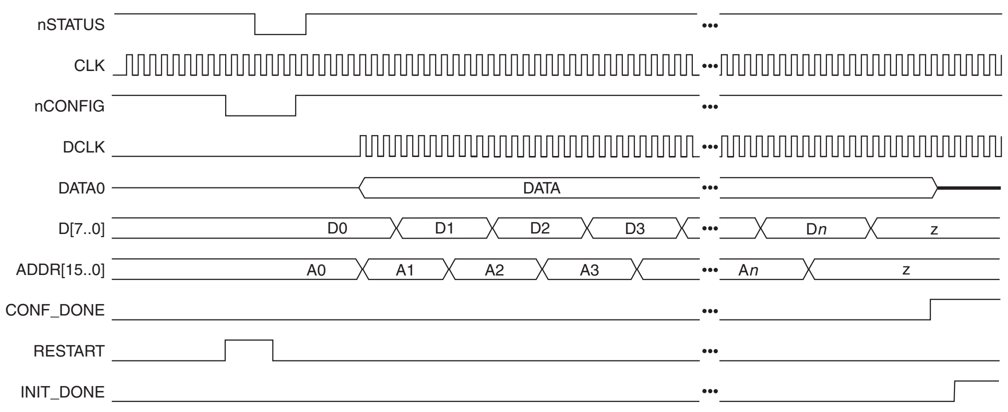
A VHDL design file called **MAXconfig**, shown in the "Configuration Design File" section, allows an EPM3128A device to control the configuration process. The **MAXconfig** design configures the FPGA using the configuration data stored in the attached flash memory. The **MAXconfig** design contains a sequencer and an address generator, which drives the correct data to the FPGA's programming pins. The **MAXconfig** design file is available on the Altera web site at **www.altera.com/document/wp/maxconfig.txt**.

When the **MAXconfig** design is reset, the **MAXconfig** design reads the data from the flash memory, one byte at a time. The **MAXconfig** design then serializes and sends the data to the APEX, ACEX, or FLEX device. The serialized data is sent to the FPGA using the passive serial interface pins such as `DCLK`, `DATA`, `nSTATUS`, `INIT_DONE`, and `nCONFIG`. Since the passive serial mode is used, the flash pins are not directly connected to the APEX, ACEX, or FLEX device.

Flash memory can be programmed prior to being put onto a board with standard programming equipment or it can be programmed in-system by a processor or test equipment. Since different flash memories have different algorithms, consult the flash memory data sheet for programming information.

Figure 10–8 shows a configuration timing waveform of an EPM3128A device downloading data to an APEX, ACEX, or FLEX device.

Figure 10–8. Configuration Timing Waveform



Configuration Design File

This section shows the **MAXconfig** design file that controls the configuration process on APEX, ACEX, or FLEX devices:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity MAXconfig is
port
    (
    clock  : in   std_logic;
    init_done: in std_logic;
    nStatus: in   std_logic;
    D      : in   std_logic_vector(7 downto 0);
    restart: in   std_logic;
    Conf_Done: in std_logic;

    Data0  : out  std_logic;
    Dclk   : out  std_logic;
```

```
   nConfig: bufferstd_logic;

--To increase the size of the memory, change the size of std_logic_vector
for ADDR output and --std_logic_vector signal inc:

   ADDR   : out  std_logic_vector(15 downto 0);
   CEn    : out  std_logic);
-- The polarity of the CEn signal is determined by the type of Flash device
end;

architecture rtl of MAXconfig is

--The following encoding is done in such way that the LSB represents the
nConfig signal:

constant start      :std_logic_vector(2 downto 0) := "000";
constant wait_nCfg_8us:std_logic_vector(2 downto 0) := "100";
constant status     :std_logic_vector(2 downto 0) := "001";
constant wait_40us  :std_logic_vector(2 downto 0) := "101";
constant config     :std_logic_vector(2 downto 0) := "011";
constant init       :std_logic_vector(2 downto 0) := "111";


signal pp           :std_logic_vector(2 downto 0);
signal count        :std_logic_vector(2 downto 0);
signal data0_int, dclk_int:std_logic;
signal inc          :std_logic_vector(15 downto 0);
signal div          :std_logic_vector(2 downto 0);
signal waitd        :std_logic_vector(11 downto 0);
--The width of signal 'waitd' is determined by the frequency. For 57 MHz
(APEX 20KE devices), --'waitd' is 12 bits. For 33 MHz (FLEX 10KE and ACEX
devices) 'waitd' is 11 bits. To calculate --the width of the 'waitd' signal
fordifferent frequencies, calculate the following:
--(multiply tcf2ck * clock frequency)+ 40
--Then convert this value to binary to obtain the width.
--For example, for 33 MHz (FLEX 10KE & ACEX devices), converting 1360 ((40us
* 33MHz)+40=1360)
--to binary code, the 'waitd' is an 11-bit signal. So signal 'waitd' will
be:
--signal waitd      :std_logic_vector(10 downto 0);

begin
--The following process is used to divide the CLOCK:
   PROCESS (clock,restart)
   begin
      if restart = '0' then
         div <= (others => '0');
      else
         IF (clock'EVENT AND clock = '1') THEN
            div <= div + 1;
         end if;
      end if;
   END PROCESS;

   PROCESS (clock,restart)
   begin
      if restart = '0' then
         pp<=start;
         count <= (others => '0');
         inc   <= (others => '0');
         waitd <= (others => '0');
      else
      if clock'event and clock='1' then

--The following test is used to divide the CLOCK. The value compared to
must be such that the
--condition is true at a maximum rate of 57 MHz (tclk = 17.5 ns min) for
APEX 20KE devices
--and at a maximum rate of 33 MHz (tclk=30ns min) for FLEX 10KE or ACEX
devices.

         if (div = 7) then
            case pp is
            when start =>
                     count <= (others => '0');
                     inc   <= (others => '0');
                     waitd <= (others => '0');
                     pp <= wait_nCfg_8us;
--This state is used in order to verify the tcfg timing (nCONFIG low pulse
width).
--Tcfg = 8µs => min= 456 clock cycle of a 57 MHz clock (APEX 20KE devices).
For different --clocks, multiply 8µs to clock frequency. For example, for
33MHz (FLEX 10KE or ACEX devices) this --value is 8*33=264. This clock is
CLOCK divided by the divider -div-.
            when wait_nCfg_8us =>
                     count <= (others => '0');
                      inc   <= (others => '0');
                      waitd <= waitd + 1;
                      if waitd = 456 then
--For 33 MHz FLEX 10KE or ACEX devices this line is: if waitd = 264 then

                  pp <= status;
                     end if;

--This state is used to have nCONFIG high.
            when status =>
               count <= (others => '0');
               inc   <= (others => '0');
               waitd <= (others => '0');
               pp <= wait_40us;

--This state is used to generate the tcf2ck timing (nCONFIG high to first
rising edge on DCLK).
```

```
--Tcf2ck = 40µs min => 2280 clock cycles of a 57MHz (APEX 20KE) clock. This
clock is CLOCK
--divide by the divider -div-
--Tcf2ck = 40µs min => 1320 clock cycles of a 33MHz (FLEX 10KE/ACEX) clock.
This clock is CLOCK --divided by the divider -div-)
--For any other clock frequency, multiply tcf2ck * clock frequency.

             when wait_40us =>
                         count <= (others => '0');
                         inc   <= (others => '0');
                         waitd <= waitd + 1;
                        if waitd = 2280 then
--For 33 MHz (FLEX 10KE or ACEX devices), this line is:    if waitd = 1320
then

                        pp <= config;
                end if;

--This state is used to increment the memory address. In the same state when
--the Conf_Done is high clock cycles are added in order to have the
initialization completed.

             when config =>
                count <= count + 1;
                if Conf_Done='1' then
                   waitd <= waitd + 1;
                end if;
                if count=7 then
                   inc <= inc + 1;
                end if;
                if waitd = 2320 then
--Modification: Add 40 clock cycles. For APEX 20KE devices, it is
2280+40=2320
--For FLEX 10KE and ACEX devices, it is 1320+40=1360. This line becomes:
if waitd= 1360 then

                pp<= init;
                end if;

             when init =>
                count <= (others => '0');
                inc   <= (others => '0');
                waitd <= (others => '0');
                if nStatus = '0' then
                   pp <= start;
                else
                   pp <= init;
                end if;

             when others =>
                pp <= start;
             end case;
```

```
          else
             pp <= pp;
             inc <= inc;
             count <= count;
          end if;
       end if;
    end if;
    end PROCESS;

    dclk_int <= div(2) when pp=config else '0';

--The following process is used to serialize the data byte :
    PROCESS (count,D,pp)
    begin
       if pp=config then
             case count is
             when "000" => data0_int <= D(0);
             when "001" => data0_int <= D(1);
             when "010" => data0_int <= D(2);
             when "011" => data0_int <= D(3);
             when "100" => data0_int <= D(4);
             when "101" => data0_int <= D(5);
             when "110" => data0_int <= D(6);
             when "111" => data0_int <= D(7);
             when others => null;
             end case;
       else
             data0_int <= '0';
       end if;
    end PROCESS;

    nConfig <= pp(0);
    CEn <= not nconfig;
    Dclk  <= '0' when pp(1)='0' else dclk_int;
    Data0 <= '0' when pp(1)='0' else data0_int;
    ADDR <= inc;
    end;
```

Conclusion

Altera provides high-density FPGAs that require larger configuration files. By using a flash memory device and an EPM3128A device in a design, a FPGA can be quickly configured.

Section IV. Board Layout Tips & Debugging Techniques

This section provides information for board layout designers to successfully layout their boards for successful FPGA configuration. This section also provides suggestions for debugging configuration problems, and includes the following chapter:

- Chapter 11, Debugging Configuration Problems

Revision History

The table below shows the revision history for Chapter 11.

Chapter	Date/Version	Changes Made
11	August 2005, v2.1	Removed active cross references refering to document outside Chapter 11.t
11	July 2004, v2.0	Renamed debugging tool to troubleshooter on page 11–6.
	September 2003, v1.0	Initial Release.

11. Debugging Configuration Problems

CF52011-2.1

Introduction

This section provides information about how Altera® FPGAs ensure configuration reliability and suggestions on laying out the configuration interface on your board to avoid configuration problems. The last section provides suggestions on debugging configuration issues.

Configuration Reliability

The architecture of Altera FPGAs have been designed to minimize the effects of power supply and data noise in a system, and to ensure that the configuration data is not corrupted during configuration or normal user-mode operation. A number of circuit design features are provided to ensure the highest possible level of reliability from this SRAM technology.

Cyclic redundancy code (CRC) circuitry is used to validate each configuration data frame (sequence of data bits) as it is loaded into the target device. If the CRC calculated by the device does not match the CRC stored in the data frame, the configuration process is halted, and the FPGA drives the `nSTATUS` pin low to indicate an error has occurred. CRC circuitry ensures that noisy systems will not cause errors that yield an incorrect or incomplete configuration.

The device architecture also provides a very high level of reliability in low-voltage brown-out conditions. The device's SRAM cells require a certain voltage level to maintain accurate data. This voltage threshold is significantly lower than the voltage required to activate the device's POR circuitry. Therefore, the target device stops operating if the V_{CC} starts to fail, and indicates an operation error by driving the `nSTATUS` pin low. The device must then be reconfigured before it can resume proper operation as a logic device. In configuration schemes where `nCONFIG` is tied to V_{CC}, reconfiguration begins as soon as V_{CC} returns to an acceptable level. The low pulse on `nSTATUS` resets the configuration device by driving `OE` low. In configuration schemes using an external host, the system must start the reconfiguration process by driving `nCONFIG` high after the power supply returns to the minimum operating voltage.

These device features ensure that Altera FPGAs have the highest possible reliability in a wide variety of environments, and provide the same high level of system reliability that exists in other Altera PLDs.

Board Layout Tips

Even though the DCLK signal (used in synchronous configuration schemes) is typically a low frequency signal, it drives edge-triggered pins on the Altera FPGA. Therefore, any overshoot, undershoot, ringing, or other noise can affect configuration. When designing the board, lay out the DCLK trace using the same techniques used to lay out a clock line. The same applies for the TCK pin. Since DATA set-up and hold times are in relation to the DCLK signal, make sure these traces are laid out accordingly.

When configuring multiple devices in a configuration chain, Altera recommends that the DCLK, DATA0, (DATA[7..0]), nCONFIG, nSTATUS, and CONF_DONE signals are tied together for every device in the configuration chain. This ensures that configuration begins and ends at the same time for each device. Additionally, if one device detects an error and pulls nSTATUS low, all devices in the chain will reset and restart configuration. For debugging purposes in your prototyping environment, it may be useful for each device to have its own pull-up to V_{CC} for the nSTATUS and CONF_DONE signals. This will allow you to determine which device is signaling an error during configuration by monitoring each nSTATUS line individually or if one device is not releasing its CONF_DONE pin.

In multi-device configuration chains, the configuration signals may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the DCLK and DATA lines are buffered for every fourth device. For multi-device JTAG chains, ensure that the TCK, TDI, and TMS lines are buffered for every device.

When using a configuration device, it is important to realize that after the configuration device sends all its configuration data, it will wait a limited amount of time for its nCS pin (tie to the FPGA's CONF_DONE pin) to reach a logic high. Enhanced configuration devices wait for 64 DCLK cycles after the last configuration bit was sent for CONF_DONE to reach a high state. EPC2 devices wait for 16 DCLK cycles. If the configuration device does not see a logic high on nCS after sending all its configuration data, it will signal an error by driving its OE pin (tied to the FPGA's nSTATUS pin) low. Therefore, if there is a long trace length between the nCS and CONF_DONE pin this could cause a configuration error since the added capacitance of a longer trace contributes to a slower rise time on the CONF_DONE signal.

Debugging Suggestions

If you are experiencing problems configuring your Altera FPGA, there are a number of actions you can take to try to identify your problem. If you have not already done so, you should read the appropriate device family chapters.

The following sections provide some suggestions to try if you are encountering configuration problems.

All Configuration Schemes

- Ensure the configuration file you are using is for the target device on your board. Double check that the SOF used to create the configuration file is for the device on your board by loading the SOF in the Quartus® II programmer and noting the Device column the programmer reports.
- Ensure your board is receiving adequate power to power the FPGA V_{CCINT} and the I/O banks where the configuration and JTAG pins reside.
- Double-check that all configuration pins are properly connected as recommended in the appropriate device family chapter. You should check the connections of these pins by probing at the pins of the device, or vias under the BGA package. (if possible, not on board traces). Specifically, ensure the `MSEL` and `nCE` pins are not left floating and are connected as indicated in the appropriate device family chapter. The `nCONFIG`, `nSTATUS`, and `CONF_DONE` signals require pull-up resistors to V_{CC} (either internal or external pull-up resistors).
- If you are not using the JTAG interface, make sure the JTAG pins on the FPGA are not left floating and are connected to a stable level as indicated in the Configuration Pins tables. Because JTAG configuration takes precedence over all other configuration methods, these pins should not be floating or toggling during configuration.
- Try to configure another device on a different board to determine if it is a universal problem which is seen on all boards or on only one device. If another board is not available, you can swap out the device with another device. If you see the problem with only one device, this points to a problem that is device specific. If you see the problem on multiple devices, this indicates that the problem is with the board or with the configuration set up.
- Probe the `DCLK` signal to ensure it is a clean signal with no overshoot, undershoot or ringing. A noisy `DCLK` signal could affect configuration and cause a CRC error. For a chain of FPGAs, you should probe each device in the chain as close to the `DCLK` pin as possible. Noise on any of these pins could cause configuration to fail for the whole chain.

- To check if the FPGA has started accepting configuration data, you can monitor the INIT_DONE pin. The INIT_DONE pin is an optional pin and can be turned on in the Quartus II software through the Enable INIT_DONE output option. The INIT_DONE pin is an open-drain output and requires an external pull-up to V_{CC}. Therefore, when nCONFIG is low and during the beginning of configuration, the INIT_DONE will be at a logic high level. After the option bit to enable the INIT_DONE pin is programmed into the FPGA (during the first frame of configuration data), the INIT_DONE pin will go low. The transition of INIT_DONE from high to low signals that the FPGA has indeed begun configuration and started to accept configuration data. If the INIT_DONE pin remains high, the FPGA has not received the proper configuration file header to indicate the beginning of configuration data.
- If the configuration device or external host has sent all configuration data and CONF_DONE has not gone high, ensure CONF_DONE has a pull-up to V_{CC} and that it is not grounded or driven low on your board.

Multi-Device Configuration Chains

- You should combine each device's SOF into one configuration file through the **Convert Programming Files** dialog box in the Quartus II software. For more information, refer to the *Configuration File Formats Chapter*.
- When generating the configuration file, ensure the configuration files are in the same order as the devices on the board.

Using an External Host (e.g., Microprocessor or CPLD)

- If using a Raw Binary File (**.rbf**) to configure your Altera FPGA(s), make sure the least significant bit (LSB) of each byte is sent first. If the file is sent in the wrong order, this will cause a configuration error.
- Scope the DATA and DCLK or nWS signals to check that all timing parameters are met as specified in the tables in the appropriate device family chapter.
- If using passive parallel asynchronous (PPA), make sure nRS is not left floating. This pin should be driven high if it is not used, otherwise configuration errors can occur.

Using a Configuration Device

- Make sure the configuration device has been programmed successfully. This can be done through the Quartus II programmer by performing a Verify on the configuration device. If the configuration device has not been programmed successfully, it will not configure the FPGA.
- If you notice no `DCLK` or `DATA` output from the configuration device, it is possible the configuration device is in a slave mode or idle state. When using a configuration device, the FPGA's V_{CCINT} supply must be powered up before the configuration device exits POR. If the configuration device exits POR before the FPGA is powered up, the configuration device will enter slave mode (EPC2/EPC1 device) or will enter an idle state (enhanced configuration device).
- To determine if the FPGA is flagging an error by driving the `nSTATUS` signal low or if the configuration device is flagging an error by driving the `OE` signal low, you can separate the `nSTATUS` and `OE` signals on your board (This can also be done by using a logic analyzer and calculating how many `DCLK` edges have occurred). This should only be done for debugging purposes. If you disconnect the `nSTATUS` and `OE` line, each signal must have a pull-up to V_{CC}. During configuration, if the FPGA pulls `nSTATUS` low, it has seen a CRC error in the configuration data. If the configuration device pulls `OE` low, it has sent all its configuration data and has not seen the `CONF_DONE` signal go high.
- If using an enhanced configuration device, make sure all pins are connected properly. The PGM pins should not be left floating; they should be driven to choose the specific page the configuration file resides. The `WP#` should be connected to V_{CC} to enable programming of the bottom boot block. The `BYTE#` (available in 100-pin packages) should be connected to V_{CC}, otherwise programming verification will fail; hence resulting in a configuration failure. In the 100-pin package, make sure the following pins are connected externally: `F-A0` to `C-A0`, `F-A1` to `G-A1`, `F-A15` to `C-A15`, and `F-A16` to `C-A16`.
- If using an enhanced configuration device, the external flash interface pins should be floating or tri-stated during in-system programming and during FPGA configuration. For more information, refer to *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet*.

Using JTAG Configuration

- Double-check that all JTAG pins are connected as recommended in the JTAG section. Specifically, make sure `TRST` (if available) is connected to V_{CC} and that `TCK` has a pull-down resistor.
- Double-check that all configuration pins are connected as recommended in the JTAG section. Specifically, make sure the `nCE` pin is connected to GND or driven low during JTAG programming. The `nCONFIG` pin must be tied to V_{CC} or driven high. Also, check that `CONF_DONE` is not held low by another device. This is important to remember for multi-device chains.
- Ensure the `TDO` signal drives out a high enough voltage to meet the next device's `TDI` minimum high-level input voltage (V_{IH}). If the `TDO` pin resides in an I/O bank whose V_{CCIO} is set to 3.3 V, the `TDO` pin will drive out 3.3 V.
- When designing the board, layout the TCK trace using the same techniques used to layout a clock line. Any overshoot, undershoot, ringing, or other noise can affect JTAG configuration.
- Probe the `TCK` signal to ensure it is a clean signal with no overshoot, undershoot, or ringing. For a chain of devices, you should probe each device in the chain as close to the `TCK` pin as possible. Noise on any of these pins could cause JTAG programming to fail for the entire chain.

For further help with debugging your configuration issues, visit the online configuration troubleshooter at **www.altera.com**.